Developing SQuIRE to map the landscape of interspersed repeat expression

by
Wan Rou Yang

A dissertation submitted to Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy

Baltimore, Maryland

July, 2018

Abstract

Transposable elements (TEs) are interspersed repeat sequences that make up much of the human genome. Their expression has been implicated in development and disease. However, RNA-seq of TE transcripts results in ambiguous multi-mapping reads that are difficult to quantify. Past approaches to TE RNA-seq analysis have excluded these reads, aligned the reads to interspersed repeat consensus sequences, or aggregated RNA expression to subfamilies shared by similar TE copies. Such approaches have lost either quantitative accuracy or the genomic context necessary to understand TE transcription and its effects. As a result, repetitive sequence contributions to transcriptomes are not well understood. Here, we present Software for Quantifying Interspersed Repeat Expression (SQuIRE), to date the first and only RNA-seq analysis pipeline that provides a quantitative and locus-specific picture of interspersed repeat RNA expression. We demonstrate that SQuIRE is an accurate and powerful tool that can be used for a variety of species. Using SQuIRE on a variety of cell and tissue types in human and mouse data, we found that only a small percentage of TEs are transcribed, and that differential expression of TEs includes transcription of longer TE-containing mRNAs and lncRNAs. Our findings illustrate the importance of studying TE transcription with locus-level resolution. SQuIRE can be downloaded at (github.com/wyang17/SQuIRE).

Acknowledgements

I am fortunate to have no shortage of mentors in this journey. I am thankful to Dr. Hyam Levitsky for his mentorship in tumor immunology and continued support as he moved into the world of industry. I am forever grateful to my primary mentor Dr. Kathleen Burns for her unwavering belief in me, encouragement, and excitement about my work. These two mentors have been exemplary role models in the pursuit of rigorous science, collaborative research, and translational medicine. I am also thankful to Dr. Sarah Wheelan for her guidance in the development of SQuIRE. This project would not have started without her, and this work greatly improved with her as a reader. Dr. Charles Drake's enthusiasm for my work and its implications for understanding broader disease has given me insight to how I plan to apply SQuIRE as I move forward in my career. I am thankful to Dr. Marc Halushka for pushing me to believe that the development of SQuIRE stands on its own merit, and to ensure my work is published, which has greatly advanced the work and will one day enhance my career as well. Dr. Lindsay Horvath has been a source of insights and constructive criticism that has pushed me to be a more rigorous and relevant scientist; this work is indebted to the generosity of her time and collaborative spirit.

Being the member of two labs has afforded me numerous scientific colleagues that have become great friends. I am thankful for the conversations about immunology and mouse work and fun outings with Dr. Michael Korrer, Dr. Han Hsuan Fu, Dr. Lu Qin, Dr. Deepak Kadayakkara, and Dr. Breann Yanagisawa from the Levitsky laboratory. I am also thankful for the support, help, and conference adventures with Dr. Chunhong Liu, Dr. Nemanja Rodic, Dr. Lindsay Horvath, Dr. Jane Welch, and Daniel Ardeljan. I am extremely fortunate to have mentored the amazing students Chloe Pacyna and Angela Hu in my time here and am looking forward to seeing their careers progress. I am also incredibly grateful for the technical assistance of Jie Fu and Jared Steranka.

I would not be here if not for the support and acceptance of Dr. Robert Siliciano, Sharon Welling, Martha Buntin and Bernadine Harper of the MD-PhD program. I am also extremely grateful to Dr. Andrea Cox for her support and mentorship as she assumed the role of MSTP director. I am also thankful the support of the Pathobiology Graduate program, including former director Dr. Noel Rose and current director Dr. Lee Martin, as well as administrators Stacey Morgan and Tracie McElroy.

The journey of the PhD can be incredibly isolating without the continued support from family and friends. I am thankful to my mother, Mei Hsiao and my brother Kevin Wu for making me the person I am today. I am thankful to my husband's family Janet, Robert, Matthew, Josh, Angela and Tiffany Purkeypile for their unwavering belief in me. I am grateful to the lifelong friends I have made in Tammy Guo, Claire Muerdter, Dr. Bipasha Mukherjee, Dr. Carolina Montano, Dr. Meghana Rao, Dr. Breann Yangisawa, and the members of the Association of Women Student MD-PhDs (AWSM).

As a working mother the completion of my PhD would not have been possible without my nanny Rachell Williams. Even though she is too young to understand what I do, I am grateful to my daughter, Rowan Purkeypile, for teaching me how to always make time for what's important. I strive to be someone she will be proud of one day.

Most of all I am thankful to my husband, Nathan Purkeypile, for his patience, dedication, and belief in me. We have gone on so many adventures together, and I look forward to continuing this next phase of my life with him by my side.

Table of Contents

List of Tables

List of Figures

1. Introduction

Transposable elements (TEs) are self-propagating mobile genetic elements. Their insertions have resulted in a complex distribution of interspersed repeats comprising almost half of the human genome [1,2]. TEs propagate using either DNA ('transposons') or RNA intermediates ('retrotransposons')[3,4]. Retrotransposons are further classified into Orders, namely long terminal repeats (LTR), long interspersed elements (LINEs), and short interspersed elements (SINEs)[5]. A subset of evolutionarily young subfamilies from the LINE-1 superfamily (i.e., L1PA1 or L1HS) [6], the SINE *Alu* superfamily (e.g., *Alu*Ya5, *Alu*Ya8, *Alu*Yb8, *Alu*Yb9) [7], as well as composite SVA (SINE-variable number tandem repeat (VNTR)-*Alu*) elements [8] remain retrotranspositionally active and generate new polymorphic insertions [9,10]. However, most TEs have lost the capacity for generating new insertions over their evolutionary history and are now fixed in the human population.

Even elements that have lost the potential to retrotranspose can still be transcribed from their locations in the genome. TEs are significant contributors of promoters [11–13] and *cis*-regulatory elements to the transcriptome [14–22]. Transcription of TEs has been implicated in physiological processes in development and early embryonic pluripotency [23,24]. Conversely, TE expression can also be subject to transcriptional silencing [25–29]. Loss of these regulatory mechanisms resulting in dysregulated TE expression has been associated with cancer [30–32], neurodegenerative diseases[33–37], and infertility [38–41]. However, a deeper understanding of how TE transcription impacts these biological processes has been limited by difficulties analyzing TE transcription in RNA sequencing (RNA-seq) data.

Due to the repetitive nature of TEs, short-read RNA sequences that originate from one locus can ambiguously align to many TEs sharing similar sequence dispersed throughout the genome. This problem is most significant for younger TEs; older elements have accumulated nucleotide substitutions over millions of years that can differentiate them and give rise to uniquely aligning TE

reads [42]. Because of these barriers, conventional RNA-seq analyses of TEs have either discarded multi-mapping alignments [18] or combined TE expression to the subfamily level [43–45]. Other groups have studied active LINE-1s using tailored pipelines, leveraging internal sequence variation and 3' transcription extensions into unique sequence [46–48]. However, these targeted approaches do not provide a comprehensive picture of TE expression.

To analyze global TE expression in conventional RNA-seq experiments, we have developed the Software for Quantifying Interspersed Repeat Expression (SQuIRE). SQuIRE is the first RNA-seq analysis pipeline available that quantifies TE expression at the locus level. In addition to RNA-seq providing expression estimations at the TE locus level, SQuIRE quantifies expression at the subfamily level and performs differential expression analyses on TEs and genes. We benchmark our pipeline using both simulated and experimental datasets and compare its performance against other software pipelines designed to quantify TE expression [43–45]. We demonstrate that SQuIRE provides a suite of tools to ensure the pipeline is user-friendly, reproducible, and broadly applicable.

The development of SQuIRE enabled us to analyze TE expression at the locus level in normal cells. We applied the SQuIRE pipeline to 31 primary cell lines belonging to epithelial, muscle, connective and nervous tissue types. We were able to discern the transcriptomic contexts of TE RNA expression. We determined that among the small percentage of TE insertions that are expressed, most transcribed TEs are part of longer pre-mRNA or lncRNAs transcripts. Only a small percentage are transcribed autonomously from an individual TE locus (ITL). We describe the expression patterns of ITLs across the 31 primary cell lines to illustrate the differences between the transcription of ITLs compared to that of longer transcripts. Our findings provide a deeper understanding of physiologic TE expression.

2. SQuIRE: Software for Quantifying Interspersed Repeat Expression

2.1 SQuIRE Overview

SQuIRE provides a suite of tools for analyzing transposable element (TE) expression in RNA-seq data (Fig. 1). SQuIRE's tools can be organized into four stages**:** *1) Preparation, 2) Quantification, 3) Analysis* and *4) Follow-up***.** In the *Preparation* stage, **Fetch** downloads requisite annotation files for any species with assembled genomes available on University of California Santa Cruz (UCSC) Genome Browser [49]. These annotation files include RefSeq [50] gene information in BED and GTF format, and RepeatMasker [51] TE information in a custom format. **Fetch** also creates an index for the aligner STAR [52] from chromosome FASTA files. **Clean** reformats TE annotation information from RepeatMasker into a BED file for downstream analyses. The tools in the *Preparation* stage only need to be run once per genome build. The *Quantification* stage includes the alignment step **Map** and RNA-seq quantification step **Count**. **Map** aligns RNA-seq data using the STAR aligner with parameters tailored to TEs that allow for multi-mapping reads and discordant alignments. It produces a BAM file. **Count** quantifies TE expression using a SQuIRE-specific algorithm that incorporates both unique and multi-mapping reads. It outputs read counts and fragments per kilobase transcript per million reads (fpkm) for each TE locus, and aggregates TE counts and fpkm for TE subfamilies into a separate file. **Count** also quantifies annotated RefSeq gene expression with the transcript assembler StringTie [53] to output annotated gene expression as fpkm in a GTF file, and as counts in a count table file. In the *Analysis* stage, **Call** performs differential expression analysis for TEs and RefSeq genes with the Bioconductor package DESeq2 [54,55]. To allow users to visualize alignments to TEs of interest visualized by the Integrative Genomics Viewer (IGV) [56] or UCSC Genome Browser, the *Follow-up* stage tool **Draw** creates bedgraphs for each sample. **Seek** retrieves sequences for genomic

coordinates supplied by the user in FASTA format. We describe further details of the SQuIRE pipeline in Methods.

SQuIRE's **Count** algorithm addresses a fundamental issue with quantifying reads mapping to TEs: shared sequence identity between TEs from the same subfamily and even superfamily. When a read fragment originating from these non-unique regions is aligned back to the genome, the read may ambiguously map to multiple loci ("multi-mapped reads"). This is not a major problem for older elements that have acquired relatively many nucleotide substitutions, and thus give rise to primarily uniquely aligning reads ("unique reads"). However, TEs from recent genomic insertions that have high sequence similarity to other loci may have few distinguishing nucleotides. Among elements of approximately the same age, relatively shorter TEs also have fewer sequences unique to a locus. Thus, discarding or misattributing multi-mapped reads can result in underestimation of TE expression.

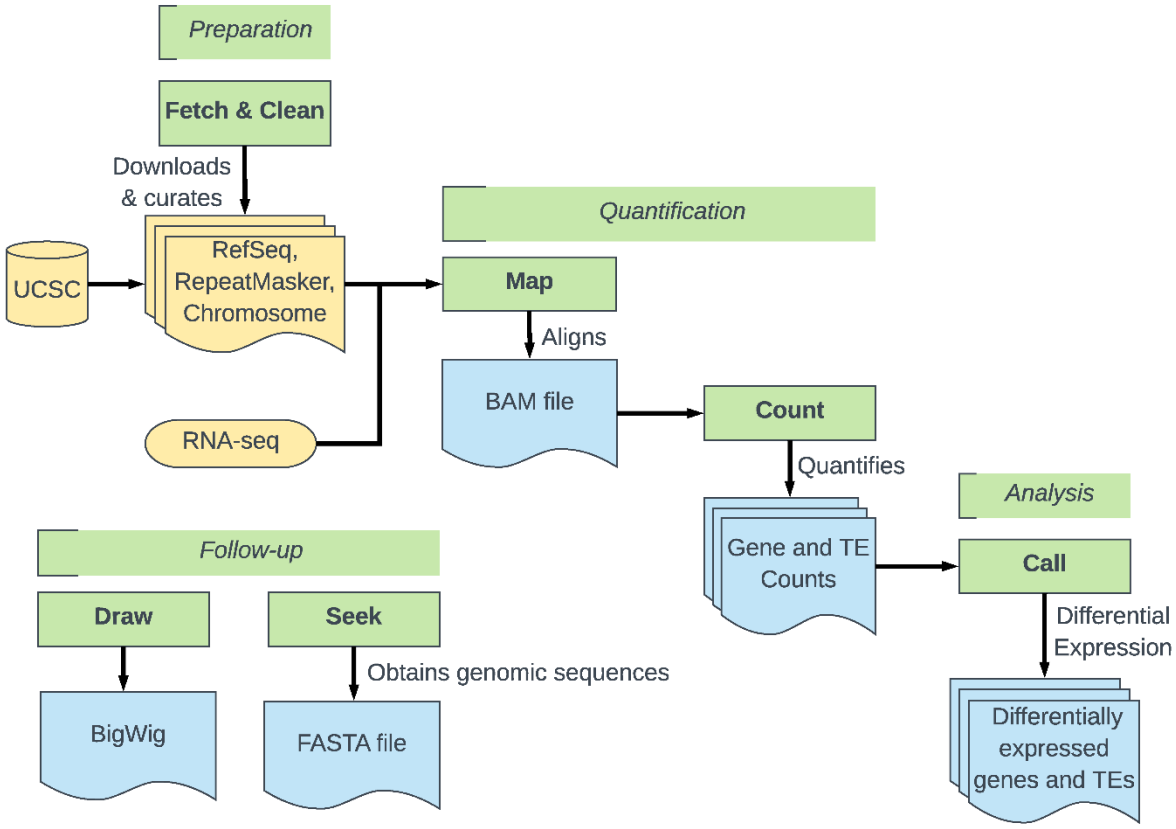


Figure 1. Schematic overview of SQuIRE pipeline.

Green boxes with bold text represent SQuIRE tools, with the pipeline stage (Preparation, Quantification, Analysis, and Follow-up) indicated above. Yellow represents inputs to SQuIRE. Blue represents SQuIRE outputs.

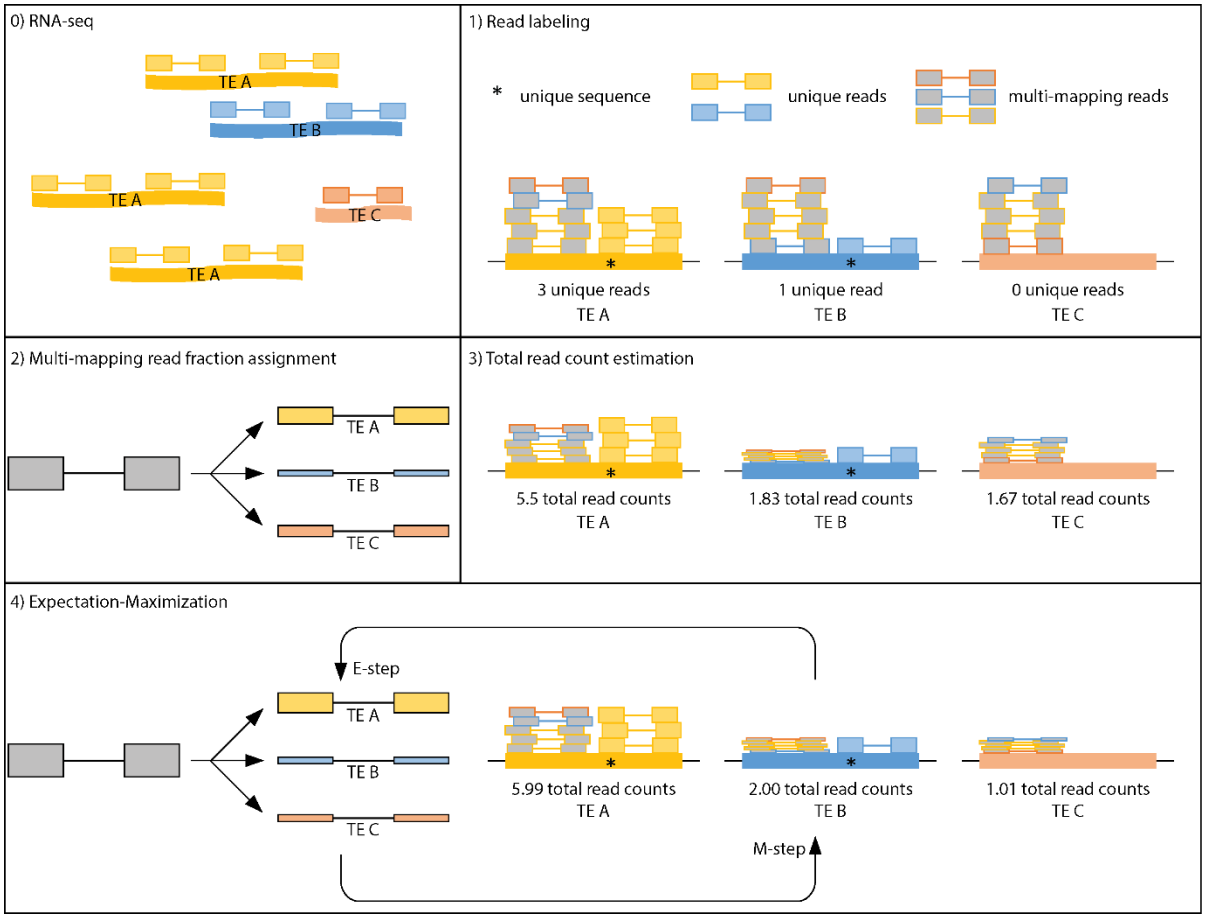


Figure 2. Schematic representation of the SQuIRE Count algorithm.

This example illustrates the quantification of RNA-seq reads from three TE transcripts with various expression levels and transcript lengths. First, ***Count*** *labels reads as unique (colored boxes) or multi-mapping (grey boxes). Uniquely mapping reads map to unique sequence in a TE (asterisks), whereas multi-mapping reads map to similar sequence shared by the three TEs in the example. Second,* ***Count*** *assigns fractions of multi-mapping reads in proportion to the normalized unique read expression of each TE. Because TE C has no uniquely aligning reads, it receives a fraction equal to 1/3, which is inversely proportional to the number of loci to which the multi-mapping read aligned. Third, the multi-mapping fractions are summed with the unique reads to give an initial total read count estimation. Finally,* ***Count*** *runs an Expectation-Maximization loop that reassigns multi-mapping read fractions for each TE (E-step), and re-estimates total read counts (M-step) until convergence*

2.2 Count Algorithm

Previous TE RNA-seq analysis pipelines have been able to quantify TE expression at subfamily-level resolution. The software RepEnrich [43] "rescued" multi-mapping reads by re-aligning them to pseudogenome assemblies of TE loci and assigning a fraction of a read inversely proportional to the number of subfamilies to which each read aligned. These multi-mapped fractions were combined with counts of unique reads aligned to each subfamily. This approach was an advance in that it used information from multi-mapped reads. However, this method results in assigning fractions that are proportional to the number of subfamilies that share the multi-mapped read's sequence, rather than each subfamily's approximate expression level. TEtranscripts [44] expanded on this rescue method by assigning an initial fractional value inversely proportional to the number of TE loci (not subfamilies) to which each read aligned. This initial fractional value was then used in an expectation-maximization (EM) algorithm, which iteratively re-distributes fractions of a multi-mapping read among loci (E-step) in proportion to their relative multi-mapped read abundance estimated from a previous step (M-step). The total of multi-mapped reads and unique reads for each loci are then summed by subfamily. However, in excluding unique reads from the EM algorithm, TEtranscripts does not incorporate empirical high-confidence data to infer TE expression levels from unique TE alignments. Furthermore, in calculating the relative expression level of multi-mapped reads, TEtranscripts normalizes read counts (c) based on annotated coordinates from RepeatMasker. Thus, when the TE transcript length (l_{TE}) is shorter than the annotated genomic length (l_a), TEtranscripts calculates a transcript coverage that is lower than the true value (ie, $\frac{c}{l_a} < \frac{c}{l_{TE}}$ when $l_{TE} < l_a$). TEtranscripts then sums the unique and multi-mapping counts for each subfamily.

In order to accurately quantify TE RNA expression at locus resolution, **Count** builds on these previous methods by leveraging unique read alignments to each TE to assign fractions of multi-mapping reads (Fig. 2). First, **Count** identifies reads that map to TEs (by at least 50% of the read length) and labels them as "unique reads" or "multi-mapped reads". Second, **Count** assigns fractions of a read to

each TE as a function of the probability that the TE gave rise to that read. Uniquely aligning reads are considered certain (i.e., probability = 100%, count = 1). **Count** initially assigns fractions of multi-mapping reads to TEs in proportion to their relative expression as indicated by unique read alignments. In doing so, **Count** also considers that TEs have varying uniquely alignable sequence lengths. To mitigate bias against the *n* number of TEs without uniquely aligning reads, these TEs receive fractions inversely proportional to the number of loci (*N)* to which each read aligned. Then **Count** assigns the remainder $(1 - \frac{n}{N})$ to the TEs with unique reads. To account for TEs that have fewer unique counts due to having less unique sequence, **Count** normalizes each unique count (C_U) to the number of individual unique read start positions, or each TE's uniquely alignable length *(L_U)*. Among all TEs to which a multi-mapping read aligned, the TEs with unique reads $(s \in T)$ are compared with each other. A fraction of a read is assigned to each TE in proportion to the contribution of the normalized unique count $(\frac{C_U}{L_U})$ to the combined normalized unique count of all of the TEs being compared $(\sum_{s \in T} \frac{Cs}{L_s})$ (Equation 1). Thus, the sum of unique counts and multi-mapped read fractions for each TE provides an initial estimate of TE read abundance based on empirically obtained unique read counts and uniquely alignable sequence.

$$f_{TE}^{r} = \frac{\frac{C_U}{L_U}}{\sum_{s \in T} \frac{Cs}{L_s}} \times (1 - \frac{n}{N}) \qquad \text{Equation 1}$$

At this point, multi-mapping reads are assigned to TEs with no unique reads based only on the numbers of valid alignments for each read. Count next refines this initial assignment by redistributing multi-mapping read fractions in proportion to estimated TE expression. To estimate expression, **Count** uses the a TE's total read count (C_{TE} = unique read counts + multi-mapped fractions from the previous step) normalized by the effective transcript length (l_{TE}): $\frac{C_{TE}}{l_{TE}}$. The effective transcript length l_{TE} is calculated as the estimated transcript length L_{TE} subtracted by the average fragment length aligned to that TE + 1, $(l_{TE} = L_{TE} - l_{avg} + 1)$, as described previously [57]. All of the TEs to which a multi-

mapping read aligned ($s \in T$) are compared with each other. A fraction of a read is assigned to each TE in proportion to the relative normalized total count ($\frac{C_{TE}}{l_{TE}}$) compared to the combined normalized total count of all of the TEs being compared ($\sum_{s \in T} \frac{T_s}{l_s}$), as shown in Equation 2. **Count** assumes this value is proportional to the probability that the TE gave rise to the multi-mapping read, and assigns that fraction of a read count to the TE. Because TEs with a count fraction of less than 1 have a low probability of giving rise to any read, those TEs are assigned a count fraction of 0. The probability that would have been assigned to the unexpressed TE then gets reassigned to the other TEs to which the read mapped.

$$f_{TE}^{r} = \frac{\frac{C_{TE}}{l_{TE}}}{\sum_{s \in T} \frac{T_s}{l_s}} \qquad \text{Equation 2}$$

After the total counts (unique and multi-mapped) of each TE are re-calculated, multi-mapped reads can be re-assigned in subsequent iterations of expectation (assigning multi-mapped read fractions to TEs) and maximization (summation of unique and multi-mapped fraction counts). These iterations can be repeated until a given iteration number set by the user or until the TE counts converge ("auto", when all of the TEs with ≥ 10 counts change by < 1%). An example of **Count** output is provided in Table 1. Further details of the **Count** algorithm are in Methods.

tx_chr	tx_start	tx_stop	TE_ID	fpkm	tx_strand	Sample	alignedsize	TE_chr	TE_start	TE_stop	TE_name	milliDiv	TE_strand	uniq_counts	tot_counts	tot_reads	score
chrX	150227860	1.5E+08	chrX|150227860|150227918|Plat_L3:CR1:LINE|224|-	4228.95	+	sample1	10355420	chrX	1.5E+08	1.5E+08	Plat_L3:CR1:LINE	224	-	74	2539.97	2541	99.96
chr4	35784285	35784363	chr4|35784285|35784363|UCON49:L2:LINE|206|-	3415.7	+	sample1	10355420	chr4	35784285	35784363	UCON49:L2:LINE	206	-	112	2758.94	2759	100
chr14	94460277	94460382	chr14|94460277|94460382|L1ME4a:L1:LINE|233|+	2698.38	-	sample1	10355420	chr14	94460277	94460382	L1ME4a:L1:LINE	233	+	36	2934	2934	100
chr13	100961881	1.01E+08	chr13|100961881|100962054|L2b:L2:LINE|283|-	2118.35	+	sample1	10355420	chr13	1.01E+08	1.01E+08	L2b:L2:LINE	283	-	132	3795	3795	100
chr22	38983462	38983650	chr22|38983462|38983650|MIR:MIR:SINE|319|+	1984.78	-	sample1	10355420	chr22	38983462	38983650	MIR:MIR:SINE	319	+	44	3864	3864	100
chr3	176423408	1.76E+08	chr3|176423458|176423572|L1M5:L1:LINE|225|+	1800.79	-	sample1	10355420	chr3	1.76E+08	1.76E+08	L1M5:L1:LINE	225	+	0	3990.67	3991	99.99

Table 1. Example output from SQuIRE Count.

tx_start = start position of left-most read aligning to TE

tx_stop = stop position of right-most read aligning to TE

TE_ID = unique ID concatenating RepeatMasker annotation (see below): coordinates, TE name, milliDiv, and annotated strand. Each TE_ID may have up to two entries if RNA-seq data is stranded, one for each transcribed strand

fpkm = fragments per kilobase transcribed length per million aligned fragments

tx strand = strand of TE transcription

alignedsize = number of fragments with valid unique or multi alignments

TE_start = annotated RepeatMasker start

TE_stop = annotated RepeatMasker stop

TE_strand = annotated RepeatMasker strand (orientation of TE insertion)

milliDiv = Base mismatches in parts per thousand (from RepeatMasker)

uniq_count = # uniquely aligning reads

tot_count = # uniquely aligning reads + sum of multimapping fractions aligned to TE

tot_reads = # multi-mapping reads aligned to TE

*score = tot_count/tot_reads * 100, which approximates how likely the TE is expressed with at least the tot_count*

2.3 Assessing Count accuracy in simulated data

To test the performance of **Count**, we simulated RNA-seq data from 100,000 randomly selected TEs from the human GRCh38/hg38 (hg38) RepeatMasker annotation. TEs were simulated with read coverages of ranging from 2-4000X and simulated counts ranging from 2-4,588. More details of the RNA-seq simulation are described in Methods. We first evaluated accuracy by how closely SQuIRE **Count** output corresponded to the simulated read counts (i.e., % Observed/Expected). However, using this calculation is not meaningful for TEs with low simulated counts: a TE with 0 counts gives an infinite value, and a reported count of 1 for a TE with 2 simulated reads gives a low 50% Observed/Expected. Thus, we were primarily interested in 'expressed' simulated TEs, considering only the 99,567 TEs with at least 10 simulated reads. Second, we evaluated SQuIRE by how often it correctly detected simulated TE expression (i.e., true positives) or misreported unexpressed TEs (i.e., false positives).

To test how well SQuIRE performed leveraging only uniquely aligning read information, we first evaluated the % Observed/Expected of TE counts with 0 E-M iterations. We found that SQuIRE accurately assigned read counts to most TEs, with a mean % Observed/Expected of 98.79%. We predicted that this accuracy would be lower for TEs with less uniquely alignable sequence. Indeed, SQuIRE was less accurate for elements with less than 10% divergence (mean of 77.35 % Observed/Expected). The most frequently retrotranspositionally active TEs (i.e., *Alu*Ya5, *Alu*Ya8, *Alu*Yb8, *Alu*Yb9, and L1HS) had counts ranging from 48-70% Observed/Expected, with a range of 79-92% Observed/Expected at the subfamily level (Table 2). This illustrates that even without the EM-algorithm, SQuIRE can distinguish expression from highly homologous TEs at the subfamily level.

		% Observed/Expected (for i E-M iterations)					
TE order	TE name	i = 0		i = auto		E-M improvement (%)	
		Locus	Subfamily	Locus	Subfamily	Locus	Subfamily
SINE	AluYa5	54.11	90.32	64.41	90.32	10.3	0
	AluYa8	69.69	79.89	85.05	88.11	15.36	8.22
	AluYb8	50.53	83.81	57.88	93.53	7.35	9.72
	AluYb9	48.47	91.6	63.25	93.94	14.78	2.34
LINE	L1HS	52.83	70.6	63.93	72.21	11.1	1.61

Table 2. % Observed/Expected before and after EM algorithm.

% of simulated reads that were reported by SQuIRE (% Observed/Expected) for frequently active human TEs at the locus and subfamily level. % Observed/Expected is improved with the use of Expectation-Maximization (EM) algorithm until convergence ("auto" number of interations) compared to no EM iterations.

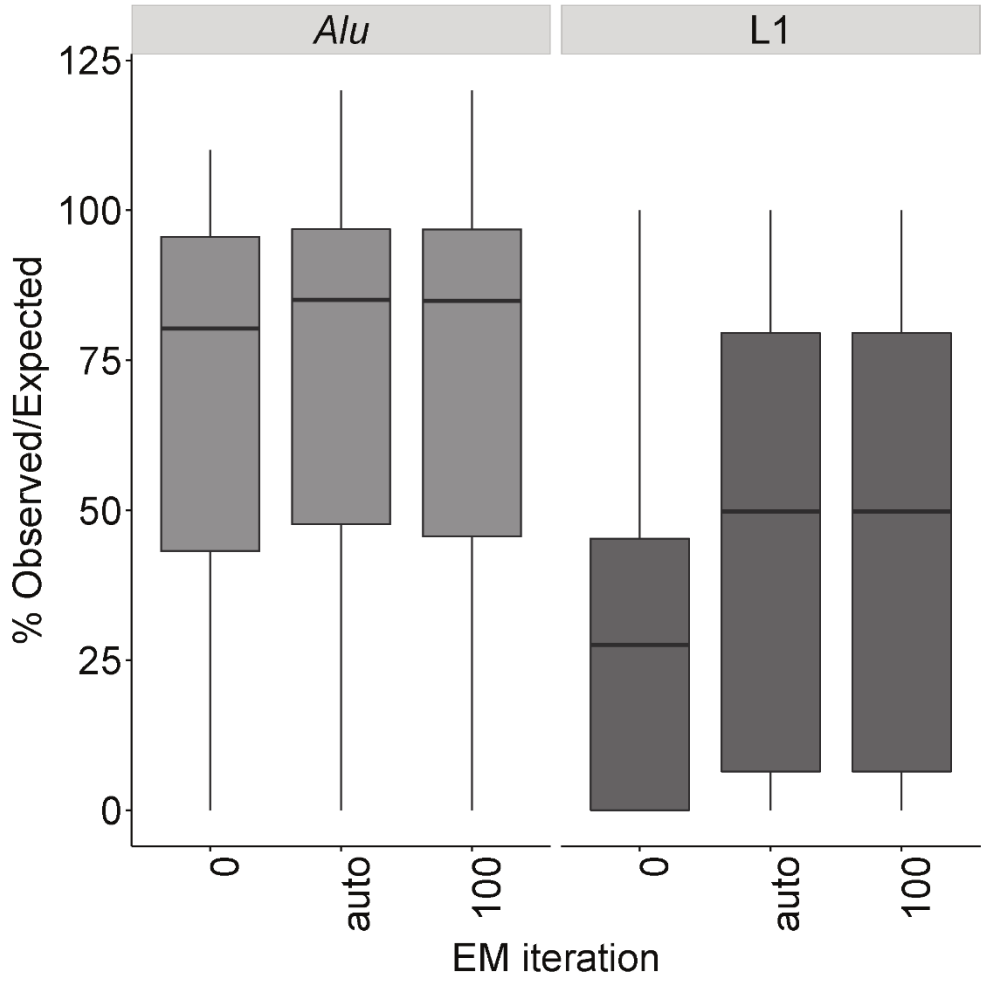


Figure 3. EM algorithm improves % Observed/Expected for young TEs.

Running EM iterations improves the % Observed/Expected for SQuIRE ***Count*** *for the frequently retrotranspositionally active Alu (AluYa5, AluYa8, AluYb8, AluYb9) and L1 (L1HS) subfamilies compared to no EM iterations (i=0), and does not degrade with increasing iterations (i=100). By default (i="auto"), SQuIRE* ***Count*** *continues the EM-algorithm until each TE with more than 10 reported read counts changes by less than 1%.*

Given the low recovery of simulated counts for younger elements when relying solely on uniquely aligning reads, we next evaluated how much adding the EM-algorithm improved **Count's** performance. We anticipated that the counts for most TEs would not change, but that younger elements with less divergence would have improved recovery of simulated reads. Indeed, the overall % Observed/Expected counts of TE loci increased only slightly by 0.14% to a total of 98.93%. However, the change in % Observed/Expected of TEs was much greater for the most homologous active elements, improving by 20.47% for young *Alu* elements and by 21.1% for L1HS loci (Fig. 4). At the subfamily level, the % Observed/Expected of active TEs was improved by 8.1% for young *Alu* elements and by 2.2% for L1HS (Table 2). Using updated transcript information in the EM-algorithm is thus particularly useful for TE biologists interested in younger elements that have previously been problematic to quantify by RNA-seq.

We also wanted to evaluate SQuIRE's ability to distinguish whether a TE is expressed or not expressed. To examine how well **Count** detected expressed TEs, we calculated the true positive rate (TPR) as the percentage of TEs with at least 10 simulated reads that SQuIRE also reported to have ≥ 10 counts. Conversely, we evaluated how often SQuIRE falsely reports TE expression by calculating the positive predictive value (PPV) as the percentage of TEs with ≥ 10 reported counts that were in fact simulated to have ≥10 reads. The true negative rate, or how often SQuIRE correctly reports that a TE is *not* expressed, is less informative for evaluating TE estimation accuracy because the number of TEs in the hg38 genome is so high (>4 million TEs) that the true negative value would outweigh the false positive value [58]. Overall, SQuIRE had both a high TPR of 98.5% and high PPV of 99.4%. These values were lower for frequently retrotranspositionally active *Alu* elements (TPR=68.75-83.33%, PPV= 64.29-100%) and L1HS elements (TPR=100%, PPV=62.86%) using only unique reads for TE expression estimation (Table 3). However, using the EM algorithm improved the TPR for *Alu* loci (TPR=85.22%-100%) by reducing false negative reports and the PPV for L1HS loci (PPV=78.57%) by reducing false positives.

		(for i E-M iterations)			
TE order	TE name	i = 0		i = auto	
		TPR	PPV	TPR	PPV
SINE	AluYa5	68.75	91.67	85.22	82.42
	AluYa8	83.33	100	100	100
	AluYb8	65.7	85.19	89.66	81.3
	AluYb9	81.82	64.29	90	64.29
LINE	L1HS	100	62.86	100	78.57

Table 3. EM improves TPR and PPV for young TEs.

True positive rate (TPR) and positive predictive value (PPV) of SQuIRE Count for recently active human TEs. The % TPR is the % of loci with ≥ 10 simulated reads which SQuIRE reports to have ≥ 10 read counts. This indicates what percentage of expressed loci are detected by SQuIRE. The %PPV is the % of loci with ≥ 10 SQuIRE reads counts that in fact have ≥ 10 simulated reads. This indicates what percentage of loci are reported to have false positive expression.

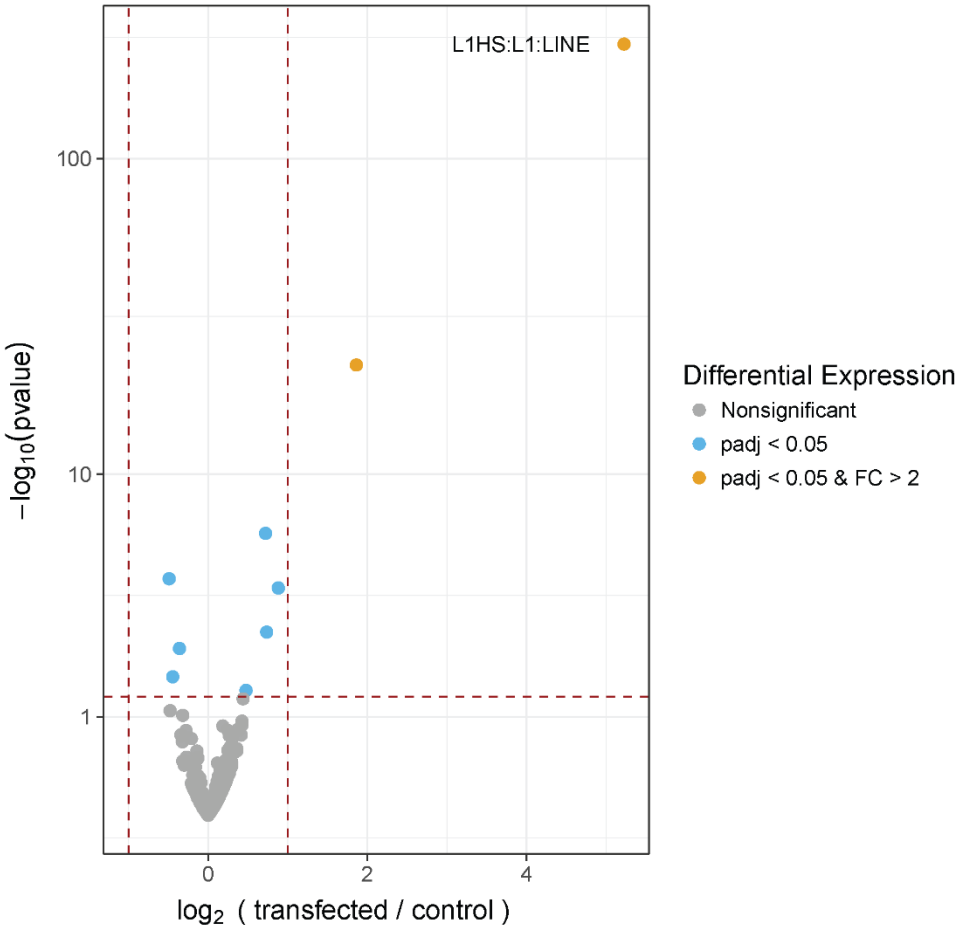


Figure 4. Volcano plot of TE subfamily expression after L1RP transfection.

The plot displays the log2 fold change comparison of mean read fragment counts between samples transfected with the L1RP or empty vector and the negative log10 of each measure's adjusted p-value.

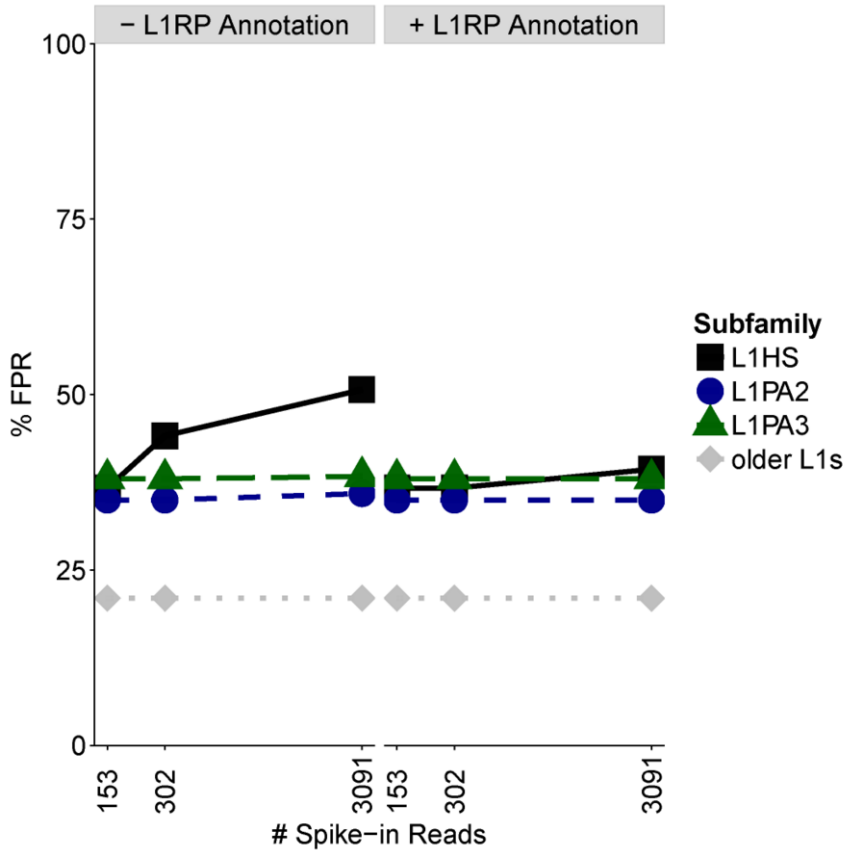


Figure 5. Non-reference annotation improves SQuIRE false positive rate.

We replicated the effects of non-reference TE expression by spiking in reads from an L1HS-expressing plasmid (L1RP) with 99% identity to the consensus sequence. We evaluated how increasing L1RP expression (153, 302, 3091 spike-in reads) affects expression estimates of reference TEs of different L1 subfamilies. False positive expression is implicated if a locus that previously had <10 reads has ≥ 10 reads after spike-in. % FPR is the percentage of loci with false positive loci relative to the total number of loci with ≥ 10 SQuIRE read counts. The FPR is robust for older L1 subfamilies with increased spike-in reads. The FPR of L1HS loci increases with greater L1RP expression without the use of L1RP annotation in the SQuIRE pipeline. The addition of L1RP annotation in a non-reference table reduces the change in false positive rate for L1HS after increasing spike-in reads.

2.4 Endogenous LINE-1 detection with Count

To assess **Count's** ability to detect endogenous LINE-1 expression using biological data, we evaluated the expression level of LINE-1 at loci previously characterized by other methods. Because genomic LINE-1 are typically 5' truncated [59], Deininger et al. performed 5' rapid amplification of cDNA ends (RACE) on cytoplasmic HEK293 RNA to enrich for full-length (6kb) LINE-1 RNA autonomously transcribed by the LINE-1 promoter sequence. They also performed RNA-seq on polyA-selected cytoplasmic HEK293 RNA to identify L1 loci that have downstream polyadenylation signal. We filtered their findings for L1 loci that had > 5 mapped RNA-seq reads from both 5'RACE and poly-A selected RNA libraries [47] to compare with SQuIRE. We then examined the expression reported by SQuIRE at these 33 loci in paired-end, total RNA from HEK293T cells (GSE113960). We found that 31 (93.4%) had > 10 SQuIRE read counts, confirming their expression. This suggests that **Count** can detect L1 expression in RNA-seq libraries that are not enriched for L1 loci.

Only a subset of the L1s evaluated by Deininger et al. belonged to L1HS, the youngest family of L1s. Because L1HS loci can be retrotranspositionally active, they can generate insertions that are polymorphic or novel compared to the reference human RepeatMasker annotation. Reads from from transcribed TE insertions that are not present in the RepeatMasker annotation can be misattributed to unexpressed, fixed TEs, which can result in "false positive" reports of expression at silent loci. To test how this affects **Count** results for other loci within the same subfamily or related subfamilies**,** we transfected HEK293T cells with an empty pCEP4 plasmid or with a plasmid containing L1RP, an L1HS with known retrotransposition activity [60,61]. The transfection of L1RP resulted in increased L1HS-aligning reads (254,681 reads) compared to L1HS loci in L1RP-negative cells (2,671 reads) (Figure 5). The differences in L1HS expression in L1RP-transfected cells was higher than what we would expect from endogenous, polymorphic insertions based on previous estimates of polymorphic and fixed L1HS expression in HEK293T cells using unique reads within 1kb downstream of L1HS loci [46]. Because Philippe et al. suggested that polymorphic L1HS insertions were transcribed at levels similar to fixed full-length L1HS loci, we sought to mimic polymorphic L1HS expression

levels more consistent with previously reported levels. To determine comparable fixed L1HS expression levels in our control HEK293T RNA-seq data, we examined the **Count** output at loci with reported expression by Phillipe et al. (145 read counts). We then downsampled the L1RP-aligning reads from L1RP transfected HEK293T cells to a similar number (153 reads). To simulate a range of polymorphic L1HS expression levels, we also downsampled RNA-seq reads that aligned to the L1RP plasmid to 2X and 20X the fixed active L1HS expression level (302 and 3,091 reads). For these downsampled reads, we identified their other, off-target alignments to the reference genome. To control for potential biological effects of L1RP transfection on TE counts, we 'spiked in' these downsampled reads from L1RP-transfected cells into RNA-seq data from HEK293T cells transfected with an empty pCEP4 plasmid. We then calculated the number of false positive L1 loci that became 'expressed' with > 10 counts after the *in silico* spike-in. We focused on the 3 youngest L1 subfamilies that share the greatest homology with the L1RP sequence (i.e., L1HS or L1PA1, L1PA2, and L1PA3) [62–64] and compared their false positive rates to older L1 loci (Fig. 6). When the alignments of 153 reads were spiked in, we found that the false positive rate (FPR) of the youngest L1 subfamilies were comparable to each other, ranging from 34-38%. However, as the spiked in alignments increased to 302 and 3091 reads, the FPR increased for L1HS to 50.68% but not the other subfamilies. This indicates that polymorphic L1HS expression primarily affects the alignments to L1HS loci, and not the loci of closely related subfamilies.

L1-mapping methods [65–68] and TE insertion detection software for whole genome sequencing [9,69–73] can identify locations of non-reference TE insertions. Validating these insertions by PCR and Sanger sequencing can provide not only unique sequence flanking the insertion but potentially also the TE sequence. Users can input a custom table to SQuIRE **Map** and **Clean** (Table 5) to add non-reference TEs and their flanking sequence to the alignment index and RepeatMasker BED file. We evaluated how incorporating the non-reference table containing information about the L1RP plasmid affected the FPR in HEK293T cell data. We found that the FPR for L1HS only increased

from 36.67% with 153 reads spiked in to 39.34% with 3091 reads spiked in. Thus, adding L1RP information improved **Count's** accuracy at higher L1RP *in silico* expression levels.

Chromosome/ Vector	Insertion_start	Insertion_stop	Strand	Subfamily:Family: Order	Insertion_Type: Polymorphism, Novel,Plasmid, Transgene	Left-Flank_Seq	Right-Flank_Seq	TE_Seq
DA_L1RP	70	6087	+	L1HS:L1:LINE	Plasmid	CGTTTAGTG AACCGTCAG ATCTCTAGA AGCTGGGTA CCAGCTGCT AGCAAGCTT GCTAGCGGC CGCGGGG	ATCCAGACATG ATAAGATACAT TGATGAGTTTG GACAAACCAC AACTAGAATGC AGTGAAAAAA ATGCTTTATTT GTGAAATTTGT GATGCTATTGC TTTATTTGTAA CCATTATAAGC TGCAATAAACA AGTTAACAACA ACAATTGCATT CATTTTATGTT TCAGGTTCAGG GGGAGGTGTG GGAGGTTTTTT AAAGCAAGTA AAACC	GGAGGAGCCAAGATGG CCGAATAGGAACAGCT CCGGTCTACAGCTCCCA GCGTGAGCGACGCAGA AGACGGTGATTTCTGC ATTTCCATCTGAGGTAC CGGGTTCATCTCACTAG GGAGTGCCAGACAGTG GGCGCAGGCCAGTGTG TGTGCGCACCGTGCGC GAGCCGAAGCAGGGCG AGGCATTGCCTCACCTG GGAAGCGCAAGGGGTC AGGGAGTTCCCTTTCCG AGTCAAAGAAAGGGGT GACGGACGCACCTGGA AAATCGGGTCACTCCC ACCCGAATATTGCGCTT TTCAGACCGGCTTAAG AAACGGCGCACCACGA GACTATATCCCGCACCT GGCTCGGAGGGTCCTA CGCCCACGGAATCTCG CTGATTGCTAGCACAG CAGTCTGAGATCAAAC TGCAAGGCGGCAACGA GGCTGGGGGAGGGGCG CCCGCCATTGCCCAGG …

Table 4. Non-reference table used to add L1RP plasmid sequence for TE read alignment and count estimation.

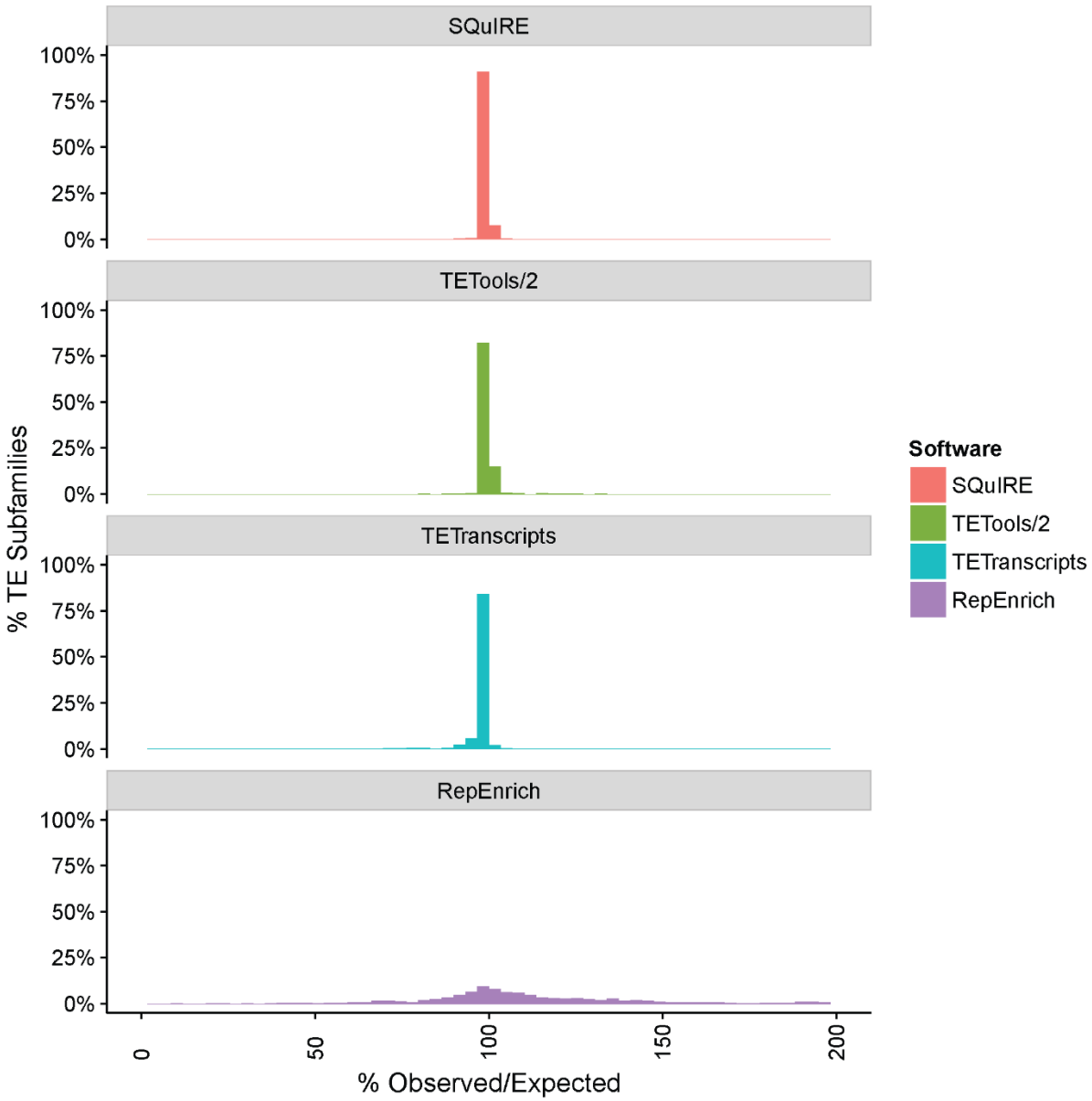


Figure 6. Comparison of TE RNA-seq tools at the subfamily level for simulated data.

Histogram of % TE subfamilies for each percentage of reported over simulated counts. SQuIRE has the tallest and narrowest peak near 100% Observed/Expected, indicating the it is correctly attributing simulated reads to the greatest number of subfamilies. Because TETools outputs in reads rather than fragments, its output is twice that of the other software.

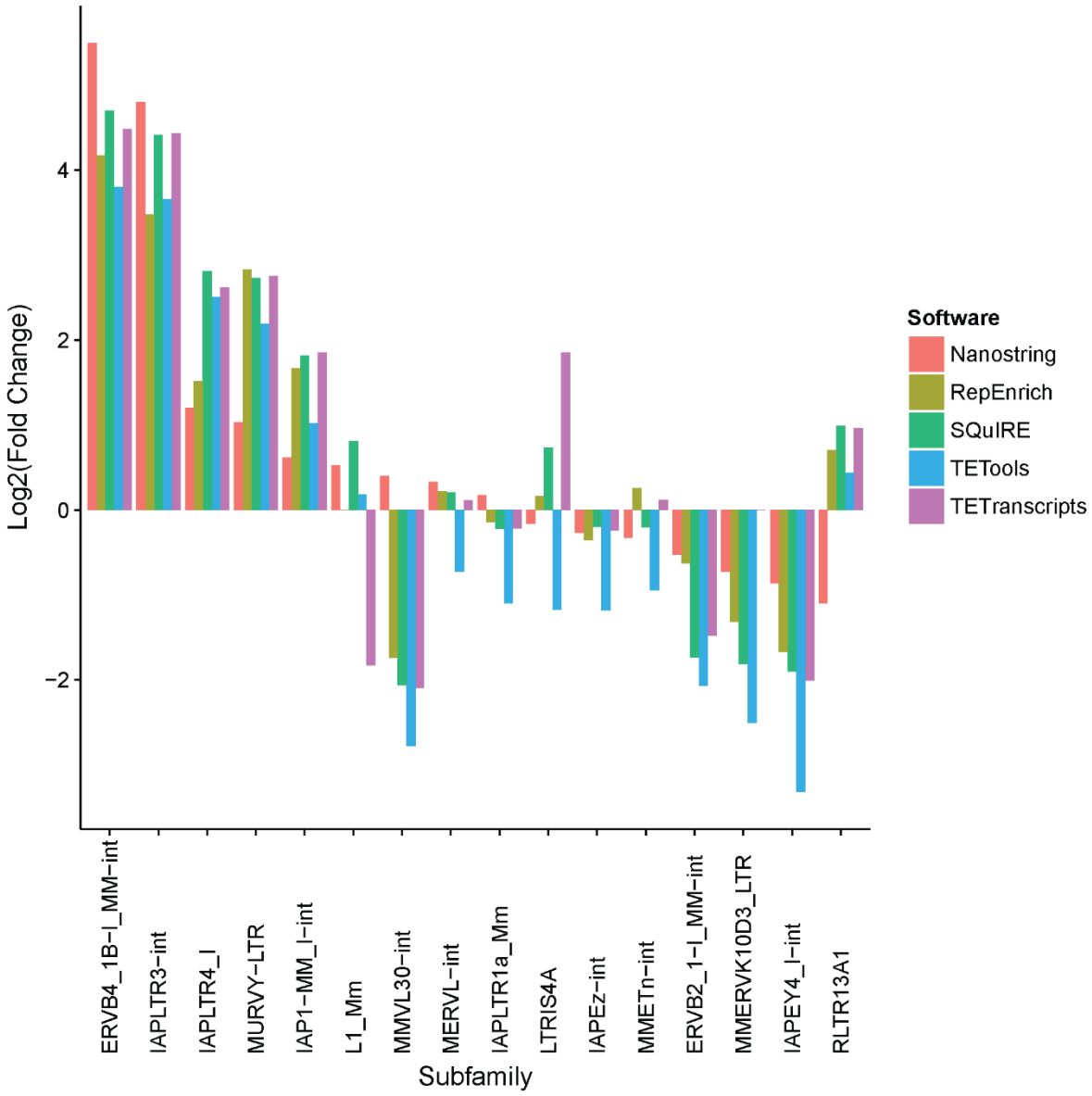


Figure 7. Bar plot comparison of TE RNA-seq tools compared to Nanostring data at the subfamily level.

Y-axis represents log2 fold changes of subfamily expression in testis compared to pooled somatic tissues (brain, heart, kidney, and liver).

2.5 Comparison to other software

Currently published TE analysis software include RepEnrich, TEtranscripts, and TETools [43–45]. Because none of these programs is capable of reporting TE locus expression, we performed comparisons with SQuIRE with aggregated subfamily estimates. We used the simulated hg38 TE data described above to compare the recovery of simulated reads to the correct subfamily among TE quantification software (i.e., % Observed/Expected). For mapping, we ran each software's recommended aligner: STAR (used by SQuIRE and TEtranscripts), Bowtie 2 (used by TETools), and Bowtie 1 (used by RepEnrich). We found that SQuIRE (99.86% ±1.46 %), TETools (100.14 ± 2.21%), and TEtranscripts (95.89 ± 16.41%) had comparable % Observed/Expected rates (Fig. 7). In contrast, RepEnrich (108.77 ± 40.67%) reported lower counts than expected for most TEs. This is likely attributable to RepEnrich's recommended use of Bowtie 1, which discards discordant reads and limits the number of attempts to align both paired-end mates to repetitive regions. To support this, we compared how often each aligner mapped a uniquely aligning simulated read to the correct location. We indeed found that Bowtie 1 failed to report unique reads more often in a paired-end library compared to single-end (Table 6).

To compare SQuIRE to other TE analysis tools with biological data, we ran each pipeline on publically available adult C57Bl/6 mouse tissue RNA-seq data [74] using GRCm38/mm10 (mm10) TE annotation. We compared the expression of subfamilies in testis compared to pooled data from brain, heart, kidney, and liver tissues. To independently evaluate the fold-changes of TE RNA between testis and somatic tissues, we also used our previously published adult C57Bl/6 mouse Nanostring results [75]. Unlike RNA-seq analysis, which infers transcript levels by counting reads, Nanostring uses uniquely mapping probes to capture and count RNA molecules. We compared the Nanostring $\log_2$ fold changes ($\log_2$FC) of TE subfamily expression in testis and pooled somatic tissue to the $\log_2$FC values found by SQuIRE, RepEnrich, TEtranscripts, and TETools (Fig. 8). Because the Nanostring probes were designed against TE consensus sequences, we do not expect exact correspondence with the RNA-seq analysis tools. We observe instances in which all TE RNA-seq

tools report contrasting results from the Nanostring output (MMVL30, IAPLTR1a_Mm, RLTR13A1). Thus in addition to comparing each pipeline with Nanostring, we also evaluated when a result deviated from the other TE RNA-seq analysis pipelines. RepEnrich failed to detect differential expression for the L1_mus_musculus subfamily (L1_Mm), and reported a direction of log2FC for the MMETn subfamily that contrasted from Nanostring. TEtranscripts similarly failed to detect differential expression of MMERVK10D3 subfamily that Nanostring and the other pipelines reported, and reported different log2FC from Nanostring, SQuIRE and TETools for L1Mm. TETools deviated from Nanostring and the other RNA-seq pipelines for the MERVL subfamily, reporting decreased expression in testis while the other methods reported upregulation. SQuIRE is the only RNA-seq pipeline that corresponded with at least two other methods for all of the subfamilies analyzed by Nanostring, suggesting that its results were the most reliable.

Aligner	Library	Library (Reads)	TP	FP	TPR	PPV
Bowtie1	Single-end	4668185	4280536	143	91.7	100
Bowtie1	Paired-end	9336370	5074922	53991	54.36	98.95
Bowtie2	Single-end	4668185	3487593	3187	74.71	99.91
Bowtie2	Paired-end	9336370	8260128	7620	88.47	99.91
STAR	Single-end	4668185	4469031	15302	95.73	99.66
STAR	Paired-end	9336370	9107935	20113	97.55	99.78

Table 5. Bowtie1 poorly detects uniquely aligning reads in paired-end libraries.

True positive rate (TPR) and positive predictive value (PPV) of identifying uniquely aligning reads with different aligners. TPR=% true positive uniquely aligning reads to total reads in the library. PPV=% true positive uniquely aligning reads to total reported uniquely aligning reads.

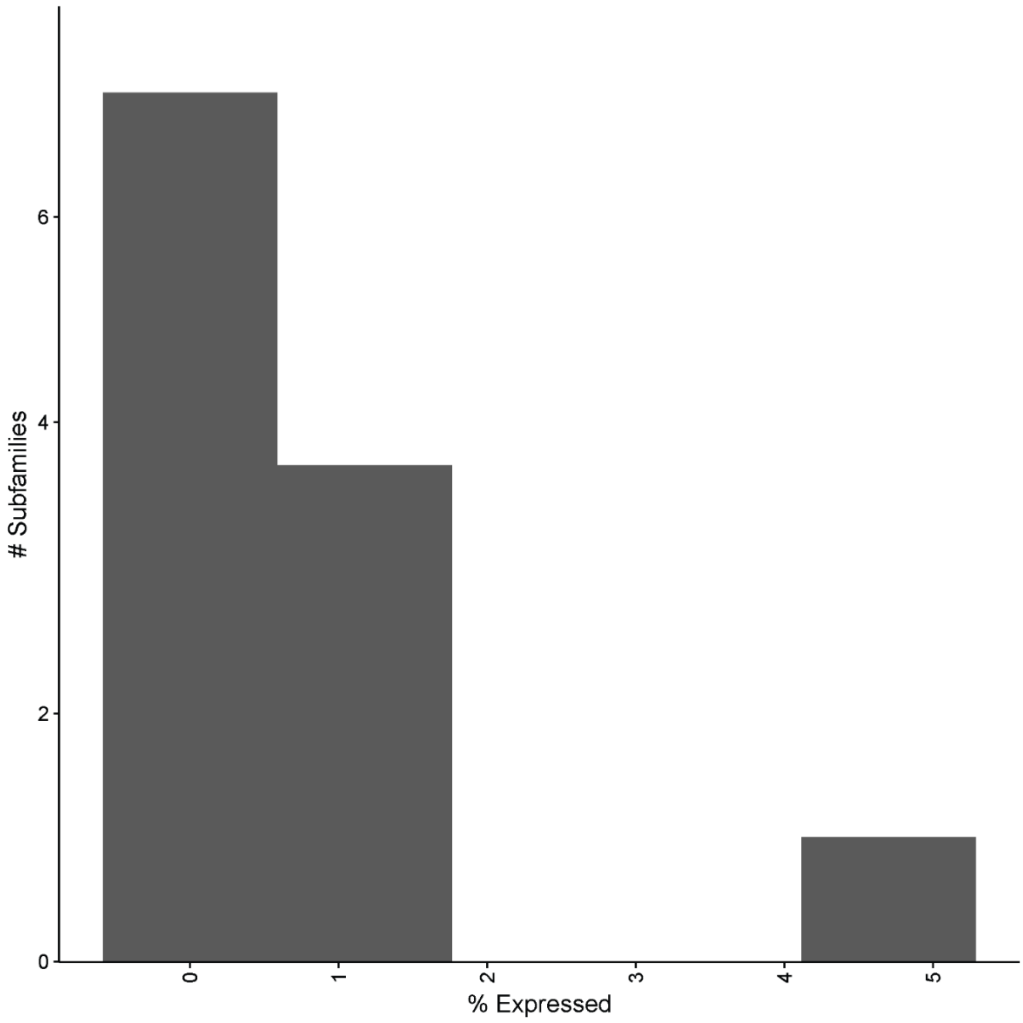


Figure 8. A small percentage of TE loci are expressed.

Histogram showing distribution of percent loci expression for TE subfamilies (among the 16 subfamilies analyzed in the previous figure). X-axis represents percentage of loci expressed.. Y-axis represents number of subfamilies). Most TE subfamilies have only 1-2% of subfamilies expressed, all TE subfamilies have 5% or fewer of their loci expressed. This information is lost when TE expression analysis is done only at the subfamily level.

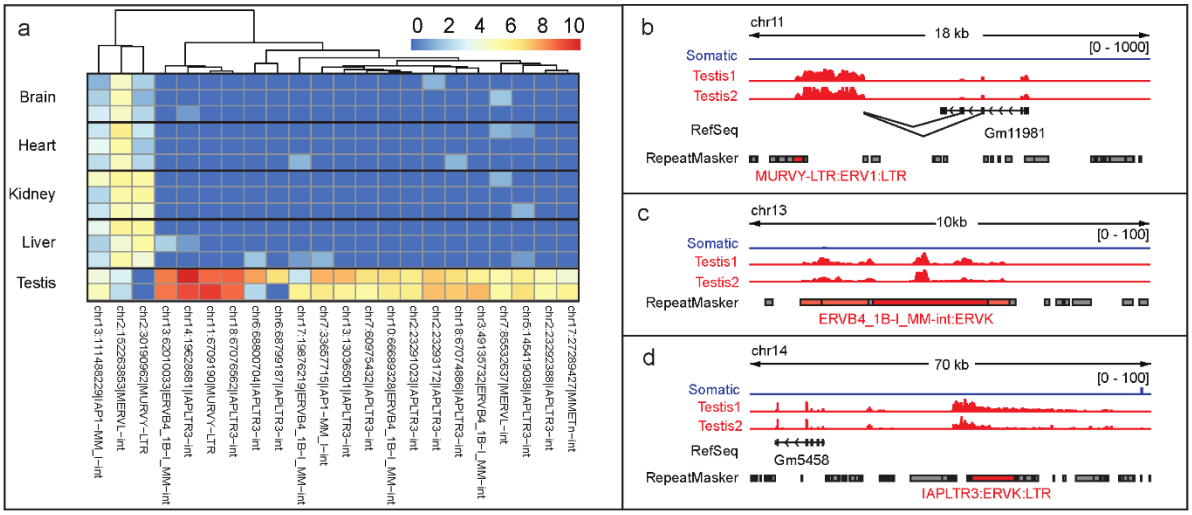


Figure 9. Differentially expressed TEs are transcribed as part of different transcript types.

a. The X-axis represents replicates of somatic and testis tissue samples from adult C57Bl/6 mouse. The Y-axis represents differentially expressed TE loci. The heatmap colors represent the log2 of total read counts +1 for each TE locus. b-d. Examples of intergenic TE loci differentially expressed in testis compared to somatic tissues. Tracks from brain, heart, kidney and liver replicates were collapsed into a single track. The scales of count expression are shown in brackets. The RefSeq track represents annotated genes. The RepeatMasker track represents transposable elements annotated in the reference genome. Transposable elements colored in red belong to the subfamily indicated; dark red indicates that that RepeatMasker entry meets significant differential expression thresholds (log2FC > 2, padj < 0.05).

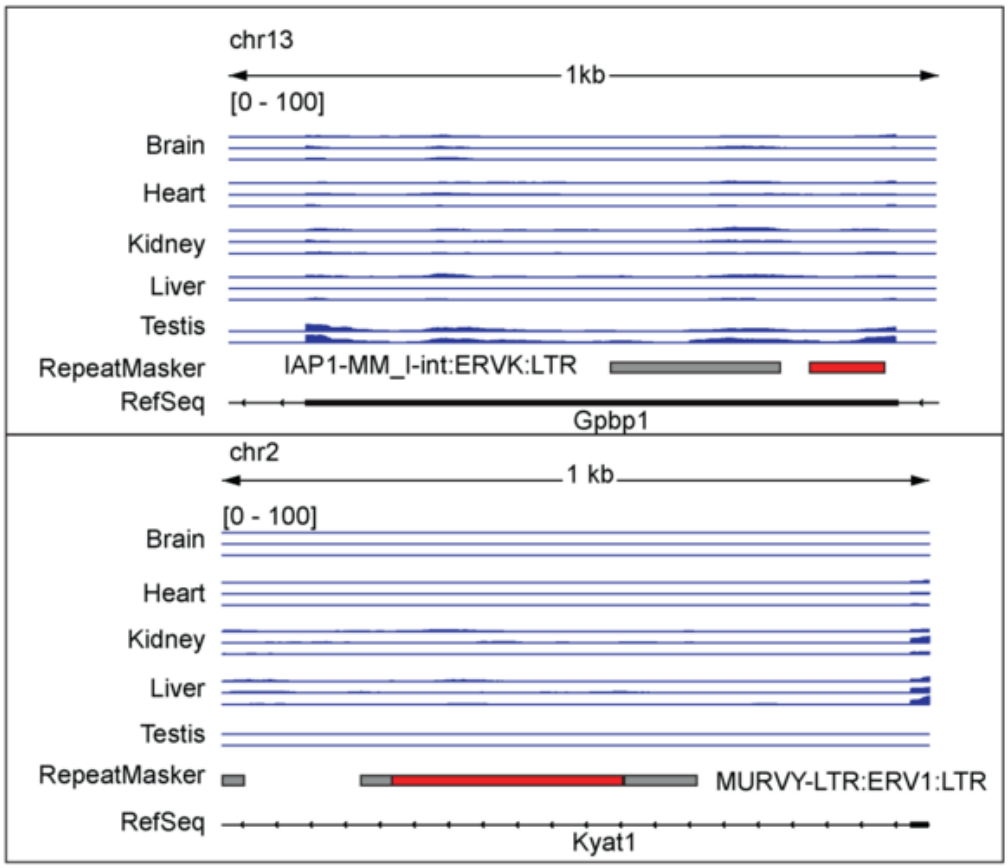


Figure 10. Examples of intragenic TE loci differentially expressed in somatic tissues compared to testis.

Replicates from brain, heart, kidney and liver are grouped in adjacent tracks. The scales of count expression are shown in brackets. The RepeatMasker track represents TEs annotated in the reference genome.The RefSeq track represents annotated genes. Transposable elements colored in red belong to the subfamily indicated; dark red indicates that the TE meets significant differential expression thresholds (log2FC > 2, padj < 0.05).

2.6 Locus-level TE expression analysis

With SQuIRE, we can closely examine the mouse RNA-seq data at the locus level. For the 16 subfamilies analyzed by Nanostring and the TE analysis tools, using SQuIRE we found that the reported subfamily-level expression was due to expression from fewer than 7% of each subfamily's loci (Supplementary Figure S5). While most subfamilies studied by Nanostring have only 1-4 significantly differentially expressed loci (log2FC >1, padj < 0.05), the IAPLTR3 subfamily has 11 loci that are all differentially expressed in testis compared to somatic tissues (Figure 5A). To test whether this was an enrichment relative to the representation of IAPLTR3 in the mouse genome, we performed a Fisher's exact test and found that IAPLTR3 loci were 10-fold more likely than expected to be differentially expressed in testis (OR: 10.56, 95% CI: 5.25-18.97, p-value < 1.61 e-08). ERVB4-1B, another LTR retrotransposon that exhibited high fold change by Nanostring, was not similarly enriched among differentially expressed TE loci. In addition to a more careful analysis of which loci are transcribed, SQuIRE enables a closer look at TE transcript structure. In examining the TE loci with the greatest differential expression in testis, we found that the transcription of the ERVB4-1B locus on chr13 did not extend beyond annotations for that element (Figure 5B). On the other hand, the IAPLTR3 loci on chr14 (Figure 5C) and chr18 are part of longer transcripts that initiate outside of the annotated TE. Altogether, this suggests while a subset of TEs may be regulated by shared TE sequence, most differential expression of TEs is locus-specific with varying transcript structures, a finding that was not evident until analysis at the locus level using SQuIRE.

To further investigate the interplay between genomic context and TE subfamily, we identified the closest genes to differentially expressed TE loci. We found a cluster of 3 loci exhibiting broad expression across somatic tissues from the IAP1, MERVL, and MURVY LTR retrotransposon subfamilies. When we examined the genomic context of these 3 loci, we found that all were located within genes with known broad tissue expression (*Gpbp1*, *Csnk2a1*, *Kyat1,* respectively) [76], with examples shown in Supplementary Figure S6. Another locus from the MURVY subfamily is in a

cluster of TEs exhibiting high testis-restricted expression. In examining the transcript overlapping the MURVY locus, we see that the transcript initiates outside of the locus and find that the transcript is an alternative splicing isoform with splice donors from the third and fourth exons of a gene ~5kb away (Figure 5D). The gene, *Gm11981*, is a long noncoding RNA (lncRNA) known to exhibit testis-restricted expression [76]. The different MURVY-containing transcript types illustrate how TE transcription can vary across loci from the same subfamily. Altogether, these findings would be lost without the use of SQuIRE to analyze TE transcription at the locus level.

2.7 Benchmarking for SQuIRE's Memory Usage and Running Time

To benchmark SQuIRE's memory usage and running time for RNA-seq data of different sequencing library sizes, we subset HEK293T cell line RNA-seq data with a mean of 32, 65, and 98 million reads. We evaluated the speed and memory performance of each *Quantification* and *Analysis* stage tool for each sequencing depth (Fig. 12) using 8 parallel threads and 64 Gb of available memory. We found that RNA-seq library size had the greatest effect on **Count**, taking 8.6 hours to complete the 3-lane library compared to 2.4 hours for the 1 lane library. The other tools took much less time and were less affected by sequencing depth. **Map** took 1-2 hours for the different libraries. **Call** running time was also independent of library size, but it was greater when including all TE counts (10 minutes) compared to subfamily counts (2 minutes). We found that the memory usage of each tool was largely independent of sequencing depth, taking between 39-40 Gb of Memory for **Map**, 30-32 Gb for **Count,** and 7-8 Gb for **Call**.

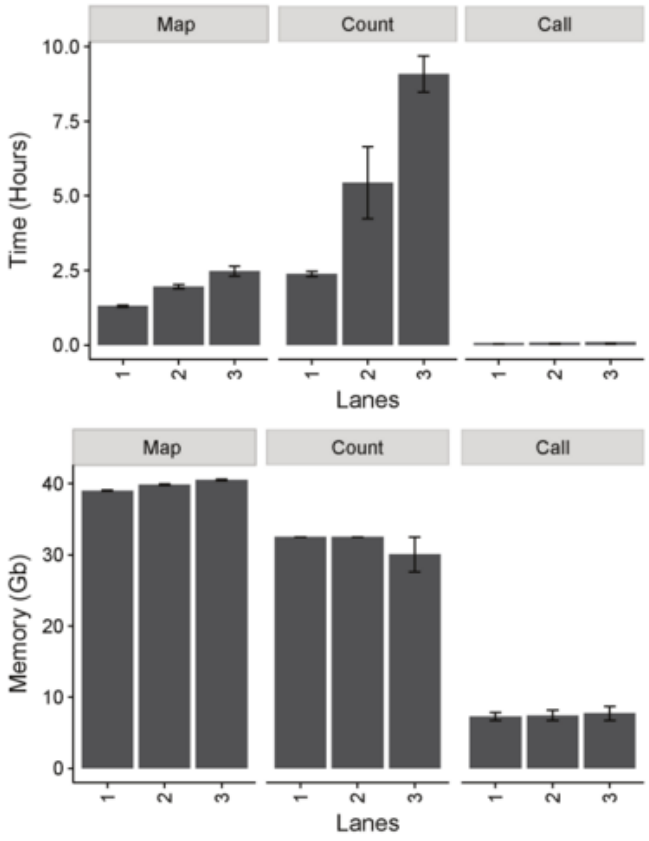


Figure 11. SQuIRE Benchmarking.

Usage data for the main modules of SQuIRE. Time (Hours) and Memory for SQuIRE ***Count, Map*** *and* ***Call****. Mean library sizes for RNA seq data were 1 lane= 32,912,528 reads, 2 lanes= 65,573,850 reads, 3 lanes= 98,757,439 reads.*

	SQuIRE	RepEnrich	TEtranscripts	TETools
Provides Locus-level TE RNA quantification	**YES**	--	--	--
Provides TE transcript strand and length	**YES**	--	--	--
Copy-and-paste installation	**YES**	--	--	--
Provides prerequisite annotation files for any species	**YES**	**--**	**--**	--
Can incorporate non-reference TEs	**YES**	**--**	**--**	**YES**
Performs alignment	**YES** – uses STAR	Recommends Bowtie 1	Recommends STAR	**YES** – uses Bowtie 1 or Bowtie 2
Uses genome for alignment	**YES**	**YES** - Genome + TE pseudogenome	**YES**	**--**
Provides gene expression quantification	**YES**	--	**YES**	**--**
Performs differential expression	**YES**	--	**YES**	**YES**

Table 6. Feature comparison of RNA-seq Analysis tools for TEs

2.8 Implementation

Our efforts at making SQuIRE easy to use has resulted in multiple features in addition to its ability to provide locus-level TE quantification (Table 7). To set up SQuIRE involves a simple installation process in which the user can copy and paste lines of code, which includes instructions for setting up prerequisite software. In addition, SQuIRE is the only program that downloads reference annotation for assembled genomes available on UCSC, allowing it to be easily adaptable to a variety of species. For genomes from non-model organisms or organism strains with high divergence from the reference annotation, SQuIRE can also use RepeatMasker software output for even wider compatibility. To ensure that the pipeline is streamlined and that the outputs are reproducible, SQuIRE also implements alignment and differential expression for the user. In making SQuIRE as user-friendly as possible, we intend to improve the reproducibility of bioinformatics in the TE field.

2.9 Discussion

We have developed Software for Quantifying Interspersed Repeat Expression (SQuIRE) to characterize TE expression using RNA-seq data. TEs are highly repeated in the genome, which can pose challenges for mapping reads unambiguously to specific transcribed loci. SQuIRE is the first RNA-seq analysis software that provides locus-specific TE expression quantification while also outputting subfamily-level expression estimates (Table 1). Our approach maximally uses information from RNAseq studies by incorporating unambiguously mapping reads as well as ambiguously mapping reads, optimally adjudicating alignments of the latter using an Expectation-Maximization (EM) algorithm. SQuIRE additionally provides empiric information on the structure of each TE transcript rather than relying on TE annotations, recognizing that TE transcripts can be shorter or longer, and sense or antisense compared to the genomic TE. We have shown that SQuIRE correctly attributes a high percentage of reads originating from TEs using simulated data. Although this percentage is lower for frequently retrotranspositionally active, less divergent TEs (e.g., *Alu*Ya5, *Alu*Ya8, *Alu*Yb8, *Alu*Yb9, L1HS), we found that implementation of the EM algorithm [44,77] improves accuracy and lowers both false positive and false negative calls of whether a TE locus is

expressed. This finding also holds in biological settings, where SQuIRE is able to correctly identify instances of full-length L1 expression in total RNA RNA-seq data from cell lines wherein previous studies had identified these loci using a combination of 5'RACE and 3' primer extension methods [47]. This confirms that SQuIRE can detect the expression of TEs in the reference genome that have in the past been problematic for global TE RNA expression analysis.

The ongoing activity of TEs also results in a significant number of mobile element insertion variants (MEI) [9,72,78]. Numerous commonly occurring structural variants owed to retrotransposition are missing in reference genome assemblies. SQuIRE provides users with two options to query transcription of these repeats. First, SQuIRE can detect transcription of polymorphic elements at the subfamily level. We have shown that SQuIRE can detect expression of the L1HS subfamily when we express an ectopic sequence. It maintains a low false positive rate of misattributing these reads to endogenous L1HS loci. Thus, SQuIRE can be useful for detecting altered regulation of young TE subfamilies even when specific loci that are expressed are unknown. Secondly, SQuIRE can directly use sequences of known, non-reference TE insertion polymorphisms to detect locus-specific expression when these are supplied as a supplement to the reference build. For example, in the human genome, L1HS element sites and sequences can be obtained by targeted TE insertion mapping [65–68] or whole genome sequencing [69–71,73]. Polymorphic TE insertions have been reported to databases such as euL1db [79], dbRIP [80] and by large studies like the 1000 Genomes Project [72]. Using SQuIRE to detect expression of user-provided, non-reference TE sequences at these loci may be a useful feature for understanding functional consequences of these insertion variants [81].

Finally, for older, retrotranspositionally inactive genomic repeats, SQuIRE very accurately assesses expression. These older elements represent the vast majority of TE loci in the human genome (>96.7%). For all TEs, SQuIRE provides the convenience of differential TE expression analysis with both locus-specific and subfamily-aggregated outputs.

The SQuIRE algorithm builds on strategies used by previous TE analysis software in line. Here, we show that SQuIRE provides additional features and improves on the accuracy of these methods, as assessed using both simulated reads and orthogonal approaches to measure $\log_2$ fold changes in mouse tissue comparisons. Our findings suggest that important biologic insights can be gained by examining TE transcription at the locus level.

To date, locus-specific studies of TE expression and activity have mostly focused on identifying transcriptionally and retrotranspositionally active L1s in the human genome [46–48,78,82–84]. These studies have shown that rare, individual loci, widely distributed in the genome generate transcripts. In applying SQuIRE to study locus-specific TE expression genome-wide in mouse tissues, we can see that this paradigm is not unique to L1s or humans. It seems a limited subset of TE loci are transcribed with complex patterns of tissue-specific expression. Furthermore, we found that the tissue expression patterns of TE loci reflect a variety of transcriptome contexts: broadly expressed mRNA transcripts, tissue-specific lncRNAs, and authentic TE 'unit' transcripts. How these TEs may affect gene regulation or biological processes remain open questions. Genome-wide analyses of TEs have indicated roles for *cis*-acting elements on transcriptional regulation [11,15,85,86], transcript splicing, and RNA function [25,87–89]. In providing locus-level TE transcript estimations, we expect SQuIRE will enable studies that dissect the regulatory impacts of TE and gene expression.

2.10 Methods

Software and Implementation

SQuIRE was written in Python 2 and tested with the following specific versions of software: STAR 2.5.3a [52], BEDtools 2.25.0 [90], SAMtools 1.8 [91], StringTie 1.3.3b [53], DESeq2 1.16.1 [54], R 3.4.1 [92], and Python 2.7.9. Details of the software parameters implemented in the SQuIRE pipeline are described in Supplemental Methods. SQuIRE was developed for UNIX environments. We provide step-by-step instructions on our README to use the package manager Conda (conda.io) to download the correct versions of prerequisite software for SQuIRE (e.g., Python, R [92], STAR,

BEDTools, StringTie, SAMtools, DESeq2). The README also instructs users how to create a non-reference table with the exogenous or polymorphic TE sequences and coordinates that they would like to add to the reference genome. Bash scripts to run each tool in the SQuIRE pipeline are also available on the website. Users can fill in crucial experiment information (raw data, read length, paired, strandedness, genome build, sample name and experimental design) into the "arguments.sh" file, which the other scripts reference to run each step with the correct parameters.

RNA-seq simulation

We randomly selected 100,000 TEs from the hg38 Repeatmasker annotation downloaded by **Fetch**. We limited our list of potential TEs to those included in TEtranscripts [44] and RepEnrich [43] to enable comparisons between these different programs. Using the selected TE coordinates we generated a BED file using **Clean** and obtained FASTA sequences using **Seek.** From these TE sequences, we used the Polyester package from Bioconductor (R version 3.4.1, Huber et al. 2015) [55] to simulate 100bp, paired-end, stranded RNA-seq reads with normally distributed fragment lengths around a mean of 250bp. We simulated a uniformly distributed sequencing error rate of 0.5%. TEs were simulated with a mean read coverage of 20X, with 250 TEs deviating from that mean between 2-100 fold.

HEK293T Cell Culture, Transfection and Sequencing

Tet-On HEK293TLD (293T) cells [93] were grown at 37C, 5% CO2 in DMEM with 10% Tet-Free FBS (Takara, Mountain View, CA) and passaged every 3-5 days as needed with regular tests for mycoplasma contamination.

LINE expression constructs were cloned into the pCEP4 backbone (Thermo Fisher Scientific, Waltham, MA) modified to confer puromycin resistance. Plasmids encoded either L1RP (MT302) or had no insert [93]. For transfection, 300,000 293T cells were plated in 2 mL volume. 24 hours later, cells were transfected using a cocktail of 2 ug plasmid DNA and 6 μL Fugene HD (Promega), and puromycin was added 24 hours later for a total of 3 days of selection. 500,000 cells were then plated in 3 wells each, and doxycycline was added 2 hours later (final concentration of 1 ug/ml) to induce L1

expression. RNA was collected after 72 hours of L1 expression using the Zymo Quick-RNA MiniPrep kit (Zymo Research, Tustin, CA). The RNA libraries of transfected 293T cells were prepared using the Illumina TruSeq Stranded Total Library Prep Kit with Ribo-Zero Gold (San Diego, CA) to provide stranded, ribosomal RNA depleted RNA. The libraries were sequenced on an Illumina HiSeq 2500, using 6 samples per lane across 8 lanes with paired-end 100bp reads. We generated a mean of 263,127,067 paired reads per sample. The raw sequencing data were deposited to the NCBI Genome Expression Omnibus (GEO) with accession number GSE113960.

HEK293T Cell RNA-seq Analysis and *In Silico* Spike-in Experiment

For detection of fixed L1 expression identified by Deininger et al. by 5'RACE and poly-A selected RNA sequencing in HEK293 cells, we ran SQuIRE **Map, Count**, and **Call** on HEK293T cell samples transfected with empty L1RP vector (DA5 and DA6). To determine the effect of L1RP transfection on the false positive rate of L1 RNA estimation, we ran **Map** and **Count** on HEK293T cells transfected with L1RP and vector. To simulate the effect of polymorphic TE expression on typical RNA-seq samples, we downsampled a transfected (DA1) and control (DA5) sample to a single lane per sample (average 32 million reads). To identify L1RP aligning reads in the L1RP-transfected cell, we used SAMtools [91] to identify reads that align to the chromosome construct provided by the non-reference table (Table 5). To downsample the L1RP-aligning reads, we used the SAMtools "-s <*INT*.*FRAC*> " option with 0.01, 1.001, and 3.0004 as inputs. The integer before the decimal indicates the seed value and the number after the decimal indicates the fraction of total alignments desired for subsampling. We then identified all alignments to the genome sharing the same Read IDs as the down-sampled L1RP-aligning reads. We used SAMtools merge to combine the alignments of L1RP-aligning reads with the BAM file of the HEK293T cell sample transfected with empty vector (DA5).

TE RNA-seq tool Comparison

Adult C57BL/6 mouse RNA-seq data were obtained from GEO with accession number GSE30352. All pipelines were run on a server with a maximum of 128 GB memory available and 8 threads (-p setting).

RepEnrich [43]– We obtained the hg38 annotation for RepeatMasker from the RepEnrich GitHub website. For the mm10 annotation, we obtained the mm10.fa.out.gz RepeatMasker [51] annotation from the RepeatMasker website. We ran the setup for RepEnrich following instructions from the website for each genome build. We then mapped the data to the genome using Bowtie 1 [94] according to RepEnrich's instructions to generate separate uniquely mapping SAM and multi-mapping read FASTQ files. These were then used for the RepEnrich software with the "–pairedend TRUE" parameter for simulated human data, and "—pairedend FALSE" for mouse data.

TETools [45]– We generated rosette files for hg38 and mm10 for TETools by taking the Repeatmasker annotation from **Clean** for the first column and the repeat taxonomy for the second column (subfamily:family:superfamily). We used the BED file from **Clean** with **Seek** to obtain TE FASTA sequences for generation of a pseudogenome for TETools. TETools was run with the "-bowtie2", "–RNApair" and "–insert 250" parameters for simulated human data and "-bowtie2","-insert 76" for mouse data.

TEtranscripts [44] –We obtained hg38 and mm10 GTF annotation from the TEtranscripts website. We aligned the data to the genome with STAR using "--winAnchorMultimapNmax 100","--outFilterMultimapNmax 100" parameters for multi-mapping. We then ran TEtranscripts with the "--mode multi" setting to utilize its expectation-maximization algorithm for assigning multi-reads for the resulting SAM file. Since TEtranscripts analyzes TE and gene expression together, we used refGene annotation obtained by SQuIRE **Fetch** for the required GTF file. We used the parameters "--format SAM", "--mode multi", "--stranded yes" for simulated human data, and "--format SAM", "--mode multi", "--stranded no" for mouse data.

Aligner Comparison

We ran the aligners Bowtie1 [94], Bowtie2 [95], and STAR [52] on the simulated TE RNA-seq data described above. We set each aligner to output a maximum of 2 valid alignments to quickly identify uniquely aligning reads with the parameter "-m2" for Bowtie 1, "-k2" for Bowtie 2, and "--outSAMmultNmax 2" for STAR. We also ran STAR with the parameters "--outFilterScoreMinOverLread 0.4 --outFilterMatchNminOverLread 0.4 --chimSegmentMin 100" to allow for discordant alignments, which STAR excludes by default. Bowtie2 reports discordant alignments by default, while Bowtie 1 can only report paired alignments. We used BEDTools [90] to intersect the BAM outputs to RepeatMasker annotation to identify the TEs to which the aligners mapped the reads. Reads that only appeared once as "uniquely aligning". We assessed whether the mapped TE matched the templating TE for the simulated read to determine if the uniquely aligning reads mapped to the correct location.

Statistical Analysis

Differential expression analysis of gene and TE expression was performed using DESeq2 [54] via the SQuIRE **Call** tool (see Methods). P-values were adjusted for multiple-comparisons with an FDR cutoff of 0.1. To determine if loci belonging to the IAPLTR3 subfamily were more likely to be differentially expressed in testis compared to other TE subfamily loci, a Fisher's exact test was performed. The Fisher's exact test was chosen due to the small percentage of TE loci that are expressed. We compared the proportion of IAPLTR3 loci in the genome that were differentially expressed in testis to the proportion of other TE subfamily loci that were differentially expressed.

Implementation of STAR aligner in Map

Map uses parameters tailored to the alignment of TEs. By default STAR only reports reads that map concordantly and to 10 or fewer locations. **Map** retains more reads mapped to TEs by reporting reads that fully map to 100 or fewer locations (--alignEndsType EndToEnd --outFilterMultimapNmax 100 –winAnchorMultimapNmax 100). For paired-end reads, **Map** also reports paired reads that map discordantly (--chimSegmentMin <read_length>) and single reads with unmapped mates (--

outFilterScoreMinOverLread 0.4 –outFilterMatchNminOverLread 0.4). **Map** can incorporate the non-reference TE sequences and generate a FASTA file that STAR adds to the genome index with the option "—genomeFastaFiles <fasta> ". To provide splicing information to the tools in the *Analysis Stage*, **Map** also uses the UCSC RefSeq gene annotation and assesses reads overlapping splice junctions with the options "—sjdbGTFfile <gtf> --sjdbOverhang <read_length -1> --twopassMode Basic". **Map** produces a sorted BAM file that includes intron and splicing information for downstream transcriptome assembly analysis.

Implementation of StringTie in Count

Count runs StringTie [53] using these default settings guided by RefSeq gtf obtained from UCSC with **Fetch. Count** uses the "-e" StringTie option to quantify expression only to annotated transcripts without assembly of novel transcripts. We convert the fpkm values to counts by multiplying the per-exon coverage by exon length normalized by read length.

DESeq2 Implementation in Call

Call incorporates the Bioconductor package DESeq2 [54,55] with its suggested parameters. Users input the sample names and experimental design (ie which samples are treatment or control), which **Call** uses to find **Count** data and create a count matrix for annotated RefSeq genes, StringTie transcripts and TEs. **Call** outputs differential expression tables and generates MA-plots, data quality assessment plots, and volcano plots.

STAR implementation in Draw

To visualize the distribution of reads across the TE, **Draw** runs STAR [52] with the parameters "–runMode input AlignmentsFromBAM –outWigType bedGraph" to provide visualization of read alignments. It will output bedgraphs of all reads ("multi") and only uniquely ("unique") aligning reads. **Draw** also compresses the bedgraphs into bigwig format for IGV [56] and UCSC Genome

Browser [96] viewing. If the RNA-seq data is stranded it will output unique and multi bedgraphs for each strand.

Further details of Count

Count uses a combination of SAMTools [91], BEDTools [90], awk and bash within a Python script to perform the algorithm described in the main text. Because the quantitation in SQuIRE relies on uniquely aligning reads, SQuIRE needed to resolve three issues in identifying uniquely aligning reads and their mapped TE location. 1) Because RepeatMasker annotation includes overlapping TE coordinates, a read can map uniquely at one genomic location corresponding to two TE loci. **Count** identifies these reads as unique by collapsing reads and their mapped TEs before labeling. The two TEs each would receive a unique count for that TE. 2) Similarly, when SQuIRE incorporates non-reference polymorphic TE insertions, its location can be confused with overlapping reference TE annotation. To address this, **Map** uses a custom chromosome name for non-reference TEs (eg. "chr3_poly") during alignments. To keep read assignments to non-reference TEs separate from assignments to annotated TEs, **Count** changes the non-reference chromosome name back to the conventional name (eg "chr3") only after collapsing reads mapped to the same location. III) For paired-end RNA-seq data, a read pair may map concordantly in only one location, particularly if one mate maps outside of the TE. However, one or both of the TE mapping mates may not be uniquely aligning, and map discordantly to other locations. In this situation, **Count** does not label these reads as "uniquely aligning", but assigns a full count to the TE and discards the discordant alignments.

Users who want to further reduce false positives can use a score value provided in the **Count** output. The score is the percentage of the reads aligned to the TE that contributed to the total count. A higher score, for example 99%, suggests greater certainty in the count assignment, and that little of the multi-mapping reads were assigned elsewhere to other TEs. Indeed, we found that this strongly correlated with % Observed/Expected with a coefficient ($r=0.94$, $p< 2.2e-16$). When we plotted the

TPR and PPV using various score thresholds, we found that using a score threshold of at least 50% maximized the combination of TPR and PPV.

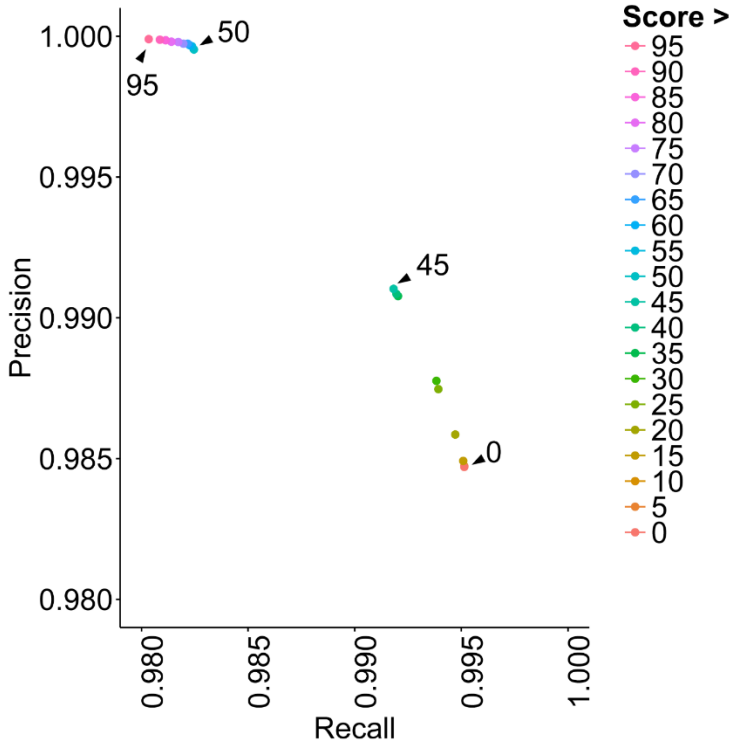


Figure 12. Precision-Recall curve of SQuIRE Count with varying confidence score thresholds.

Precision= Σ "True positive"/ Σ "Positive"). Recall=Σ "True positive"/ Σ "True". Positive=SQuIRE reported the TE has a count >10. True=TE was simulated to express > 10 reads. A score threshold of >50 maximizes precision and recall.

3. Landscape of Transposable Element Expression in Human Cells

3.1 Introduction

Since the human genome was first sequenced, it has long been characterized as islands coding exons surrounded by a sea of 'dark matter' of unknown function[1]. A large portion of that dark matter is comprised of transposable elements. TEs are self-mobilizing DNA sequences interspersed throughout the genome [2]. Whereas DNA transposons "cut-and-paste" their genomic sequence into a new location, retrotransposons are TEs that mobilize using a "copy-and-paste" mechanism via an RNA-intermediate. Retrotransposons consist of long interspersed nuclear elements (LINEs), short interspersed nuclear elements (SINEs) and long terminal repeats (LTRs)[4]. Their method of propagation has resulted in similar copies of shared sequence throughout the genome.

TE insertions have been critical in shaping the human genome[17,97]. TEs contain promoters, cis-regulatory sequences, and other functional domains that affect their transcription [26,85,98–100]. These functional domains may be adapted (or 'exapted') by lncRNA and mRNA genes in which they insert[14,32,100–102]. They may be transcribed as exapted exonic sequence, transient intronic sequence, or "downstream of gene" transcription as part of longer transcripts[88,103–105]. In contrast, TEs may also be autonomously transcribed from their own promoters. While this is necessary for a TE to be retrotranspositonally active, TEs that have lost their ability to mobilize may still be transcribed. We term TEs that are transcribed independent of longer transcripts as "individual TE loci" (ITLs).

TEs can thus be expressed as part of different transcript types. However, the extent of TE transcription in these different contexts has been heretofore unknown. The repetitive nature of TEs has posed computational and experimental challenges for the analysis of TE transcription. Although TEs have been associated with tissue specificity, pluripotency, and cell differentiation, past studies

have been primarily down at the subfamily level[15,24,106]. Without locus level information, these studies have run the risk of conflating TE-driven transcription with background transcription as part of pre-mRNA and lncRNA transcripts. Here, we define the landscape of TE expression in normal human cells using our TE analysis software SQuIRE (Software for Quantifying Interspersed Repeat Expression) to discern TEs in pre-mRNAs, lncRNAs and ITL RNAs. Our study reveals that ITLs are transcribed from infrequent, discrete loci in complex cell- and tissue type-specific patterns.

3.2 Results

To profile TEs expressed in normal human cells, we performed RNA-seq in primary human umbilical vein endothelial cells (HUVEC) using a stranded ribosomal RNA (rRNA) depletion library. To obtain locus-specific estimates of TE RNA expression, we ran the SQuIRE pipeline on the RNA-seq data. Using a stranded RNA-seq library enables detection of whether a TE is transcribed in the sense and/or antisense direction. We observed that 16% of expressed TEs had reads on both strands, though the read count of one strand usually outnumbered the other by at least 2-fold (Figure 13). We focused on the TE orders with the greatest genomic contributions: long interspersed element (LINE), short interspersed elements (SINE), and long terminal repeat (LTR) retrotransposons, and DNA transposons (DNA)[5],[107],[108]. Out of 4,245,814 reference genome TEs within those orders, 235,409 (6%) had 20 reads and > 0.1 fpkm on at least one strand and a confidence score of >95%, indicating high likelihood of expression (Table 7.). This represented a wide range of all but the youngest TE subfamilies. We did not observe a strong correlation between TE age and expression (r= 0.1, P < 2e-16), suggesting that differences in TE expression were not an artifact of TE alignment issues. We used this expressed, high-confidence subset for further analysis.

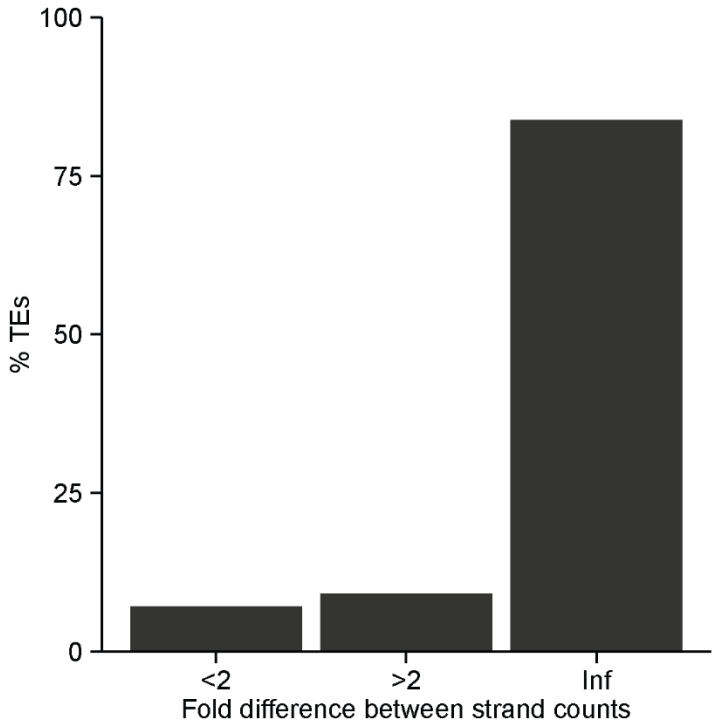


Figure 13. Most TEs are expressed on one strand.

Barplot of % of all TEs with low, high, or infinite absolute fold differences in expression on sense and antisense strands. TEs with <2X fold difference in counts between strands make up 16% of TEs. TEs with an infinite fold difference have read counts on only one strand.

TE Order	% TEs
DNA	6.56
LINE	6.61
LTR	4.64
SINE	4.79

Table 7. A low percentage of TEs is expressed. Percentage of TEs within each TE order with > 20 reads and > 0.1 fpkm on at least one strand.

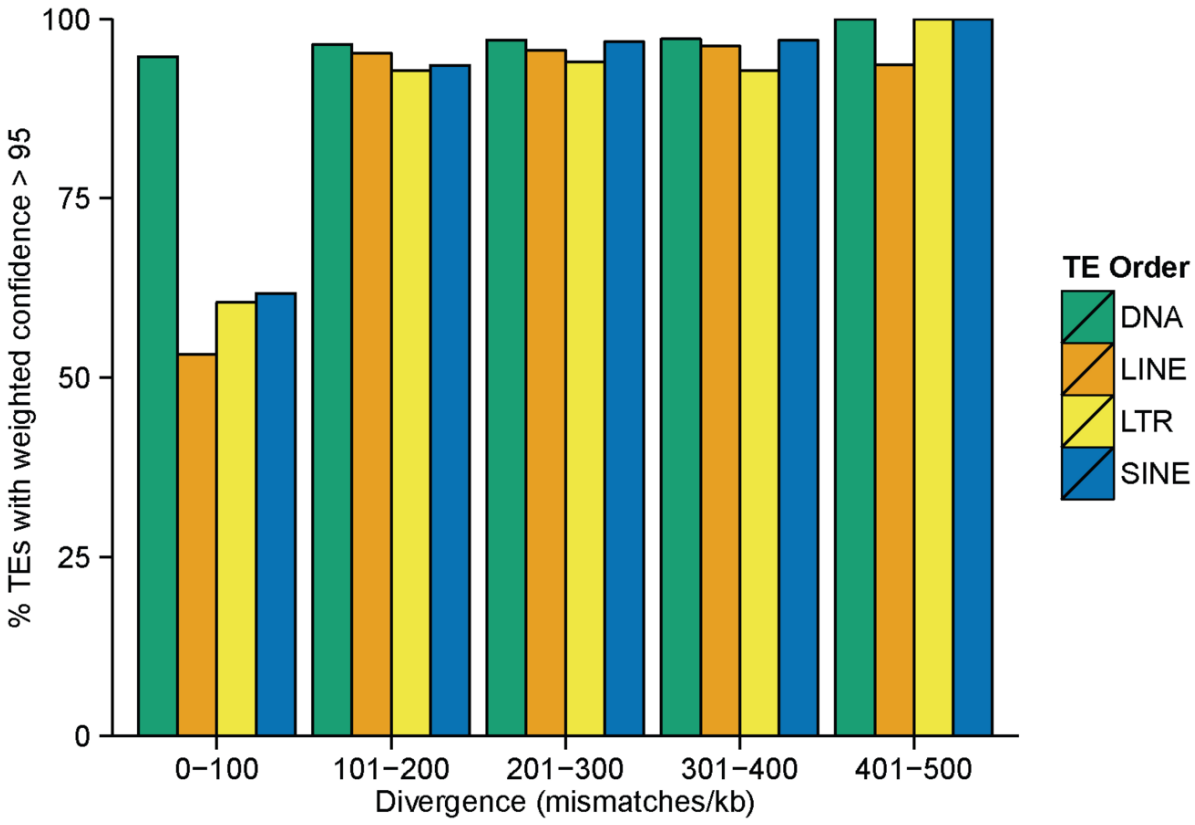


Figure 16. Analysis of high-confidence TE counts is representative of all but the youngest TEs.

Percent of TEs within each TE order with weighted confidence > 95% is high in TEs with divergence > 100 mismatches to consensus sequence/kb. Divergence is used as a proxy for approximating TE age.

The distribution of TE expression loci was disproportionate to their representation in the genome (Figure 14a). Furthermore, we found that 99% of transcribed TEs had reads aligning beyond the 5' or 3' ends of the annotated TE sequence, suggesting they were part of larger transcripts (Figure 15). To examine the distribution of expressed TEs further, we categorized them by their positions upstream, in 5' UTRs, introns, exons, and 3 ' UTRs, and downstream of the nearest gene. We found that the overwhelming majority (87%) of expressed TEs arose from introns (Figure 16a). Only 15,311 out of 204,319 (7%) intronic TEs were also found using a poly-A selection library of the same RNA, suggesting that the rest were likely part of precursor mRNA (pre-mRNA) transcripts, consistent with previous studies (Figure 16b)[109]. Consistent with this, rRNA depletion increased the number of identified intergenic TEs 2-fold (Figure 16c). Relative to their representation in genomic poositions relative to the nearest gene, TE expression was most enriched in 3'UTRs (OR = 14.88, 95% CI = 14.42-15.35, $P < 2 \times 10^{-16}$) and least in 5'UTRs (OR = 2.59, 95% CI = 2.35-2.84, $P < 2 \times 10^{-16}$), and depleted in intergenic spaces (OR = 0.07, 95% CI = 0.07-0.08 $P < 2 \times 10^{-16}$) (Figure 14b). The transcribed TEs within and downstream of genes were primarily in sense orientation to the nearest gene (98% intragenic, 85% downstream) and independent of the annotated TE strand (Figure 14c). The expression of downstream TEs may be the result of downstream of gene (DoG) read-through transcription[105]. Indeed, the number and expression level of downstream TEs decreased with distance from the nearest gene ($r = -0.04$ $P < 1 \times 10^{-7}$) (Figure 14d). This was not significant for upstream TEs, which were fewer in number (Downstream: 16,311 TEs, Upstream: 6,268). The extent of TE expression from genes highlights that the majority of TE expression is driven by the transcriptional mechanisms of the surrounding or upstream gene.

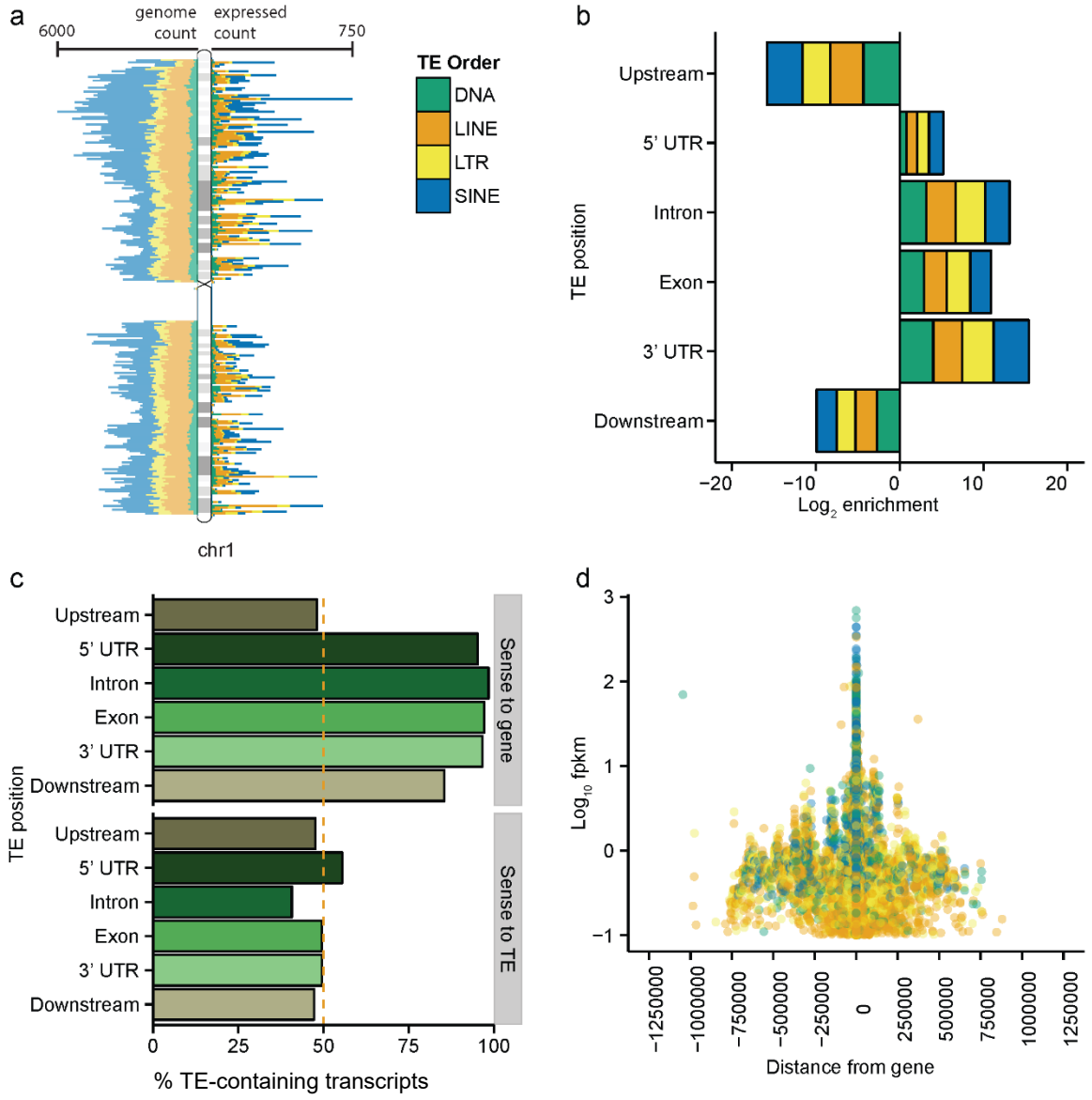


Figure 14. Characteristics of TE expression.

a) Distribution of TE expression across chromosome 1. Transposable elements are expressed disproportionately relative to their representation in the genome. Left, stacked histogram of genomic reference LINE, SINE, LTR, and DNA TEs on chromosome; right, stacked histogram of subset of those TEs that are expressed (weighted confidence > 95%, counts > 20). DNA elements are represented in green, LINEs in orange, LTRs in yellow, and SINEs in blue. b) Transposable elements of all TE orders are enriched within genes and depleted in intergenic spaces in HUVEC. Log2 fold enrichment of % TEs expressed in ribosomal rRNA depleted HUVEC RNA relative to their representation in the reference genome. c) The direction of intragenic and downstream TE transcription is driven by the strand of the nearby gene. Percent of TE-containing transcripts that are sense to the nearest gene or TE annotation. Transcripts of TEs on both strands were treated separately. d) TE expression level in log10 of counts per kilobase annotated TE length per million reads (fpkm) relative to distance from the nearest gene. Intragenic TEs have distances of 0, upstream TEs have negative value distances and downstream TEs have distances greater than 0.

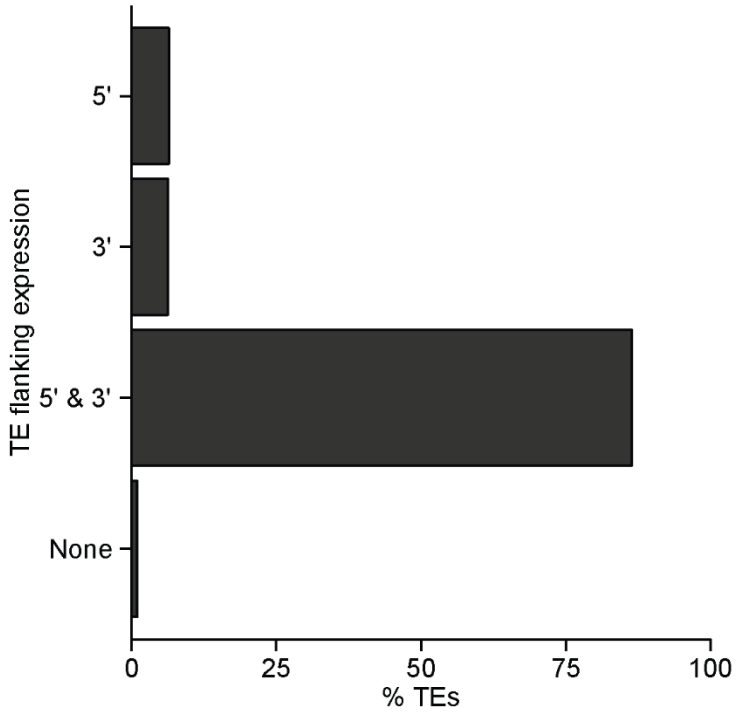


Figure 15. Most TE-containing RNA transcripts extend beyond TE sequence.

Percent of all TEs that are flanked at the 5' end, 3' end, or both (5' & 3' flanked), and percent of TEs that have no flanking expression.

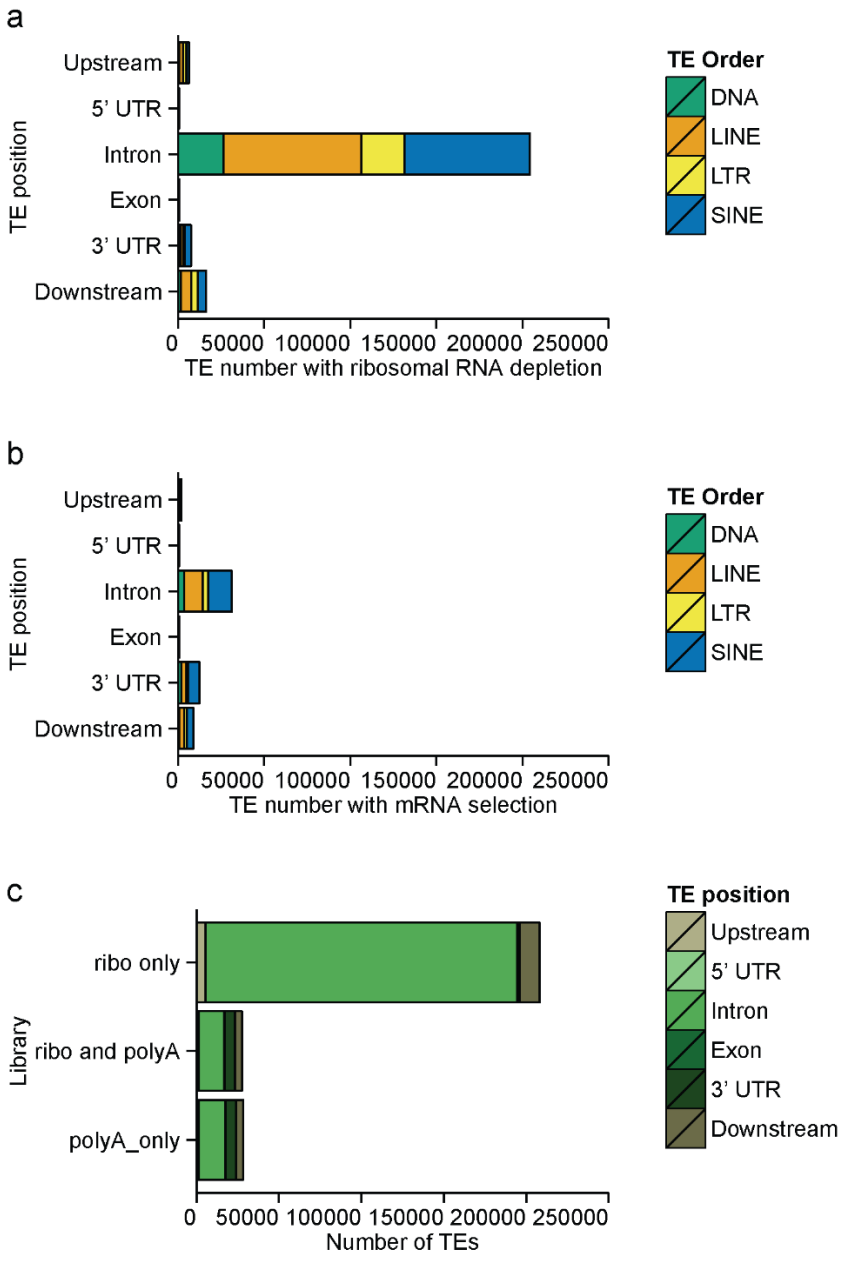


Figure 16. Ribosomal RNA depletion of HUVEC RNA enriches for transposable elements, particularly intronic elements, as compared to poly-adenylated mRNA selection.

a) Count of TEs arising from upstream, 5' UTR, exonic, intronic, 3'UTR, and downstream compartments using a ribosomal RNA depletion HUVEC library. b) Count of TEs arising upstream, 5' UTR, exonic, intronic, 3'UTR, and downstream compartments using a poly-A selection HUVEC library. Green represents DNA transposons, orange represents LINEs, yellow represents LTRs, blue represents SINEs. c) Number of transcribed TEs detected by ribosomal RNA depletion library, poly-A selection library, or both. Brown colors indicate intergenic TEs, green indicates TEs arising from genes.

We next set out to further characterize the longer transcripts containing TEs. As we have previously observed, most TEs were expressed close to each other on the same strand with flanking expression. We also observed enriched TE expression downstream of genes, which we expected were due to read-through transcription. To treat read-through TEs as part of the same transcripts as intragenic TEs, we joined the coordinates of all TEs within 5kb of each other on the same strand intersected the coordinates with RefSeq gene annotations, and lncipedia and NONCODE lncRNAs annotations. Out of 16,798 TE-containing transcripts with an average fpkm > 0.1, 3,707 (22%) were completely intergenic, 5,051 (29%) were completely within a RefSeq gene, 4,676 (28%) overlapped and flanked one gene, and 3,364 (21%) overlapped multiple genes. Interestingly, there were 3,633 instances of transcripts anti-sense to annotated genes, which supports reports of TE involvement in sense-antisense pairing to regulate RNA stability [110,111]·[112]·[113]. Among flanking transcripts overlapping a single gene, 13% were oriented antisense to the gene (Table 8). In looking more closely at antisense transcripts, we noticed a subset of 11 transcripts that were > 50 kb long and > 100 kb upstream and antisense from the nearest gene. When we examined one of these transcripts near the Activated Leukocyte Cell Adhesion Molecule gene (*ALCAM*) in other cell types, we found high antisense expression at either the 5' end or at ~500 kb intervals upstream from the gene, but not both; there was no upstream expression when *ALCAM* expression was low (Figure 17). This suggested the involvement of chromatin looping that brought the distant region close to a bidirectional promoter, allowing for antisense transcription. When we compared 6,168 intergenic and antisense transcript coordinates with databases of long noncoding RNAs (lncRNAs), 2,248 (36%) overlapped with lncRNA annotation[50]·[114]·[115]. However, 1,235 of those lncRNAs transcripts were longer than reported, and another 3,920 were not annotated as lncRNAs at all. A contributing factor may be our ability to resolve repetitive content; unannotated lncRNAs had 38% more low confidence TEs ($P <$ 0.001). SQuIRE may thus provide a bottom-up approach to mapping TE-rich lncRNA expression.

Position relative to gene	Sense		Antisense	
Upstream	360	(8%)	244	(5%)
Downstream	2492	(55%)	175	(4%)
Both	1058	(23%)	177	(4%)

Table 8. Number of TE-containing transcripts that overlap a single gene categorized by their position and strand orientation relative to the gene.

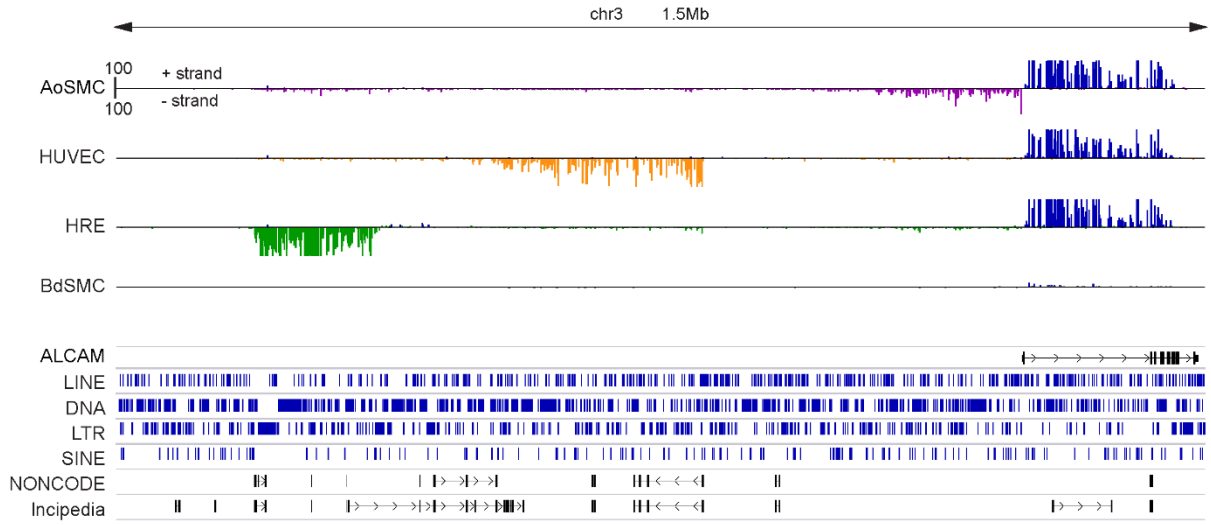


Figure 17. Antisense long non-coding RNA (lncRNA) expression upstream of the ALCAM gene.

Plus-strand expression is depicted as upright bars, minus-strand expression is depicted as upside-down bars. When plus-strand gene expression at ALCAM is high in aortic smooth muscle (AoSMC), human umbilical endothelial (HUVEC), human renal epithelial (HRE) cell types, there is also minus-strand expression initiated in close proximity to the promoter (AoSMC), at 500kb (HUVEC), or at 1Mb (HRE) upstream. Minus-strand expression was low or absent at these locations when ALCAM gene expression was low in the bladder smooth muscle cell type (BdSMC). Positions of TEs and NONCODE and lncipedia tracks are shown (bottom). The expressed TE region in HUVEC (blue, chr3:104,569,124-104,911,050) extended beyond the lncRNA annotations in NONCODE and lncipedia databases. The expressed TE regions in AoSMC (purple, chr3:104,569,124-104,911,050) and in HRE (green, chr3:103,486,909-105,401,256) were not annotated in NONCODE or lncipedia databases.

After grouping nearby expressed TEs in HUVEC RNA into lncRNA and pre-mRNA transcripts, we noticed single-TE intergenic transcripts. To determine if these transcripts were derived from individual TE loci (ITLs) or low-expressing lncRNAs, we ran the SQuIRE pipeline on RNA-seq data from additional early-passage primary cell types representing four tissue types (epithelial, muscle, connective, and nervous tissue) for a total of 31 datasets (Table 9). Nervous tissue cells included both non-neuronal (astrocytes, retinal pigment epithelial cells) and neuronal (cortical and dopaminergic neurons) cells. To identify TEs that are transcribed independently from nearby gene transcription or pervasive transcription from open chromatin, we excluded TEs expressed within 40kb of other TEs with high confidence scores. From the remaining ungrouped individual TEs we further excluded those that overlapped same-strand lncRNA annotation or had low-confidence TE expression within 5kb. Using these more stringent criteria we identified 128 intergenic single-TE transcripts (Figure 18).

Sample	Cell name	Tissue	Cell type code	Lonza catalog #	Media	# R1	# R1 aligned	# R1 removed->100 alignments	# R2	# R2 aligned	# R2 removed->100 alignments
JH-01	Astrocytes	Nervous	NH-A	CC-2565	ABM	104471541	16734016	173052	104471541	16508298	176654
JH-02	Bronchial Epithelial Cells	Epithelial	NHBE	CC-2541	BEBM	150044582	20977201	175381	150044582	21732619	190472
JH-03	Articular Chondrocytes	Connective	NHAC	CC-2550	CGM	150411161	24913726	182460	150411161	24523984	188784
JH-04	Umbilical Vein Endothelial Cells	Muscle	HUVEC	C2517A	EBM	101437827	15612697	204169	101437827	15412373	208790
JH-05	Microvascular Endothelial Cells	Muscle	HMVEC	CC-2527	EBM	145306522	25585415	344842	145306522	25371676	356872
JH-06	Adult Dermal Fibroblasts	Connective	NHDF-Ad	CC-2511	FGM-2	109966976	17607311	156394	109966976	17356006	158476
JH-07	Neonatal Dermal Fibroblasts	Connective	NHDF-neo	CC-2509	FGM-2	116297362	16221909	177115	116297362	15953977	180027
JH-08	Cardiac Fibroblasts	Connective	NHCF-V	CC-2904	FGM-3	145307299	19434467	157461	145307299	19130187	162675
JH-10	Adult Epidermal Keratinocytes	Epithelial	NHEK-Ad	192627	KBM-Gold	99230384	16698295	132771	99230384	16505043	136457
JH-11	Neonatal Epidermal Keratinocytes	Epithelial	NHEK-neo	192907	KBM-Gold	105295191	16417102	146545	105295191	16193962	148699
JH-12	Melanocytes	Connective	Melano	CC-2586	MBM	122998526	34925795	135870	122998526	34431741	140581
JH-13	Mammary Epithelial Cells	Epithelial	HMEC	CC-2551	MEBM	132919896	19485983	135013	132919896	18977598	138024
JH-14	Mesangial Cells	Connective	NHMC	CC-2559	MsBM	103645711	16748597	114869	103645711	16537015	117869
JH-15	Osteoblasts	Connective	NHOst	CC-2538	OBM	105081784	17113893	129767	105081784	16858813	131866
JH-16	Prostate Epithelial Cells	Epithelial	PrEC	CC-2555	PrEBM	139778855	19865520	203404	139778855	19606632	209681
JH-17	Renal Cortical Epithelial Cells	Epithelial	HRCEpiC	CC-2554	REBM	142067399	16307398	126011	142067399	16020699	130835
JH-18	Renal Epithelial Cells	Epithelial	HRE	CC-2556	REBM	111719136	15682605	155406	111719136	15437996	158759
JH-19	Renal Proximal Tubule Cells	Epithelial	RPTEC	CC-2553	REBM	152096250	19190882	190407	152096250	18878849	196423
JH-20	Retinal Pigment Epithelial Cells	Nervous	HRPEpiC	194987	RtEBM	120497465	15748554	125601	120497465	15557435	128943
JH-21	Aortic Adventitial Fibroblasts	Connective	AoAF	CC-7014	SCBM	120211998	18561804	152883	120211998	18262976	155121
JH-22	Periodontal Ligament Fibroblasts	Connective	HPdLF	CC-7049	SCBM	118898009	15858142	176681	118898009	15681232	179944
JH-23	Prostate Stromal Cells	Connective	PrSC	CC-2508	SCBM	145233321	21287774	168514	145233321	20929585	173471
JH-24	Skeletal Muscle Myoblasts	Muscle	HSMM	CC-2580	SkBM-2	149264697	21435826	189340	149264697	21099800	196845
JH-25	Skeletal Muscle Cells	Muscle	SKMC	CC-2561	SkBM	112514596	16502658	127338	112514596	16502658	127338
JH-26	Aortic Smooth Muscle Cells	Muscle	AoSMC	CC-2571	SmBM	113807352	15721845	161076	113807352	15480861	162528
JH-28	Myofibroblasts	Connective	InMyoFib	CC-2902	SmBM	146751813	21002619	163131	146751813	21356991	174040
JH-29	Prostate Smooth Muscle Cells	Muscle	PrSMC	CC-2587	SmBM-2	146663555	18711229	125486	146663555	18323391	129704
JH-30	Smooth Muscle Cells	Muscle	BdSMC	CC-2533	SmBM-2	110147689	17707323	179059	110147689	17484196	181779
JH-31	Aortic Endothelial Cells	Muscle	HAEC	NA	ECM	134747757	19209995	223771	134747757	18987453	226251
JH-42	Dopaminergic Neurons	Nervous	Dneuron	NA	SRM	163558814	30630126	498188	163558814	31133963	527452
JH-46	Cortical Neurons	Nervous	Cneuron	NA	KoSRM	171546721	35188620	569011	171546721	34682123	582666

Table 9. Tissue type and RNA sequencing information of 31 cell types.

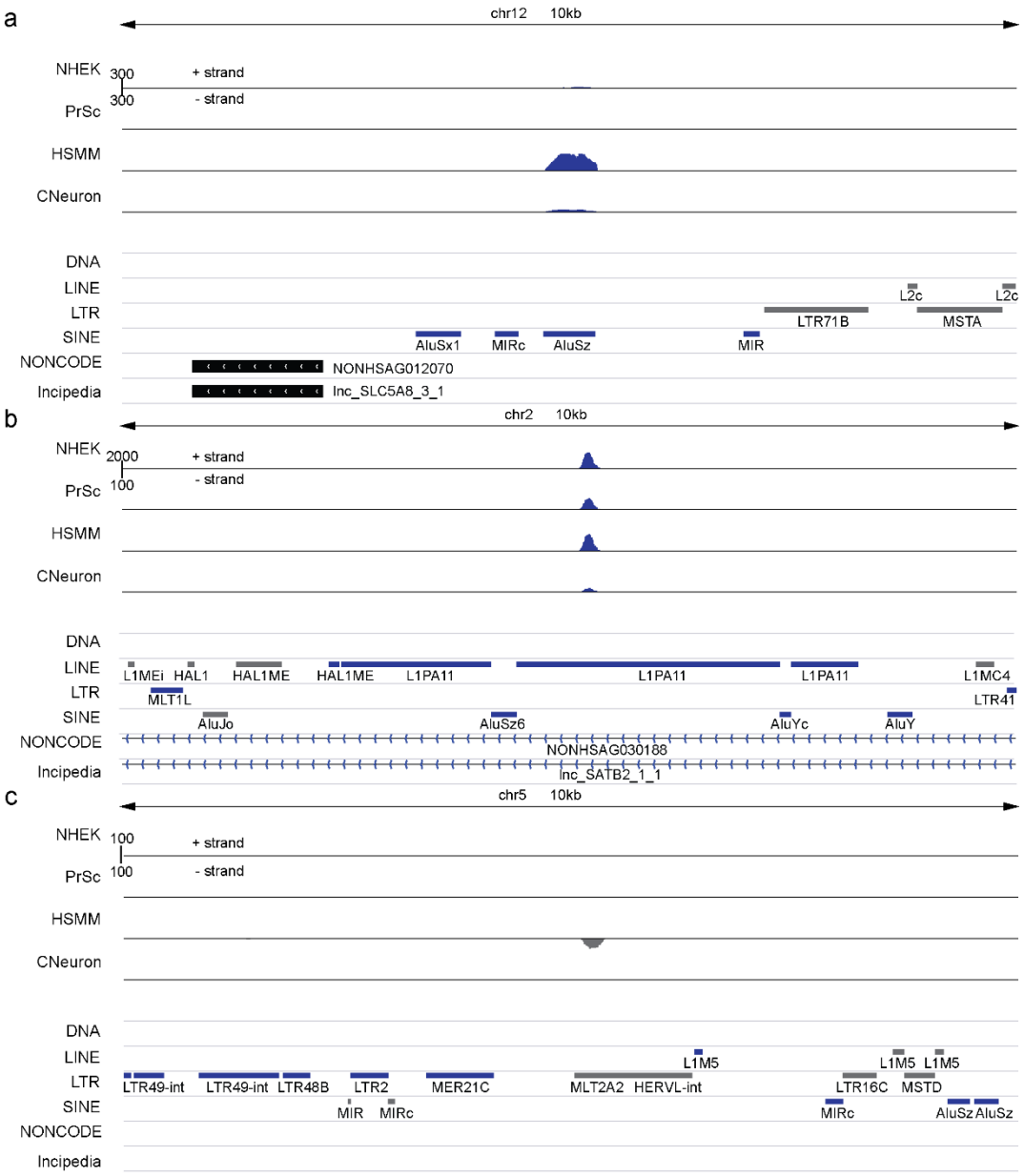


Figure 18. Examples of retrotransposon TE expression from individually transcribed loci (ITLs) in adult epithelial keratinocytes (NHEK), prostate stromal cells (PrSc), skeletal muscle myoblasts (HSMM) and cortical neurons (CNeuron).

Plus strand expression is depicted as upright positive counts, while minus strand expression is depicted as upside-down negative counts. ITLs have no other TE expression within at least 5 kb in either direction on the same strand. a) Transcription of a SINE subfamily AluSz at chr12:101,230,117-101,230,409 covers much of the length of the annotated sequence. There was no annotated lncRNA overlapping the TE. The transcript is sense to the strand of the TE RNA intermediate. b) Transcription of a LINE element L1PA11 at chr2:198,894,019-198,897,256 is truncated (250bp) relative to the annotated sequence length (3237bp). The transcript is sense to the strand of the TE RNA intermediate, and antisense to an annotated, spliced lncRNA. c) Transcription of LTR element MLT2A2 at chr5:5,090,378-5,090,926. The transcript is sense to the strand of the TE, which is the LTR region of a HERVL element.

Sample	Library preparation	# R1 reads	# R1 aligned	# R1 removed->100 alignments	# R2 reads	# R2 aligned	# R2 removed->100 alignments
HUVEC-stranded	Illumina TruSeq Stranded Total Library Prep Kit with Ribo-Zero Gold	175152058	22629980	268645	175152058	22423902	275671
HUVEC-polyA	Illumina TruSeq RNA Library Preparation Kit v2 kit	145165251	6053631	92631	145165251	5969775	91724

Table 10. RNA sequencing information of HUVEC in ribosomal RNA-depletion and poly-adenylated mRNA selection libraries.

We further characterized this class of single-TE transcripts. To assess if the type of TE phylogeny differs between ITL TEs and intergenic lncRNAs TEs, we performed Fisher's exact tests to test for enrichment of TE orders categorized as an ITL or part of an intergenic lncRNA transcript compared to the representation of each TE order among all expressed TE transcripts. Whereas non-ITL, intergenic lncRNAs TEs were more likely to contain LTR transposons (OR = 2.46, 95% CI = 2.43-2.49, $P < 2 \times 10^{-16}$) and were depleted in SINEs (OR = 0.57, 95% CI = 0.56-0.58, $P < 2 \times 10^{-16}$), ITLs were enriched for SINES (OR = 4.26, 95% CI = 2.88-6.43, $P < 1 \times 10^{-14}$) and not likely to be LTRs (OR = 0.57, 95% CI = 0.29-1.03, $P > 0.05$) (Figure 19a). Expression of ITLs (mean = 2.33 fpkm, sd = 8.84 fpkm) was higher than the average non-ITL TE expression (mean =1.18 fpkm, sd = 27.52, $P < 1 \times 10^{-7}$) (Figure 19b). With an average of 158.54 ± 72.83 mismatches/kb, ITLs were also younger compared to TEs from other loci (212.91 ± 81.22, $P < 1 \times 10^{-13}$). In contrast to gene and lncRNA TEs, 88% of the ITL transcripts were in sense orientation with the annotated TE strand, and only 39% were sense to the nearest gene (Figure 19c). Retotransposon ITLs were more likely to be transcribed without flanking transcription than TEs part of larger transcripts (OR=27.06, 95% CI =22.38-32.65, $P < 2 \times 10^{-16}$) and 87% initiated transcription at or downstream from the TE start. Among all the ITLs, 8 had > 0.1fpkm and > 20 counts on both strands; 7 of them were on the opposite strand of a longer lncRNA and could be involved in lncRNA regulation. The other was a DNA transposon (Tigger19b) that also had the highest expression with a maximum of 118 fpkm and 2711 reads (Figure 20a). DNA transposons, although only 4% of ITLs, also differed from retotransposon ITLs by being more likely to have flanking transcription (OR=48.75, 95% CI = 8.27-1954.64, $P < 1 \times 10^{-12}$), with 98% initiating upstream of the TE sequence (Figure 20 b and c). Nevertheless, all of the DNA ITLs were transcribed in sense to the TE strand, and nearby TEs exhibited no expression. Together these findings suggest ITL expression is driven by neither pre-mRNA nor lncRNA transcription.

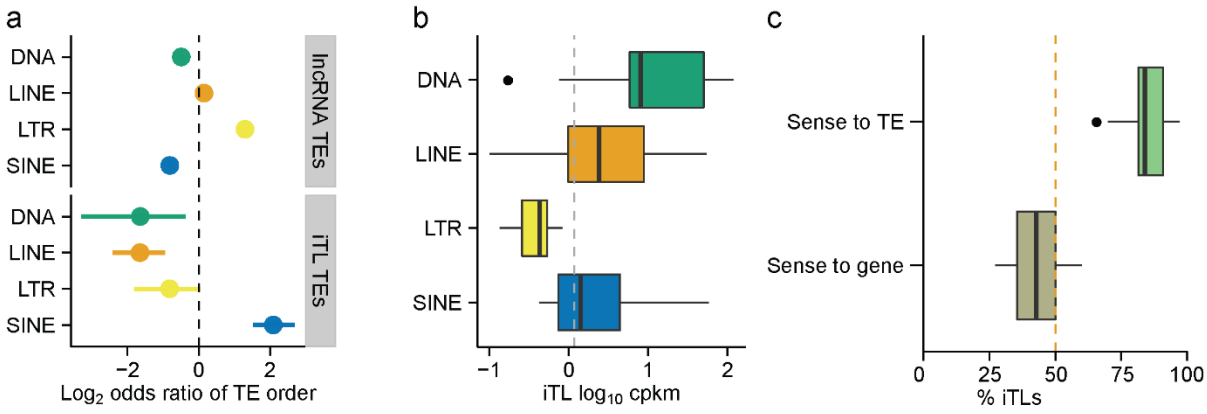


Figure 19. Characteristics of ITLs.

a) SINEs disproportionately contribute to ITLs (OR=4.26, 95% CI=2.88-6.42, P< 1 x10-14) relative to other TE orders (DNA OR=0.32, 95% CI=0.10-0.77, P< 0.01, LINEs OR=0.32, 95% 0.19-0.52, P< 1 x10-6, LTR OR=0.57, 95% CI=0.29-1.03, P > 0.05). Other intergenic TE-containing transcripts (lncRNAs) are enriched in LTRs (OR=2.46, 95% CI=2.43-2.49, P< 2 x10-16) and depleted in SINEs (OR=0.57, 95% CI=0.56-0.58, P< 2 x10-16) and DNA TEs (OR=0.71, 95% CI=0.70-0.72, P< 2 x10-16). Dashed line indicates an odds ratio of 1, or no enrichment or depletion. b) Log10 fpkm ITL expression across TE orders. Dashed line represents expression level of average non-ITL TEs (1.17fpkm). Average fpkms: DNA=27.86, LINE=7.10, LTR=0.43, SINE=4.97. c) ITLs directionality is driven by the encoding TE rather than nearby gene or lncRNA transcription. Percent of ITLs across all cell types that are sense to TE or gene direction is shown. Dashed line at 50% indicates no strand bias. ITLs are most commonly sense to the annotated TE strand (87%), but not to the nearest gene's strand (38%).

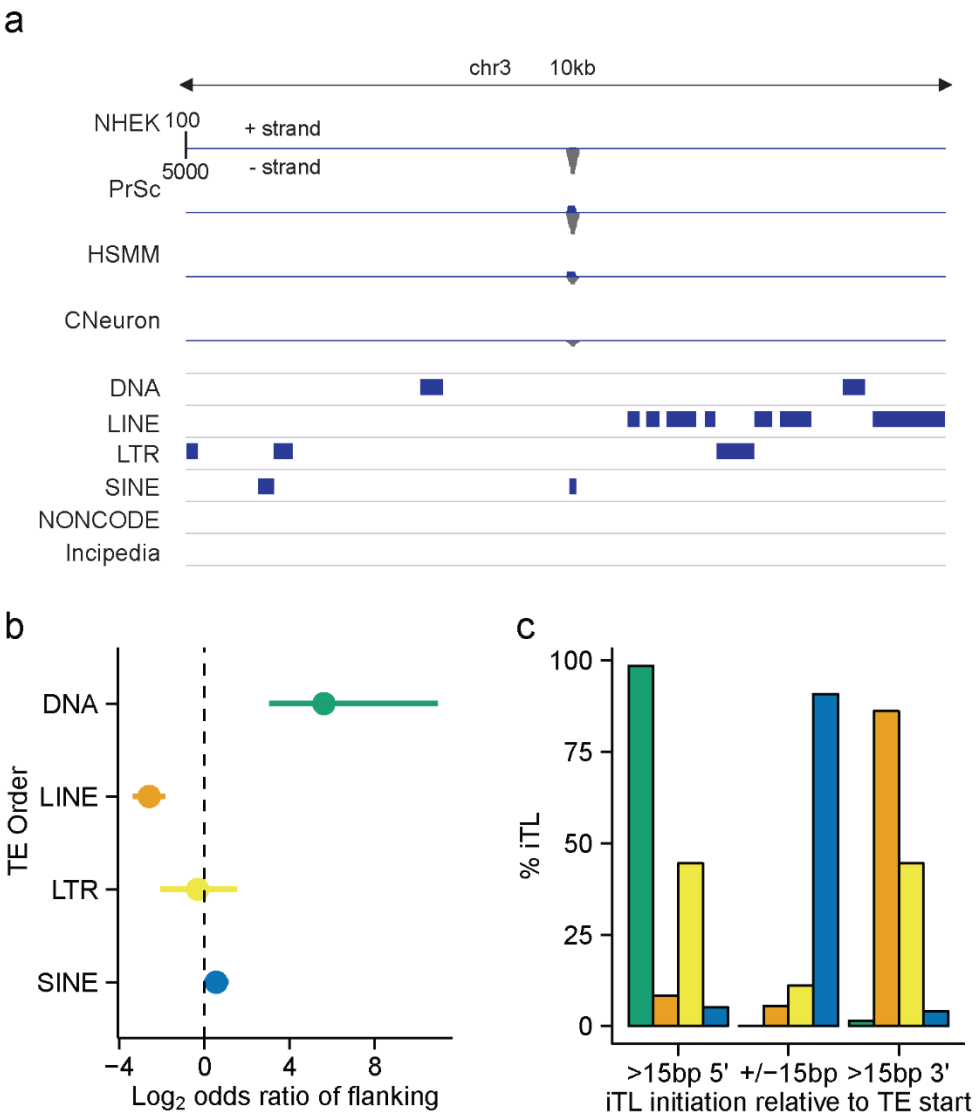


Figure 20. DNA transposon ITL expression patterns across multiple cell types.

a) The highest expressing ITL (mean expression on + strand: 1340.10 reads, 52.21 fpkm) is a plus (+) strand Tigger19 at chr3:82807390-83807468 with expression both strands. b) Odds ratio of having flanked expression of ITLs from different superfamilies. DNA transposon ITLs are the only TE order significantly likely to have transcripts extending beyond the 5' and 3' ends of the TE annotation (OR: 48.75, 95% CI= 8.27-1954.64, P < 1x10-12. Dashed line indicates equal likelihood of having and not having flanking expression among ITLs of TE order. c) Percent of ITLs from each TE order with transcript starts initiating upstream (>15bp 5'), at (±15bp), or downstream (>15bp 3') of the TE annotated start.

To support that ITL transcripts are distinct from intergenic and gene-overlapping transcript regions, we performed non-supervised hierarchical clustering treating using these three transcript types. We found that we were able to accurately group the cells according to organ and tissue type (Supplementary Fig. 9). The ITLs were overrepresented in number among all TE-containing transcribed regions (OR 21.42, 95% CI = 11.81-38.48, $P < 1 \times 10^{-19}$) and were highly cell-specific. Only five were expressed in all 31 cell types while 79 ITLs were expressed in only one cell type (Supplementary Fig. 10). We noticed that cells of epithelial and nervous lineage expressed significantly more ITLs on average than those derived connective tissue (Connective: 12.09, Muscle 13.88, Epithelial: 19.50, Nervous: 21.25 average ITLs, $P < 0.05$) (Supplementary Fig. 11). When we examined ITL expression patterns across cells, we identified two clusters of TEs particularly involved in characterizing epithelial and neuronal cells (Fig. 6). In addition, we found that SINEs (particularly the *Alu* and MIR subfamilies) were enriched in epithelial cells (OR=1.80, 95% CI=1.17-2.81, $P < 0.05$) and depleted in neuronal cells (0.44, 95% CI=0.22-0.87, $P < 0.05$), and LINEs (particularly L1s) were enriched in neuronal cells (OR=2.50, 95% CI =1.20-5.03, $P < 0.05$) and depleted in epithelial cells (0.51, 95% CI=0.29-0.88, $P < 0.05$) (Supplementary Fig. 12). Despite this enrichment, LINE expression in fpkm was significantly lower in nervous tissue cell types compared to all other cells (nervous tissue: 1.37 fpkm, non-nervous tissue 9.06 fpkm, $P < 1 \times 10^{-7}$) (Supplementary Fig. 13). Studying the landscape of ITL expression suggests a complex regulation involving both TE and cell type.

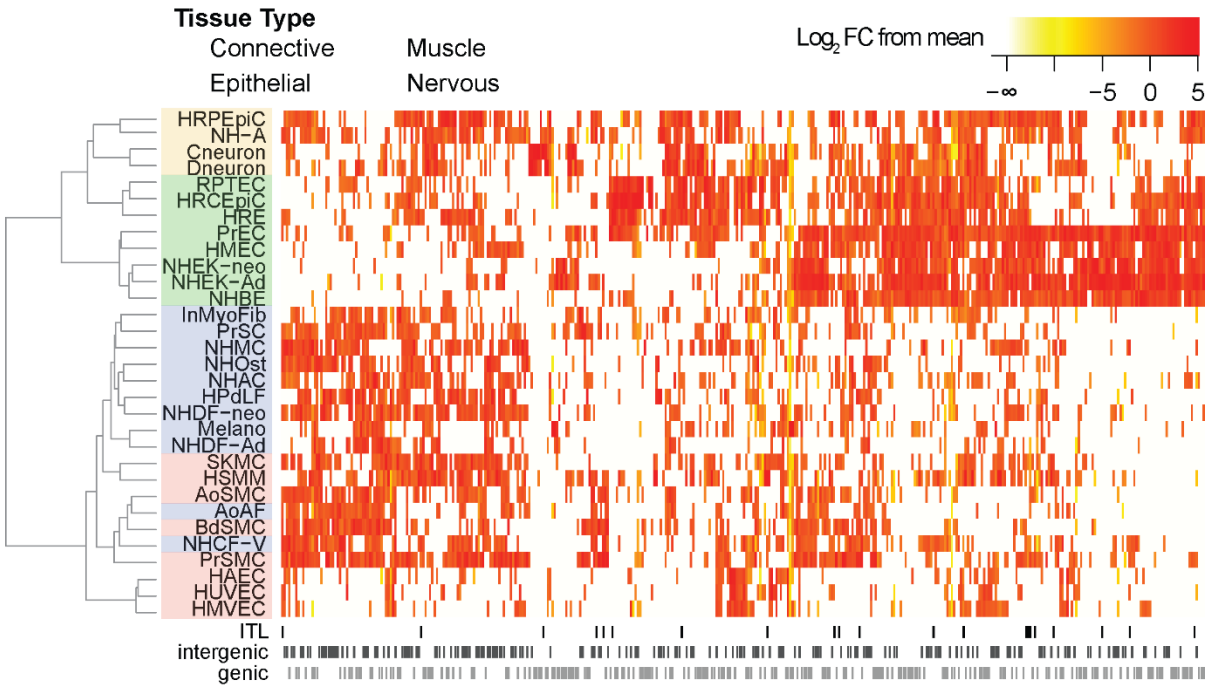


Figure 21. TE-containing transcribed regions can be used to group cell types by tissue and organ type.

Colors in the heatmap represent log2 fold-change from mean region fpkm expression across cell types. White indicates absence of expression in that cell type. Transcribed regions are expressed, high-confidence TEs grouped within 40kb of each other. Transcribed regions either overlap genes ("genic"), are intergenic, or are ITLs as indicated on x-axis. Cell type abbreviations: HRPEpiC: retinal pigment epithelial cells; NH-A: astrocytes; Cneuron: cortical neurons; Dneuron: dopaminergic neurons; RPTEC: renal proximal tubule cells; HRCEpiC: renal cortical epithelial cells; HRE: renal epithelial cells; PrEC: prostate epithelial cells; HMEC: mammary epithelial cells; NHEK-neo: neonatal epidermal keratinocytes; NHEK-Ad: adult epidermal keratinocytes; NHBE: bronchial epithelial cells; InMyoFib: Myofibroblasts; PrSC: prostate stromal cells; NHMC: mesangial cells; NHOst: osteoblasts; NHAC: articular chondrocytes; HPdLF: periodontal ligament fibroblasts; NHDF-neo: neonatal dermal fibroblasts; Melano: Melanocytes; NHDF-Ad: adult dermal fibroblasts; SKMC: skeletal muscle cells; HSMM: skeletal muscle myoblasts; AoSMC: aortic smooth muscle cells; BdSMC: bladder smooth muscle cells; NHCF-V: cardiac fibroblasts; PrSMC: prostate smooth muscle cells; HAEC: aortic endothelial cells; HUVEC: umbilical vein endothelial cells; HMVEC: microvascular endothelial cells.

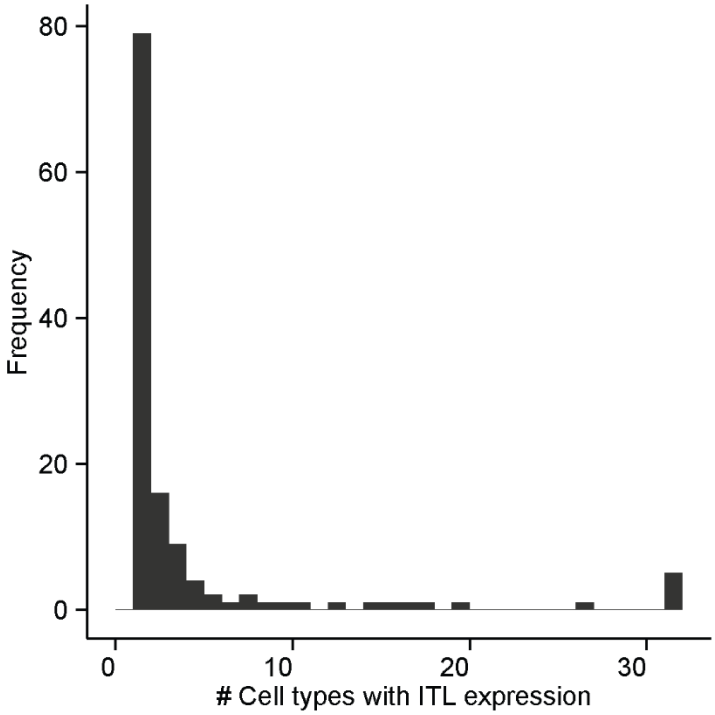


Figure 22. Most ITLs are expressed in only one cell type.

Histogram of number of ITLs expressed in various numbers of cell types.

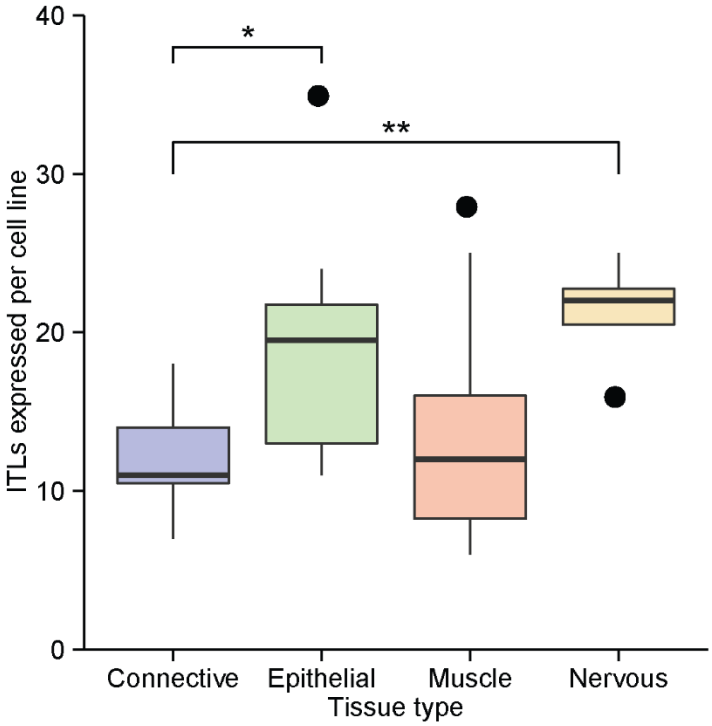


Figure 23. Epithelial and nervous tissue cell types express more ITLs on average than connective tissue cell types.

Boxplot of number of ITLs expressed per cell type in connective, epithelial, muscle and nervous tissue cell types. The single asterisk (*) indicate significance with $P < 0.05$. Double asterisks (**) indicate $P < 0.01$.

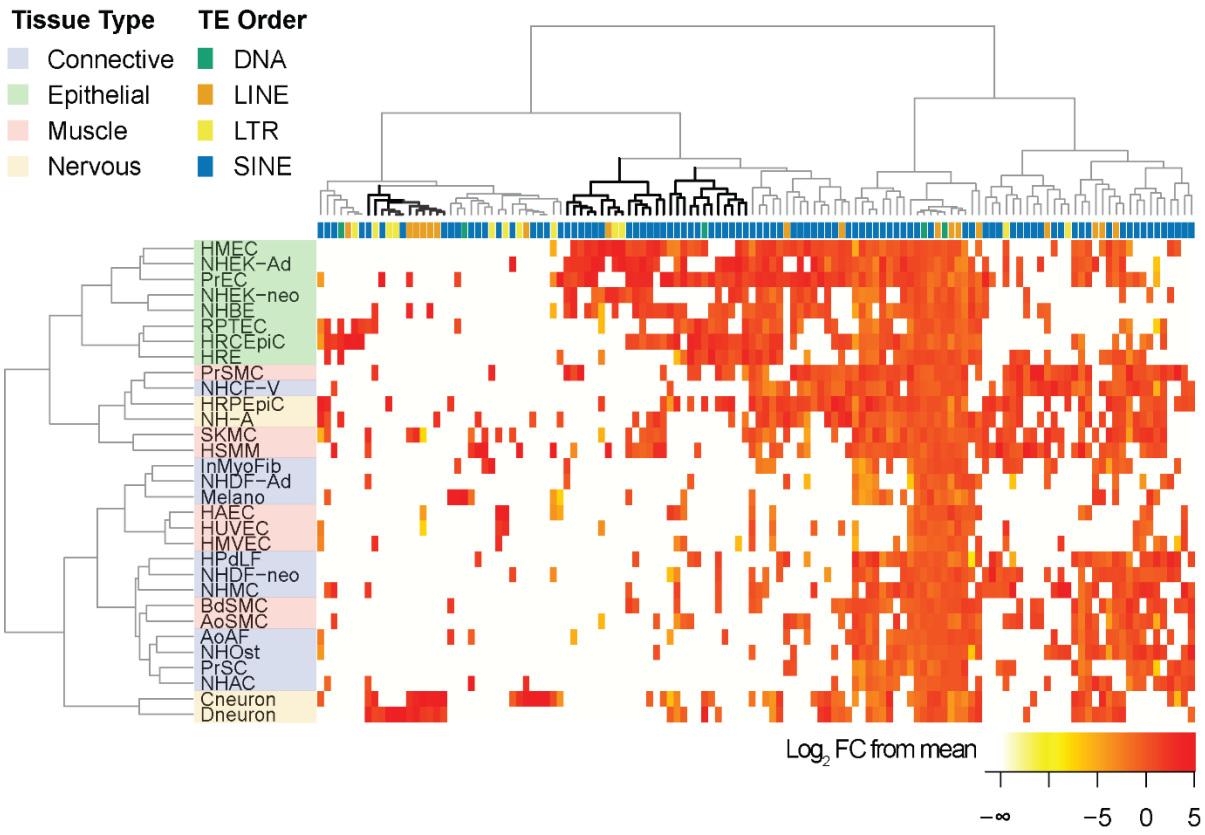


Figure 24. Clustering ITL expression by cell type reveals epithelial and neuronal-specific patterns of expression.

ITL transcripts are indicated on the x-axis and colored by TE order (green: DNA, orange: LINE, yellow: LTR, blue: SINE). Cell types and their tissue types are indicated on the y-axis (blue: connective, green: epithelial, red: muscle, yellow: nervous tissue cell types). Colors in heatmap represent log2 fold-change from mean ITL expression across cell types. White indicates absence of ITL expression in that cell type. Bolded cluster lines onthe x-axis indicate clusters particularly involved in grouping neuronal and epithelial cell types by ITL expression. Cell type abbreviations: HRPEpiC: retinal pigment epithelial cells; NH-A: astrocytes; Cneuron: cortical neurons; Dneuron: dopaminergic neurons; RPTEC: renal proximal tubule cells; HRCEpiC: renal cortical epithelial cells; HRE: renal epithelial cells; PrEC: prostate epithelial cells; HMEC: mammary epithelial cells; NHEK-neo: neonatal epidermal keratinocytes; NHEK-Ad: adult epidermal keratinocytes; NHBE: bronchial epithelial cells; InMyoFib: Myofibroblasts; PrSC: prostate stromal cells; NHMC: mesangial cells; NHOst: osteoblasts; NHAC: articular chondrocytes; HPdLF: periodontal ligament fibroblasts; NHDF-neo: neonatal dermal fibroblasts; Melano: Melanocytes; NHDF-Ad: adult dermal fibroblasts; SKMC: skeletal muscle cells; HSMM: skeletal muscle myoblasts; AoSMC: aortic smooth muscle cells; BdSMC: bladder smooth muscle cells; NHCF-V: cardiac fibroblasts; PrSMC: prostate smooth muscle cells; HAEC: aortic endothelial cells; HUVEC: umbilical vein endothelial cells; HMVEC: microvascular endothelial cells.

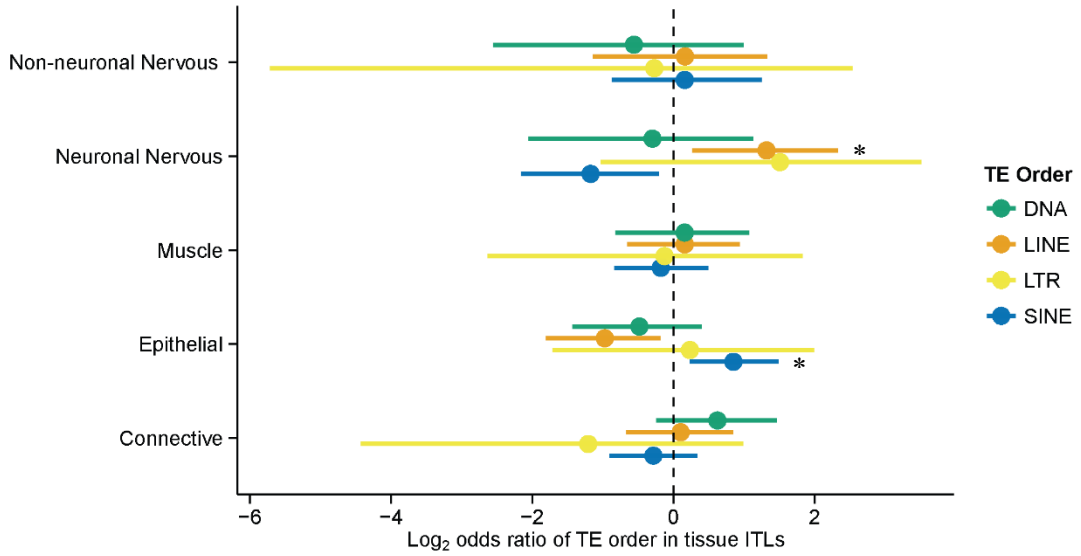


Figure 25. Comparison of enriched TE orders across tissue types.

Log2 odds ratio of each cell type expressing ITLs from DNA, LINE, LTR and SINE TE orders.Line represents 95% confidence intervals. Neuronal cell types are more likely to express LINE TE order ITLs (OR = 2.50, 95% CI = 1.20-5.03, P < 0.05) and epithelial cell types are more likely to express SINE TE order ITLs (OR = 1.80, 95% CI = 1.17-2.81, P < 0.05).

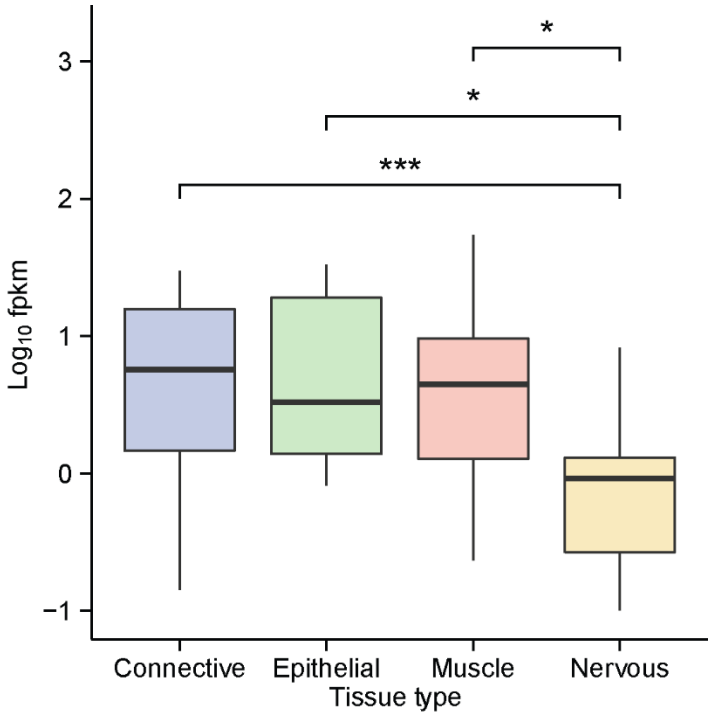


Figure 26. Expression of LINE ITLs is lower in nervous tissue cell types compared to other cell types.

Boxplot of log10 fpkm of ITLs in connective, epithelial, muscle and nervous tissue cell types. Double asterisks (**) indicate p-value < 0.01.

3.3 Discussion

By using ribosomal-depleted, high-depth, and stranded RNA sequencing and mapping to individual loci with our SQuIRE bioinformatics pipeline, we were able to resolve the genomic loci and structure of TE-derived transcripts and provide the most comprehensive analysis of TE expression in noncancerous human cells to date. While most TEs were expressed as part of pre-mRNA transcripts and 3' read-through, we have also identified extensive lncRNAs TE expression and individual TE locus (ITL) expression. We have differentiated between lncRNAs and ITLs by enriched TE phylogeny and patterns of expression across cell types, which was previously impossible to resolve by RT-PCR, microarrays, or mapping to TE consensus sequences. We have also discovered that ITL transcription varies by tissue type, particularly in nervous tissue cell types, which is complementary to previous evidence of active retrotransposition in the brain[116]. By precisely quantifying which single TEs are transcribed across normal cells, we are poised to better understand TE expression in disease.

3.4 Methods

Primary Cell Culture. The aortic endothelial cell type (HAEC) was obtained during cardiac transplantation and cultured as previously described[117]. All other non-neuronal primary cells were obtained from Lonza (Walkersville, MD). All primary cells were cultured according to manufacturer's specifications for no more than 6 passages (Supplementary Table 3). We isolated total RNA with the miRNeasy kit from Qiagen (Hilden, Germany) according to the manufacturer's protocol. We used the Agilent BioAnalyzer to assess RNA integrity and approximate RNA concentration.

Differentiation of human embryonic stem cells cells to cortical neurons. Human cortical neurons were differentiated from H1 hESCs (human embryonic stem cells, Wi Cell, Madison, WI) using our recently developed RONA (rosette-type neural aggregates) method[118]. Briefly, detached hESC colonies were grown in suspension in human ES cell medium without FGF2 (defined as

knockout serum replacement medium, KoSRM) in low attachment 6-well plates (Corning, Corning, NY), supplemented with Noggin (50ng/ml ; R&D systems, Minneapolis, MN) or Dorsomorphin (1μM, Tocris Bioscience, Bristol, UK) and SB431542 (10μM, Tocris Bioscience) from day 2 to day 6. Free-floating embryoid bodies (EBs) were attached and supplied with N2-induction medium (NIM) containing DMEM/F12 (Invitrogen, Carlsbad, CA), 1% N2 supplement (Invitrogen), 100 μm NEAA (Invitrogen), 1 mM Glutamax (Invitrogen), and heparin (2 μg/ml; Sigma, St. Louis, MO) from day 7 to day 16. Highly compact 3-dimensional column-like neural aggregates were collected and maintained as neurospheres in Neurobasal medium containing B27 minus vitamin A (Invitrogen), 1 mM Glutamax 1 day. For neuronal differentiation, dissociated neurospheres were maintained in neural differentiation medium containing Neurobasal/B27 (NB/B27, Invitrogen), BDNF (20ng/ml, PeproTech, Rocky Hill, NJ), GDNF (20 ng/ml, Peprotech), ascorbic acid (0.2 mM, Sigma), dibutyryl cAMP (0.5mM, Sigma).

Differentiation of human embryonic stem cells cells to dopaminergic neurons. H1 human embryonic stem cells were cultured using a standard protocol for inactivated mouse embryonic fibroblasts. Differentiation of hES cells to dopaminergic neurons was done as previously described[119]. Briefly, single hES cells were cultured on Matrigel-coated plates at a density of 40,000 cells/cm^2 in serum replacement media (SRM) containing FGF8a (100ng/ml, R&D Systems), SHH C25II (100ng/ml, R&D Systems), LDN193189 (10μM, Stemgent, Cambridge, MA), SB431542 (10μM, Tocris Bioscience), CHIR99021 (3μM, Stemgent) and Purmorphamine(2μM, Stemgent) for the first five days. Next, the cells were maintained in neurobasal medium containing B27 minus vitamin A, 1% N2 supplement along with LDN193189 and CHIR99021 for six days. In the final stage, a single cell suspension was made and seeded at a density of 400,000/cm^2 on polyornithine (15μg/ml) - and laminin (1μg/ml) - coated plates in neurobasal media containing B27 minus Vitamin A, BDNF (20ng/ml), GDNF (20ng/ml), TGFβ3 (1ng/ml, R&D Systems) ascorbic acid (0.2mM), dibutyryl cAMP (0.5mM) and DAPT (10μM, Tocris Bioscience) until maturation.

RNA-seq Preparation and Sequencing. The RNA libraries of all 31 cell types were prepared using the Illumina TruSeq Stranded Total Library Prep Kit with Ribo-Zero Gold (San Diego, CA) to provide stranded, ribosomal RNA depleted RNA. The libraries were sequenced on an Illumina HiSeq 2500, using 2 cell types per lane with paired-end 100bp reads. We generated a mean of 129,105,260 million paired reads per sample. In addition, two HUVEC RNA libraries were prepared to compare Illumina TruSeq Stranded Total Library Prep Kit with Ribo-Zero Gold with the TruSeq RNA Library Preparation Kit v2 kit that selects for poly-adenylated mRNA (Supplementary Table 5). These two preparations generated a mean of 160,158,655 paired reads per sample. Reads were assessed for sequencing quality using FastQC (http://www.bioinformatics.babraham.ac.uk/projects/fastqc/) and trimming was deemed unnecessary. RNA-seq data was submitted to the NCBI database under BioProject PRJNA321055.

SQuIRE Pipeline. We aligned the RNA-seq data using the SQuIRE Map tool, which incorporates STAR[52] and SAMTools[91] commands. The SQuIRE Map command used the "—read_length 100 –pthreads 8 –build hg38" parameters. We quantified TE expression using the SQuIRE Count tool, which outputs RNA expression estimates in counts and fpkms. The SQuIRE Count command used the "—build hg38 –pthreads 8 –strandedness 1 –EM auto" parameters.

TE analysis. We selected for TEs with a weighted confidence greater than 95%, more than 20 counts and an fpkm greater than 0.1 on either strand as high-confidence, expressed TEs. For TEs with expression on both strands, we treated each strand separately. The relationship between age (using divergence in base mismatches/kb from the consensus sequence as a proxy) was calculated using Pearson's correlation. Enrichment analysis of TEs in different gene and intergenic compartments was conducted using Fisher's Exact test, with p-values adjusted for multiple comparison's using the Benjamini-Hochberg procedure [120].

Grouping multiple TEs into transcripts in HUVEC. We used the GenomicRanges package to group TEs with more than 1 read count within 5kb of each other on the same strand. The expression

level of these transcripts is the average fpkm of all TEs within each grouping. Multi-TE transcripts were also compared to lncRNA coordinates in RefSeq, lncipedia and NONCODE databases [50],[114],[115]. We compared the weighted confidence values of TEs in known lncRNAs to those of other TEs using Student's t-test with Welch's approximation for degrees of freedom.

Identifying ITLs. We selected for TEs with a weighted confidence greater than 95%, more than 20 counts and a fpkm greater than 0.1 on either strand. We used the GenomicRanges package to group these high-confidence, expressed TEs within 40kb of each other on the same strand. The genomic coordinates of these groups were overlapped across 31 cell types. The expression level of these transcribed regions is the average fpkm of all TEs within each grouping. Out of 6,088 TEs without other high-confidence TE expression within 40kb, we further filtered out those that had expressed TEs with any weighted confidence (including < 95%) within 5kb on the same strand. The remaining TE loci coordinates were overlapped with Refseq, NONCODE and lncipedia databases of lncRNAs. TE transcripts with no nearby TE expression on any cell type within 5kb and that were not part of lncRNA annotation were considered individual TE loci (ITLs).

ITL Analysis. Enrichment of TE orders in genic, intergenic (lncRNA), and ITL TEs was conducted using Fisher's Exact test, with p-values adjusted for multiple comparison's using the Benjamini-Hochberg method. The ages of ITL and non-ITL TEs were compared using Student's t-test with Welch's approximation for degrees of freedom. The likelihood of a TE being flanked was calculated using Fisher's Exact test, with p-values adjusted for multiple comparison's using the Benjamini-Hochberg method. ITL overrepresentation among transcribed regions was also calculated with Fisher's Exact test. The number of ITLs expressed and their average fpkm level were compared between connective, epithelial, muscle and nervous tissues using pairwise Student's t-tests with Welch's approximation for degrees of freedom, with p-values adjusted for multiple comparison's using the Benjamini-Hochberg method. Enrichment and depletion of ITL TE orders among different tissue types was calculated using Fisher's Exact test, with p-values adjusted for multiple

comparison's using the Benjamini-Hochberg method.

Heatmap and clustering. The transcribed region fpkm expression of each cell type was divided by the region's mean fpkm expression across all cell types. Transcribed regions with a mean fpkm > 0.1 across all cell types were analyzed. We took the $\log_2$ of these values, and log values of regions that had no expression in a cell type ($\log_2 0 = -\infty$) were replaced by a negative number larger than the most negative $\log_2$ fold change (-15). The 400 transcribed regions with the greatest absolute sum of these $\log_2$ fold changes were used for clustering. The Manhattan distances between cell types and TE groups were clustered using the ward method. We used the heatmap.2 package in R (http://CRAN.R-project.org/package=gplots) to generate heatmaps with these clusters.

4. Landscape of Individual TE Loci Expression in Human Cancers

4.1 Introduction

Healthy cells maintain a steady state with protective mechanisms to prevent uncontrolled growth and propagation of cell damage[29,121,122]. Without those protective mechanisms, some of that damage can be mediated by unchecked transposable element expression [6,122]. Transposable elements (TEs) are genomic sequences that have generated self-propagating insertions throughout our evolutionary history, ultimately making up almost half of our human genome [1,2]. Even though only a few subfamilies of TEs are still capable of self-propagation, a larger subset retain intact sequences to enable transcriptional and translational activity[7,82]. Their transcription is inhibited by genome wide methylation of repetitive sequences, as well as other innate molecular mechanisms of TE inhibition[123,124]. Cancer has been linked with dysregulation of many of these mechanisms[124–128]. In addition, TE expression has been linked to tumorigenesis and cancer progression, and increased somatic insertions have been found in many cancer types. However, a comprehensive expression analysis of all TEs in cancer has not yet been elucidated at the locus level.

The locus-specific study of TE expression in cancer has been limited due to past difficulties with analyzing RNA-seq expression. Because TEs have replicated themselves by generating insertions throughout the genome, their sequences are repetitive. Short-read alignment from next generation sequencing can lead to an RNA-seq read mapping to multiple TE loci with shared sequence. Past RNA-seq approaches have either discarded these multi-mapping sequences or collapsed TE quantification to the subfamily level among multiple TE copies sharing high sequence identity[43–45,129]. To better quantify TE expression at the locus level, we developed SQuIRE which leverages a TE locus' uniquely mapping reads to estimate the probability of it originating a multi-mapping read that ambiguously maps to that and

other locations (described in Chapter 2). In addition, TE insertions can be located within the bounds of longer genes, so the expression of TEs and longer transcripts can be conflated. We have previously identified TEs that are transcribed as part of longer transcripts as in which the TE's expression is regulated independent of the TE's sequence, which we define as TE-extrinsic regulation. We have also identified examples of TE-intrinsic regulation, transcripts in which the TE's sequence is the driver of expression, either providing promoter sequence at the start of a longer transcript or as an individually transcribed locus (ITL). To distinguish these types of expression, we have expanded on the SQuIRE pipeline to distinguish these types of TE regulated expression in a high-throughput manner.

To investigate TE expression in cancer, we applied SQuIRE to 752 patient cases from the Cancer Genome Atlas (TCGA) project from 10 different cancer types[130]. To focus on TEs for which we have high confidence that their regulation is intrinsic to the TE sequence, we focused on ITL expression rather than including TEs in longer transcripts in our analysis. We compared ITL expression from TCGA cancer samples to 640 healthy samples from matched tissue types from Genotype Tissue Expression project (GTEx) [131]. In addition to comparing between tumor and normal and across cancer types, we also correlated differences in TE expression with clinical data.

Our findings are the most comprehensive analysis of TE expression in human cancer cells. Using SQuIRE, we are able to characterize TE expression at the locus level. This allows us to understand the genomic source and transcript type of differentially expressed TEs in normal tissues and cancer. Identifying differentially expressed TEs in cancer may provide sources of biomarkers and therapeutic targets as well as shed light on tumor pathogenesis.

4.2 Results

Organ	Type	Database	Number of Samples
Bladder	normal	GTEx	4
	BLCA	TCGA	68
Breast	normal	GTEx	110
	BRCA	TCGA	69
Colon	normal	GTEx	58
	COAD	TCGA	75
Brain	normal	GTEx	64
	GBM	TCGA	54
Kidney	normal	GTEx	15
	KIRC	TCGA	75
Liver	normal	GTEx	71
	LIHC	TCGA	58
Lung	normal	GTEx	79
	LUSC	TCGA	64
	LUAD	TCGA	53
Ovary	normal	GTEx	50
	OV	TCGA	52
Pancreas	normal	GTEx	58
	PAAD	TCGA	69
Prostate	normal	GTEx	66
	PRAD	TCGA	52
Stomach	normal	GTEx	65
	STAD	TCGA	63
		Total	1392

Table 11. Samples analyzed from GTEx and TCGA databases.

BLCA=bladder carcinoma. BRCA=breast carcinoma. COAD=colon adenocarcinoma. GBM=glioblastoma. KIRC=kidney renal carcinoma. LIHC=liver hepatocellular carcinoma. LUSC=lung squamous carcinoma. LUAD=lung adenocarcinoma. OV=ovarian serous cystadenocarcinoma. PAAD = pancreatic adenocarcinoma. PRAD=prostate adenocarcinoma. STAD=stomach adenocarcinoma.

To investigate the effects of malignancy on TE regulation, we sought to characterize the landscape of ITL expression in cancer samples. To compare between tumor and normal samples, we selected samples TCGA from 10 different cancer types. Because adjacent normal tissue may have molecular similarities with tumor that may not be detectable upon surgical resection, we compared TCGA samples with samples from GTEx from the same organ type. The number of samples used in this analysis for each organ is shown in We then ran the SQuIRE pipeline on each sample using the GRCh37/UCSC hg19 assembly for alignment, reference gene and TE annotation. This included aligning RNA-seq data to the genome using **Map**, quantifying TE expression with **Count**, and outputting bedgraph tracks for visualization with **Draw.** To identify TEs that are transcribed as ITLs, we developed a new module, **Flag**, which compares transcribed TE expression levels and coordinates from **Count** to RefGene annotation and assembled transcript annotation using StringTie. **Flag** then evaluates if the TE is expressed as part of a previously annotated or novel longer transcript, or if it transcribed as an ITL. More details of **Flag** are described in Methods. We subset the SQuIRE output for ITLs for further characterization of TE expression across our TCGA and GTEx samples. Because many ITLs are partially expressed, we further filtered our analysis for "full-length" ITLs that were transcribed at >90% of the annotated length.

ITL expression is not greater in tumor samples compared to normal samples.

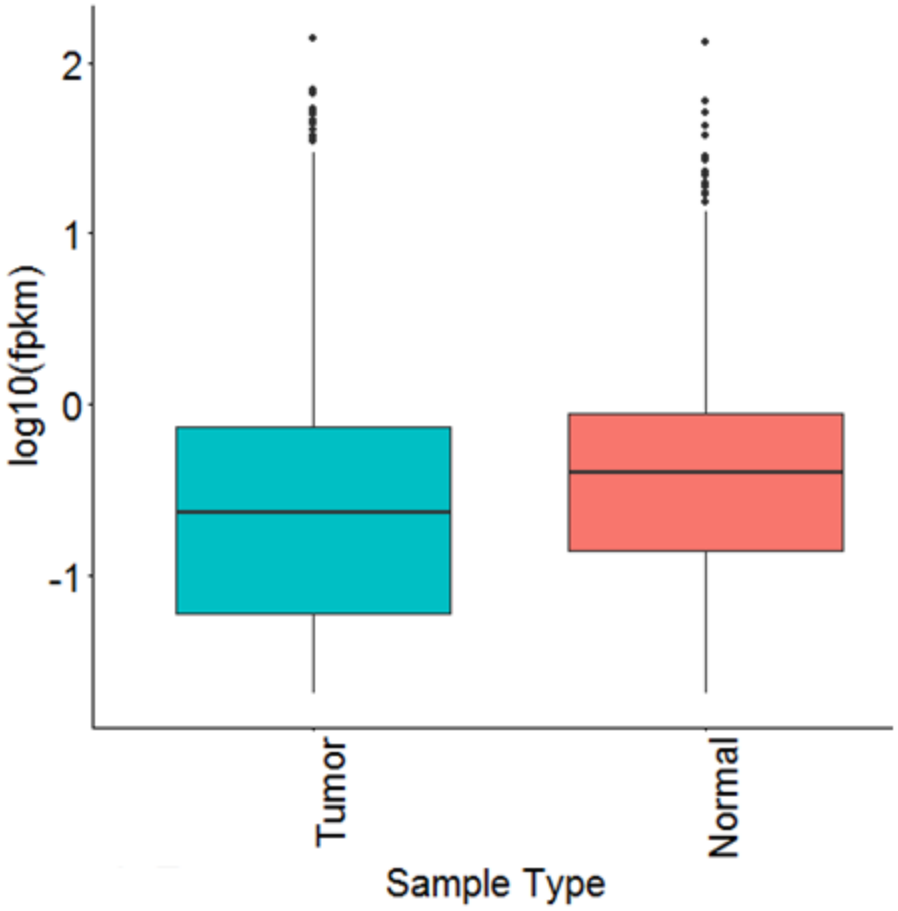
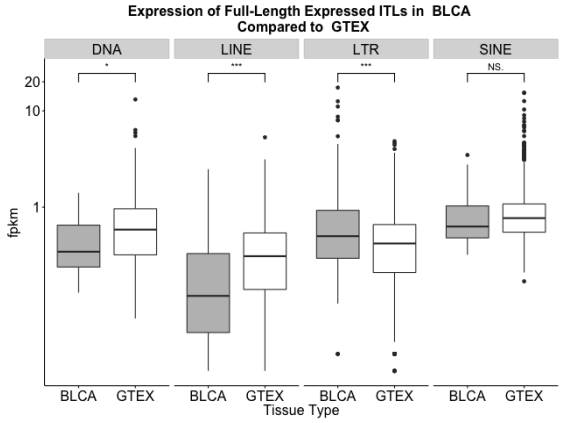
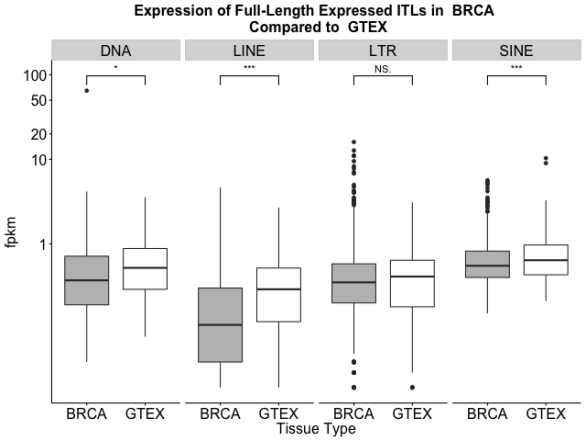
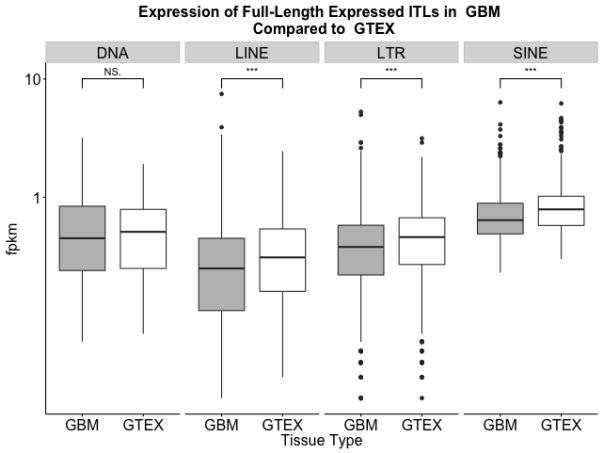


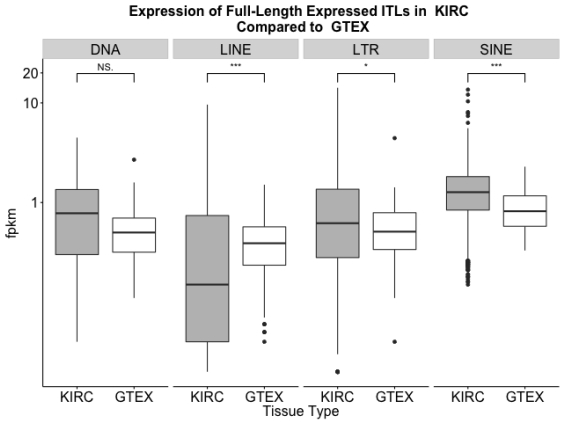
Figure 27. ITL expression levels as fragments per kilobase per million reads (fpkm) is not significantly increased in tumor samples.

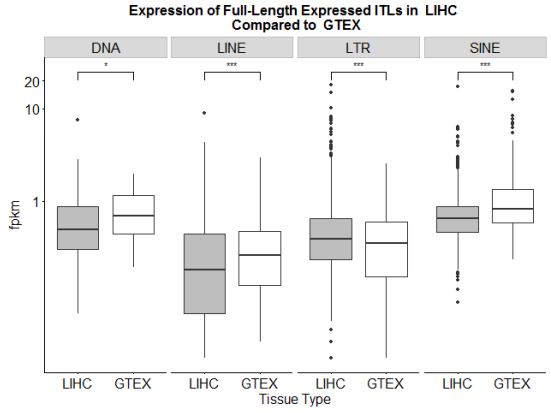
Boxplot comparison of mean expression level of ITLs between pooled cancer and pooled normal samples. ITLs are full-length as defined as having a transcript length > 90% the annotated length.

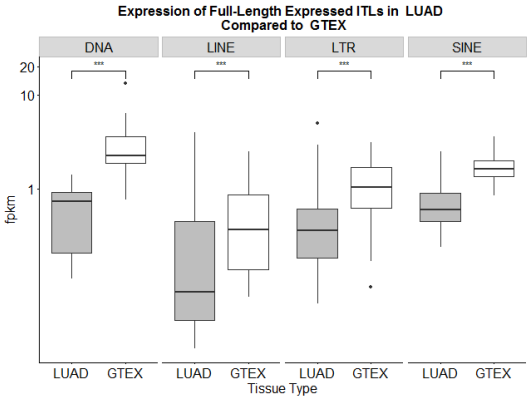

Expression of Full-Length Expressed ITLs in BLCA
Compared to GTEX
DNA
LINE
LTR
SINE
fpkm
Tissue Type
BLCA
GTEX


Expression of Full-Length Expressed ITLs in BRCA
Compared to GTEX
DNA
LINE
LTR
SINE
fpkm
Tissue Type
BRCA
GTEX


Expression of Full-Length Expressed ITLs in GBM
Compared to GTEX
DNA
LINE
LTR
SINE
fpkm
Tissue Type
GBM
GTEX


Expression of Full-Length Expressed ITLs in KIRC
Compared to GTEX
DNA
LINE
LTR
SINE
fpkm
Tissue Type
KIRC
GTEX


Expression of Full-Length Expressed ITLs in LIHC
Compared to GTEX
DNA
LINE
LTR
SINE
fpkm
Tissue Type
LIHC
GTEX


Expression of Full-Length Expressed ITLs in LUAD
Compared to GTEX
DNA
LINE
LTR
SINE
fpkm
Tissue Type
LUAD
GTEX

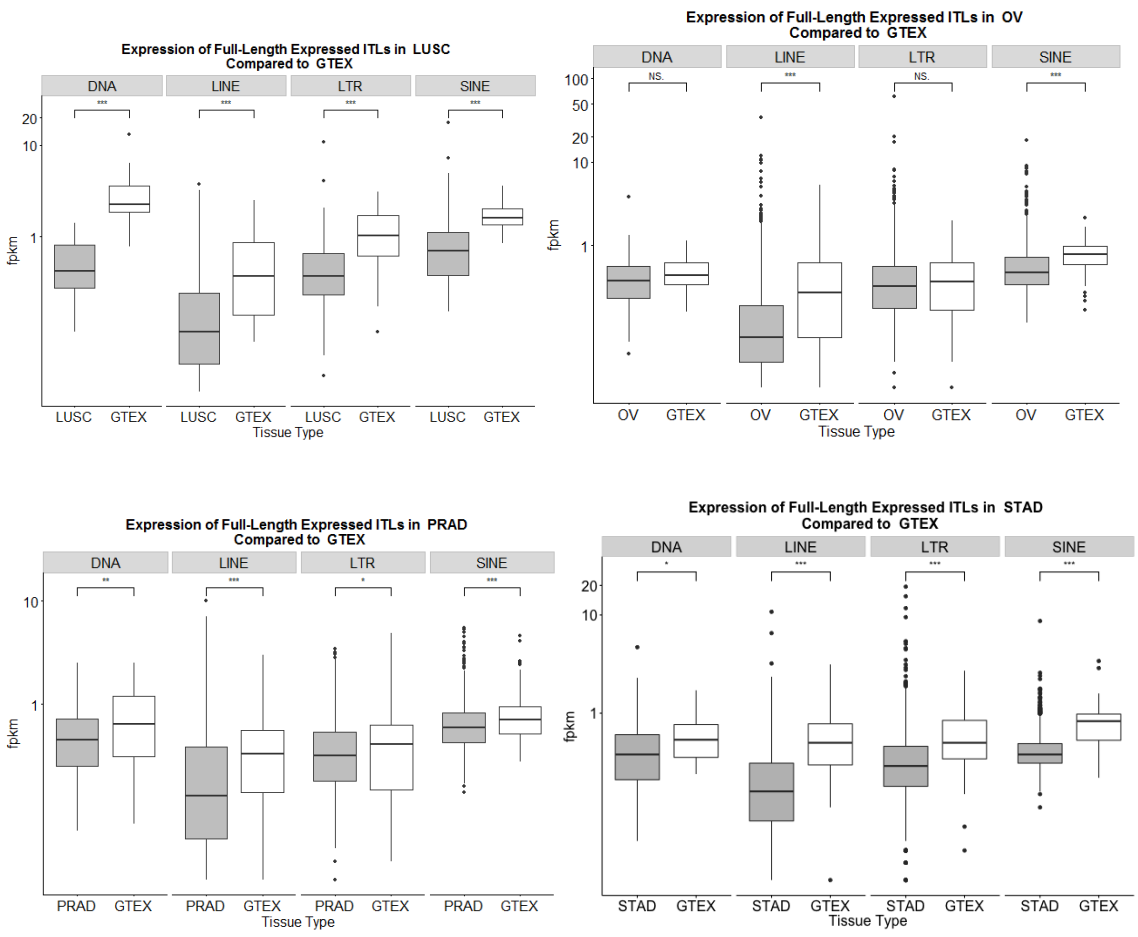


Figure 28. Comparison of ITL expression level between cancer and normal by TE order and cancer type.

*Boxplot of ITL expression as fragments per kilobase per million aligned reads (fpkm). ITLs are full-length as defined as having a transcript length > 90% the annotated length. Student's t-tests were performed comparing the particular cancer type and matched normal sample from GTEx. P-values were adjusted for multiple comparisons with an FDR < 0.05. *=p-value < 0.05. ** = p-value < 0.01 *** = p-value < 0.001. DNA=DNA transposon, LINE=Long interspersed nuclear element, LTR=long terminal repeat retrotransposons, SINE=short interspersed nuclear elements.*

To determine if ITL expression is de-repressed in cancers, we first evaluated the mean ITL expression of level as fragments per kilobase per million aligned reads (fpkm) to account for differences in library size across samples. To see if changes in TE expression were global, we first evaluated pooled cancer sample ITL expression compared to normal samples. We found that the mean ITL expression was not significantly different between cancer and normal samples when examining all cancer types (Figure 27Figure 28.). To see if this due to variations in cancer type that reduced the ability to detect significant changes in TE expression, we then compared mean ITL expression within each cancer type with normal samples from the same organ. We similarly found no differences in expression within each cancer type. Finally, to assess if changes in TE expression were specific to a subset of TE types, we parsed ITLs by their TE order phylogeny and compared between cancer and normal samples from the same organ. With this method of analysis we found that in fact, TEs within the same order exhibited higher mean levels of expression in normal samples than in cancer samples in several organ types (Figure 28.).

Tumor samples are more permissive for TE expression of distinct ITLs

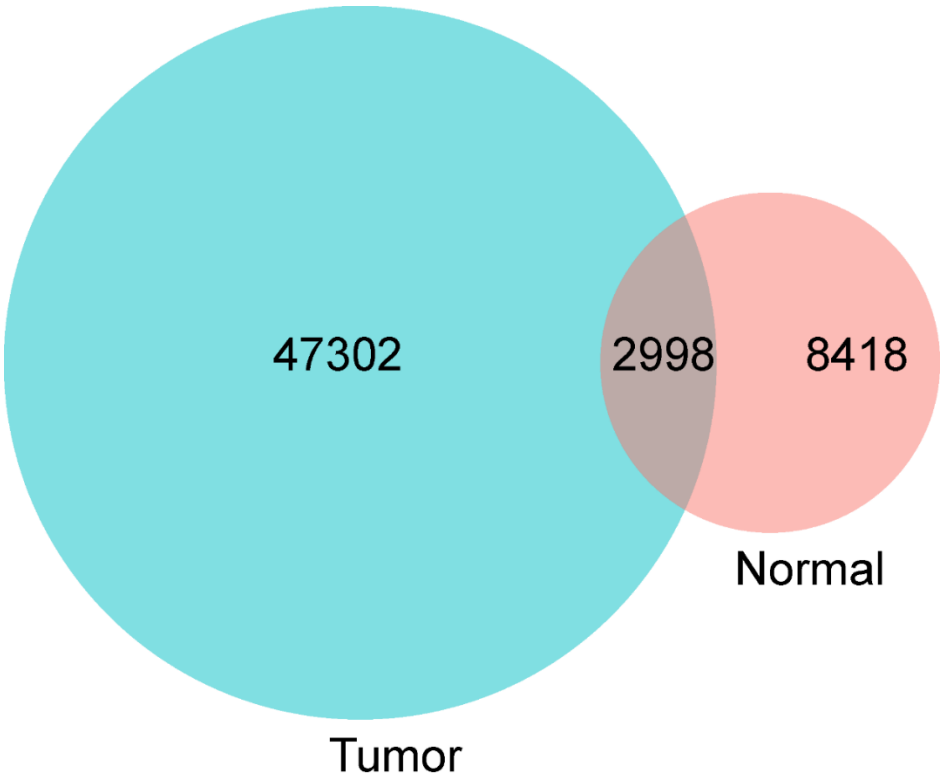


Figure 29. Distinct ITLs expressed in tumor and normal samples.

Venn diagram of numbers of distinct TE loci expressed as individual transcripts (ITLs) exclusively in tumor samples, exclusively in normal samples, and in both tumor and normal samples.

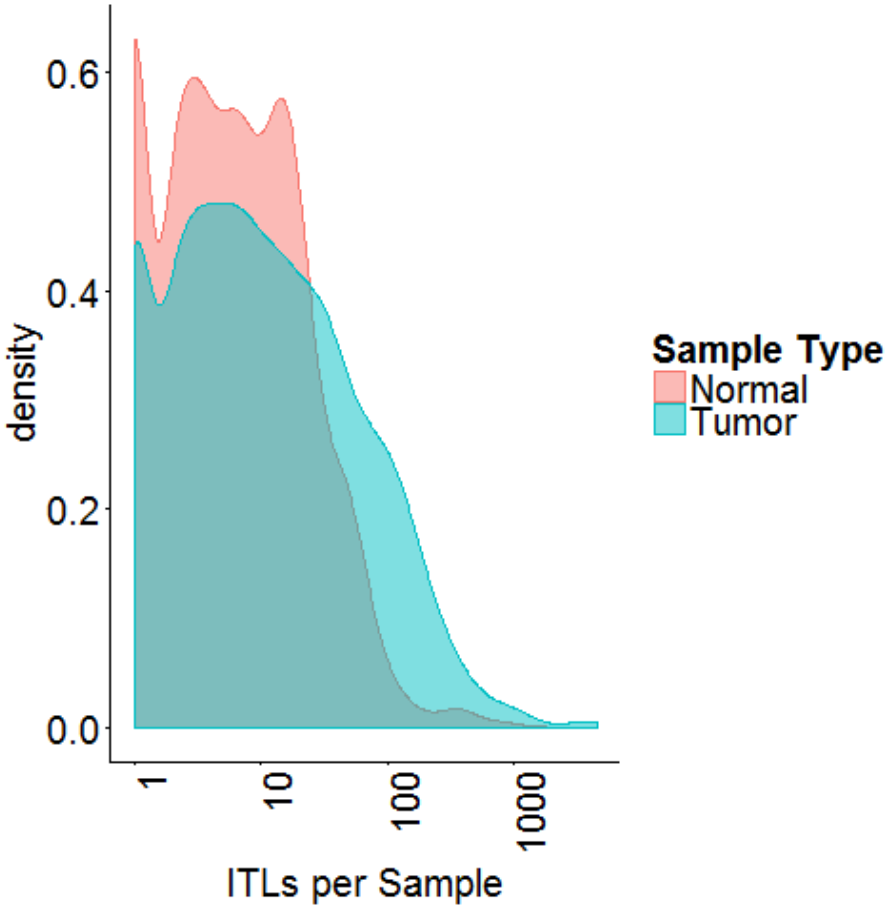


Figure 30. A greater fraction of tumor samples express high numbers of ITLs per sample.

Density plot of permissiveness, or ITLs per sample expressed in tumor and normal samples.

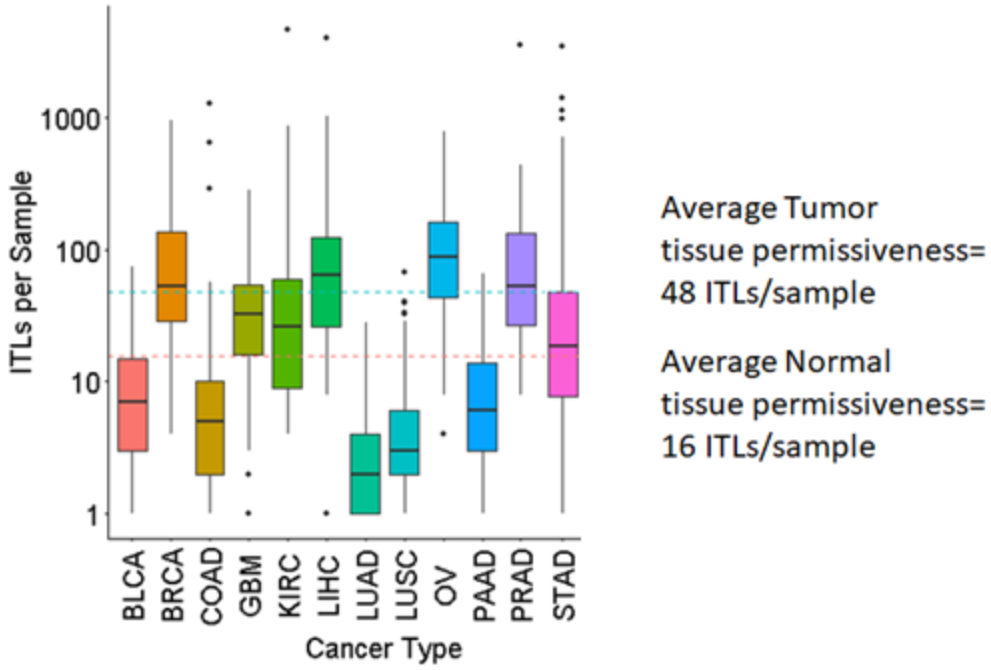


Figure 31. ITL permissiveness varies across cancer types.

Boxplot comparison of ITL permissiveness across cancer types. Blue horizontal line represents average number of ITLs per sample in tumors, and red horizontal line represents average number of ITLs per sample in GTEx samples.

Because we were expecting derepression of TE expression and increased ITL expression levels, we next sought to determine if the loci expressed in cancer were distinct from the loci expressed in normal samples. To assess this, we identified the different ITLs expressed in tumor and normal samples and evaluated how often they were expressed in both tumor and normal samples (Figure 29). Among the 47,302 ITLs expressed in our cancer samples, 2,998 (6.3%) that were also expressed in normal samples. Conversely, these ITLs expressed in tumor and normal samples represent 35% of the 8,418 ITLs expressed in normal samples. This suggests that while the overall expression levels of ITLs does not increase in cancer, the number of distinct loci are increased. To support this, we analyzed the number of ITLs expressed per sample and normalized for the number of cancer and normal samples. We used the number of ITLs expressed per sample as a proxy for "permissiveness" for TE expression. This is depicted in the density plot shown in Figure 30. As expected, permissiveness is increased in cancer samples: 48.6 ITLs per sample in cancer compared to 15.5 ITLs per sample in normal samples (Student's t-test, p-value <0.05). When comparing across cancer types, we found that particular cancer types exhibited greater permissiveness than others; in particular breast carcinoma (BRCA), liver hepatocellular carcinoma (LIHC) and ovarian carcinoma (Figure 31).

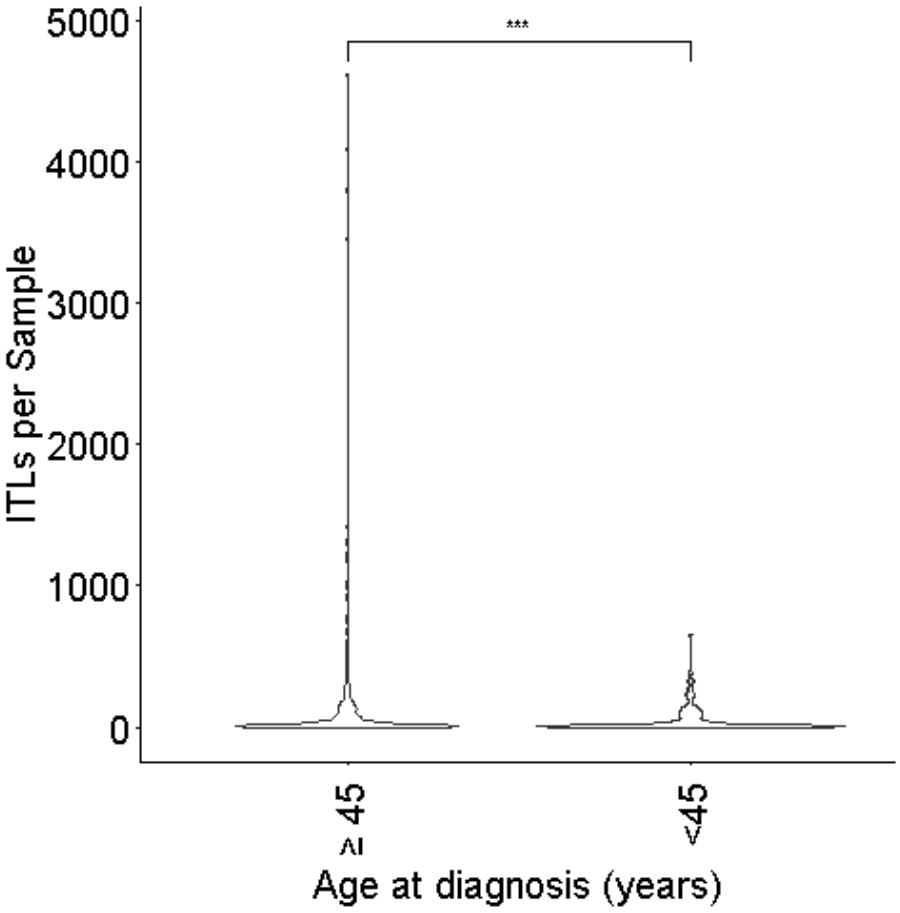


Figure 32. Older patients have greater permissiveness to TE expression.

*Violin plot with x-axis patients who are at least 45 years old or less than 45 years old and y-axis number of ITLs per sample as a proxy for permissiveness to TE expression. *** = Student's t-test, $p<0.001$. Number of patients ≥ 45 years old: 1,103, Number of patients <45 years old: 1,103.*

TCGA provides patient clinical data for their cancer samples; all cases had accompanying patient age and sex. To evaluate if these variables correlated with ITL permissiveness. Among cancer types with representation among both men and women, we did not find a significant difference in permissiveness between sexes. When we plotted patient age with ITLs per sample, we found a marked increase in ITLs among patients > 45. We performed a Student's t-test and found that the mean ITLs per sample was significantly greater in patients > 45 years old compared to younger patients (> 45: 95 ITLs/sample, < 45: 51 ITLs/sample, p-value 0.000875), as shown in Figure 32. Similarly, performing a Fisher's Exact test confirmed that highly permissive samples (> 100 ITLs per sample) were enriched in patients at least 45 years old, with an odds ratio of 85.16 (95% CI 58.37-124.84, p-value <0.0001). We performed an ANOVA to determine if this relationship was dependent on cancer type, and found no interaction between age and cancer type for ITL permissiveness.

Tumor ITLs are more likely to be LTRs and LINEs.

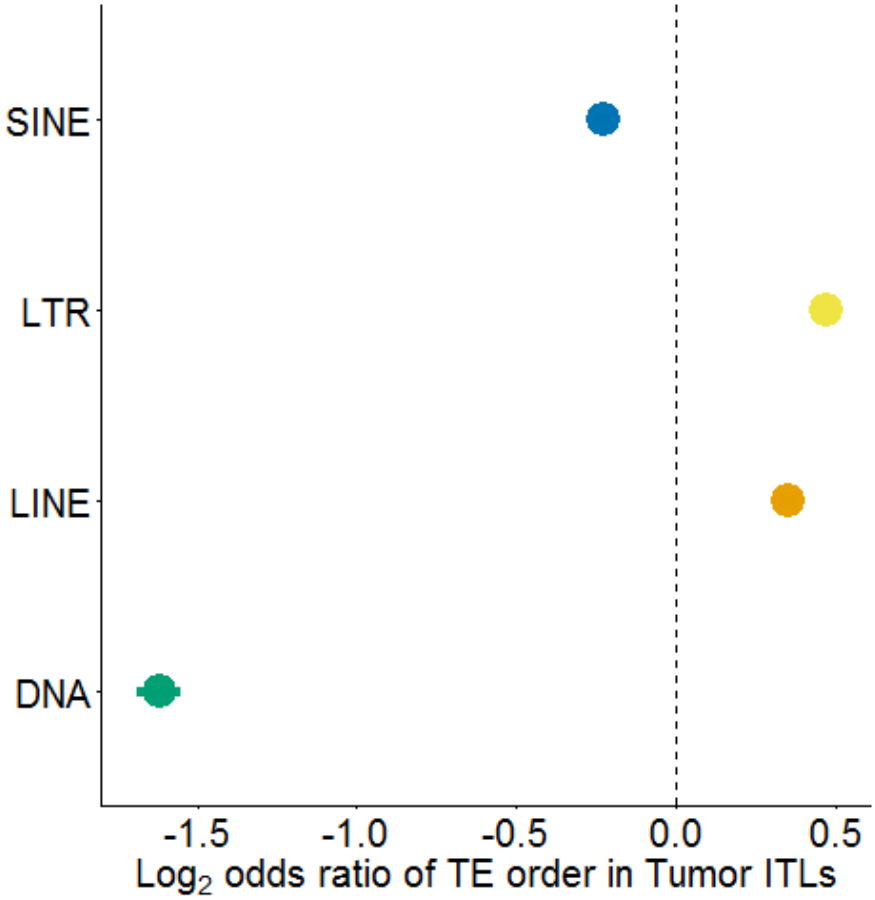


Figure 33. Likelihood of a tumor-specific ITL to belong to one of the above TE orders compared to their presence in genome.

Each point represents the odds ratio of ITL belonging to particular TE order from Fisher's Exact Test compared to the representation of all ITLs in all cancer samples. Horizontal line represents 95% confidence interval. P-values were adjusted for multiple comparisons for a false discovery rate of <0.05.

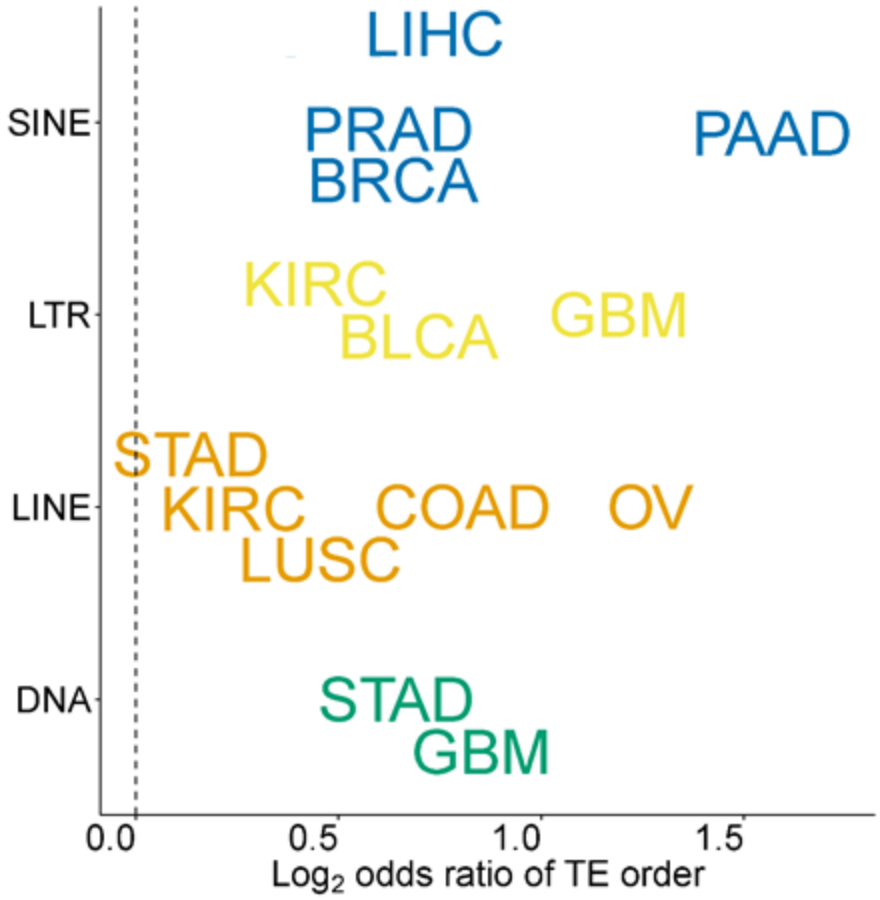


Figure 34. Enrichment of TE orders across different cancer types.

Center position of each cancer type text represents the odds ratio of ITL belonging to particular TE order from Fisher's Exact Test compared to the representation of all ITLs in all cancer samples. P-values were adjusted for multiple comparisons for a false discovery rate of <0.05. Only TE orders with significant enrichment in a cancer type are depicted here.

To assess if particular types of ITLs are enriched in cancer compared to normal samples, we categorized ITLs by their TE order and performed a Fisher's Exact Test comparing if particular TE orders were enriched in cancer among all TEs expressed in both tumor and normal samples (Figure 33). We found that LINE and LTR ITLs were more likely be expressed overall in cancers. We wanted to see if this enrichment varied for various cancer types and performed repeated Fisher's Exact tests for each cancer type, adjusting for multiple comparisons for a false discovery rate of 0.05 (Figure 34). We found that LINE ITLs were enriched in colon cancers relative to LINES in the genome, and LTR ITLs were enriched in glioblastoma. However, SINE ITLs were enriched in pancreatic adenocarcinoma.

Frequency of ITL expression across samples

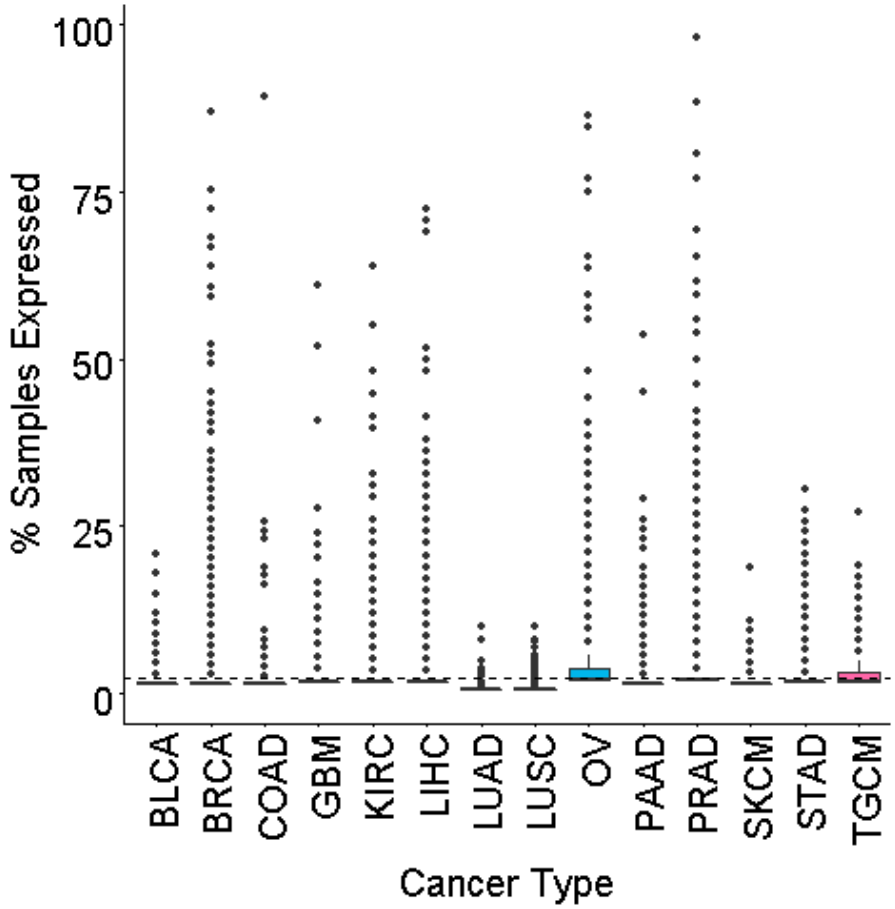


Figure 35. Tumor-specific ITLs are rarely expressed across all cancer types.

Dotted line represents mean % samples expressed per ITL across all cancers.

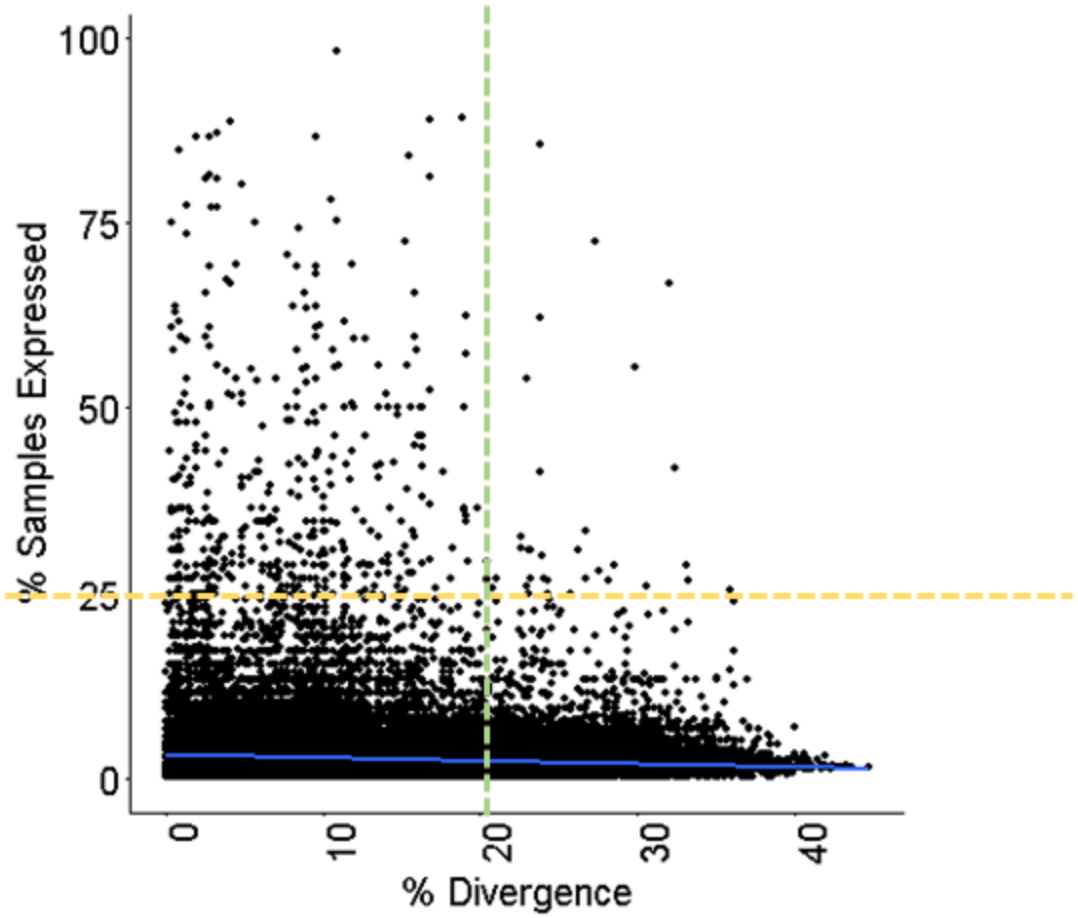


Figure 36. Percentage (%) of samples with ITL expression verses the divergence of the ITL from its consensus sequence. The proportion of samples showing expression is inversely correlated with % divergence of the ITL from the consensus sequence.

ITLs that are expressed in > 25% samples (horizontal yellow line) appear concentrated among relatively younger ITLs with < 20% divergence (vertical green line).

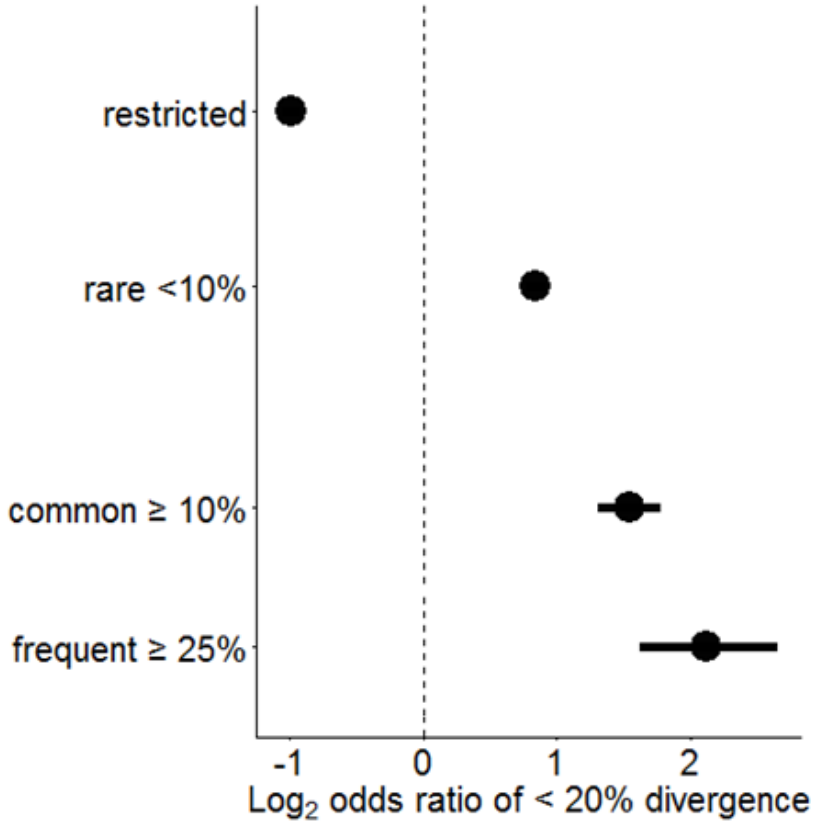


Figure 37. ITLs that are expressed in more than 1 sample are more likely to be < 20% divergent from the subfamily consensus sequence.

Fisher's exact tests were performed to evaluate odds ratio that a particular ITL is <20% divergent from the consensus sequence. This enrichment is greatest among ITLs that are expressed in >25% of samples.

We next ascertained if tumor-specific ITLs were expressed more frequently across samples than normal-specific ITLs. We found that while both tumor-specific and normal-specific ITLs were generally sample-specific (normal mean: 1.72 samples per ITL, tumor mean 1.88 samples per ITL), a greater percentage of tumor-specific ITLs (24.74%) were expressed in more than one sample compared to normal sample-specific ITLs (19.71%). The mean percentage of samples expressed per ITL was low for tumor-specific ITLs among all cancer types (Figure 35). To assess if the ITLs expressed in permissive samples were more likely to be expressed in multiple samples, we performed Fisher's exact test. We did not find any interaction between sample permissiveness and ITL frequency. This suggests that the mechanisms repressing ITL expression are not targeting particular loci. To assess if how often an ITL was expressed in multiple samples correlated with how young the TE locus was, we correlated number of samples expressed per ITL with % divergence from the consensus sequence as a proxy for age (

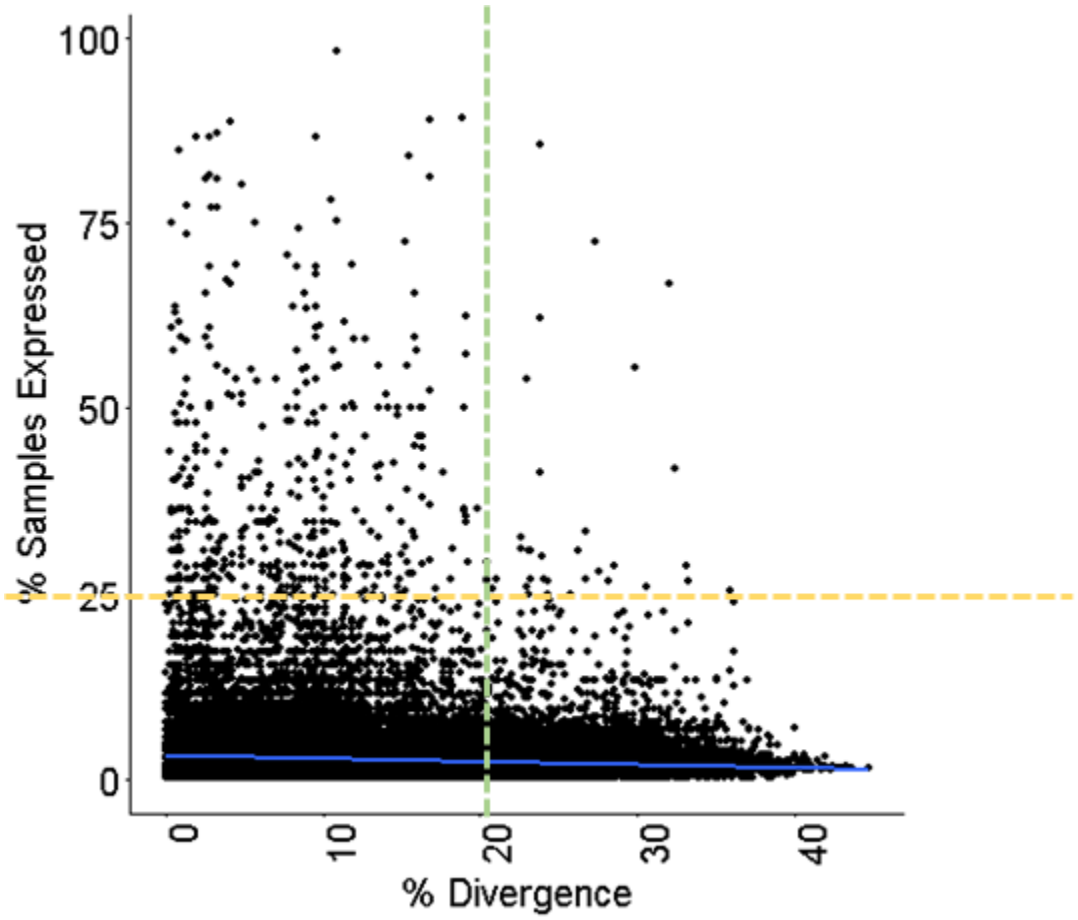


Figure 36). We found the frequency of expression inversely correlated with the age (r=-0.11, p-value < 2.2 e-16). In particular, we observed more frequently expressed ITLs among TEs with less than 20% divergence from the consensus sequence, which we confirmed by performing a Fisher's exact test for enrichment of these younger TEs among different ITL expressing frequencies, while accounting for multiple comparisons using a false discovery rate < 0.05. We categorized ITL frequency as restricted (1 sample), rare (<10% of samples, > 1 sample) common (10-25% of samples) and frequent (at least 25% of samples). We found that the enrichment for TEs with < 20% divergence was greatest among the "frequent" category of ITLs expressing in > 25% of samples (Figure 37). This suggests that younger elements are more likely to retain sequence required for intrinsic transcriptional activity.

4.3 Discussion

Malignancy is characterized by the disruption of multiple processes that regulate transcription[31,83,124–128,132]. Many of these processes are involved in suppressing TE expression[15,40,121,125]. We assessed the differences in TE expression between cancer and normal cells. In particular, we focused on the expression of individual TE loci to assess TE-intrinsic regulation of expression rather than TE-extrinsic regulation of TE-containing longer transcripts.

Here we demonstrate that changes in TE expression in cancer samples center around the distinct loci that are expressed compared to normal samples. We show that most ITLs expressed in tumors are cancer-specific and patient-specific. Such locus-specific regulation has been missing with past subfamily-level analyses of TE expression in cancer.

Moreover, we found that the ITLs that are expressed in multiple cancer samples are more likely to be younger insertions with less sequence divergence from the consensus sequence. However, the threshold for divergence (<20%) , equivalent to ~100 million years old, is a broader definition of a young element among the transposable element field, which largely focuses on retrotranspositionally active TEs[42].

We were surprised to find that overall TE expression levels (as fpkm) is not globally increased in cancer compared to healthy samples. Instead, we find enhanced permissiveness to TE expression represented by the increased numbers of isolated loci expressed per sample in cancer samples. This suggests that dysregulation of TE transcription is largely reflected by the presence or absence of TE expression, rather than differences in expression levels. We further determined that TEs are more likely to be expressed in older patients. This supports findings that global hypomethylation in cancer is age-dependent [133]. Our findings illuminate patterns of expression of TEs in cancer at the locus level.

4.4 Methods

RNA-seq Data

We obtained a total of 752 tumor of 10 of the most common cancer types from The Cancer Genome Atlas (TCGA) database using API download protocol. For a true normal control, we used 640 RNA-seq samples from the Genotype Tissue Expression (GTEx) database, downloaded with the dbGaP platform. TCGA samples were converted from the downloaded BAM format to raw FastQ files using BEDTools BAMtoFastQ[90]. These samples are all unstranded and paired-end. TCGA RNA-seq sample read lengths ranged from 48-78 bp and GTEx samples were consistently 76 bp. We selected for RNA-seq library sizes greater than 20 million bp.

SQuIRE RNA-seq Pipeline

We performed SQuIRE analysis in three steps. We used **Map** to align RNA-seq reads to the UCSC GRCh37/hg19 genome with STAR[52]. We then used **Count** to quantify the expression of individual TEs. From the locus-specific **Count** outputs, we used **Flag** to identify individually transcribed loci (ITLs). Details of the **Flag** algorithm are futher described below.

SQuIRE Flag algorithm

We developed the SQuIRE **Flag** module to distinguish transcripts from individual TE loci (ITLs) from pre-mRNA and lncRNA transcripts. It incorporates the assembler software StringTie [53].

Transcript assemblers use a multi-step process that first assembles transcripts on each sample, then merges the novel annotations together, and finally runs the assembly again using the new annotation. SQuIRE **Flag** follows this process while adding analysis steps to compare TE expression with Stringtie-assembled transcript expression within each sample and then compare transcript types across samples.

In assembly step 1, **Flag** uses StringTie with UCSC's RefGene annotation for a guided assembly of RNA-seq transcripts, to be run in parallel on all samples. We used all default settings except we allowed a greater percentage of multi-mapped reads (-M 0.99, default = 0.95), and allowed a wider gap between reads to be processed together (100 instead of 50). These changes allow for greater sensitivity in including TE-containing transcripts. In assembly step 2, **Flag** uses the Stringtie "transcript merge mode" to combine the output gtfs from step 1 into a single merged gtf. We used higher stringency thresholds for this mode to reduce the identification of spurious transcripts, requiring greater than 2.5X coverage (default =0), greater than 1 fpkm (default = 0), and a minimum isoform fraction greater than 0.01. Assembly step 3 then uses this merged gtf to rerun a guided Stringtie transcript assembly on each sample. SQuIRE next compares the resulting transcripts with expressed TE count information from the SQuIRE **Count** outputs using BedTools Intersect and labels them as potential ITLs, lncRNA, or pre-mRNA. Assembly Step 4 then compares the transcript labels of TEs across all samples, so that a TE that was labeled as an ITL in one sample but an lncRNA or read-through in another sample, is re-labeled as being transcribed from a longer transcript.

SQuIRE **Flag** identifies transcript type by intersecting transcribed TE coordinates with RefSeq genes and the Stringtie transcript assembly. Expressed TEs that overlap with RefSeq coordinates (on the same strand, if stranded) are labeled as lncRNA or pre-mRNA. TEs that are expressed within 10kb of the nearest RefSeq gene are considered read-through. TEs that are downstream and sense (if stranded) to the nearest gene ("DoG readthrough), and TEs that are upstream and antisense (if stranded) to the nearest gene ("antisense_readthrough"). Novel Stringtie

transcripts that are >10kb from the nearest gene are compared to TE coordinates. Transcripts that extend > 1 kb beyond the TE annotation or contain multiple TEs of different orders (a LINE and an LTR for instance) are labeled as potential lncRNA. TEs of the same order transcribed within 1kb of each other may be separate entries of the same element (for example the internal and LTR sequences of the same endogenous retrovirus), so they are considered together. Similarly, if multiple TEs are transcribed within 10kb of each other, they are labeled as potential lncRNA. SQuIRE **Flag** thus labels a TE pre-mRNA, lncRNA, or read-through if they are transcribed from 1) an annotated gene 2) a longer novel Stringtie assembled transcript 3) a collection of nearby TEs of different superfamilies. Transposable elements of the LINE, SINE, LTR, DNA, and SVA superfamilies that are not part of these longer transcripts are labeled as potential individual TE loci (ITL) transcripts.

Flag outputs

For each sample **Flag** provides the Stringtie outputs of a gtf file from steps and 1 and 3, abundance file describing fpkm, tpm, and coverage of each stringtie transcript. **Flag** also creates SQuIRE-specific files "preflag" file with intermediate data used to label transcript type, a "spliced TE" file providing intron coordinates that overlap with TEs, and a "flag file" that provides SQuIRE **Count** data, coordinate and expression information of the nearest downstream and upstream expressed gene, the coordinates of any overlapping Stringtie transcript, the transcript type, and whether the TE is spliced. At the project level from comparing all samples, **Flag** provides the merged gtf produced by Stringtie, a combined flag file that gives final transcript and splicing labels for all samples, and Stringtie generated gene- and transcript- level count matrices.

TE Analysis

All post-SQuIRE analysis was performed in R[92] with RStudio.[134]. To select for the expressed, full-length ITLs, we further filtered the TEs for transcripts with > 10 counts, > .01 fpkm, and > 90% of the RepeatMasker reference length.

For TE analysis, we evaluated statistical significance using T tests and R package ggsignif and created figures using R package ggplot2.[135] To quantify permissiveness, we counted the number of ITLs expressed in each sample. To identify common ITLs across samples, we counted the number of samples that express each ITL within the tissue types. We used the fpkm of ITLs in each sample to compare ITL expression between tumor and normal samples. We investigated the enrichment of orders and subfamilies ITLs in each tumor type compared to all samples using the Fisher's exact test.

5. Conclusions

The role of transposable elements (TEs) in the human genome has long been underestimated. Indeed, the software RepeatMasker, which this study used to identify TE genomic locations, was originally used for 'masking' TE sequences from genomic analyses[51]. Nevertheless, comparative genomics studies have shown that TEs are major evolutionary contributors to the composition, diversity, and function of our genome. The transposition activity of TEs presents a double-edged sword to its host. Despite their potential to create deleterious insertions, TE mobilization can generate gene duplications, add new functional domains to coding and noncoding genes, and provide novel cis-regulatory sequences to modulate nearby gene activity[17,19,97,136]. Thus, rather than being "dark matter", TEs are a shining light critical for a deeper understanding of our genome.

Yet, despite these genomic contributions and comprising almost half of our DNA, prior to this study a global picture of TE transcription had not been performed. TE biologists have primarily focused on young, polymorphic elements (L1HS and young *Alu*s). While retrotransposition activity *a priori* requires TE transcription, this ignores older TEs with disrupted coding sequence that may still retain intact promoters allowing for the generation of RNA transcripts. TE transcription has also been studied orthogonally via evaluation of their epigenetic regulation, identifying TE sequence motifs that promote [32,98] or repress TE transcription [27,28]. Such studies that have focused on autonomous transcription of TEs are largely siloed from studies of TEs in other contexts. Despite being major contributors to long noncoding RNAs (lncRNAs) [97,102], the relationship between lncRNAs transcription and autonomous TE regulation has not been thoroughly understood. Similarly, the retrovirology field has progressed the study of endogenous retrovirus independent of the transposon field, resulting in conflicting nomenclature for LTR retrotransposons.

The siloing of TE-related fields contributes to an ongoing Catch-22 in understanding TE transcription. A unified understanding of TE transcription has been hampered by the limited tools for

quantifying repetitive sequences in RNA-seq data. Studies of TEs have therefore focused on orthogonal indicators of TE transcription, e.g. retrotransposition, epigenetic markers, and LINE-1 and LTR protein expression[18,66,124,132,137,138]. In turn, the lack of studies investigating how TEs are transcribed has limited efforts in developing software to study TE transcription, which currently relies on genomic annotations of TE insertions. However, due to the variety of genomic contexts of TE insertions, the genomic boundaries of a TE labeled by RepeatMasker may not fully encompass the transcript containing the TE sequence.

Transposable element transcription has been difficult to study due to both the repetitive nature of these sequences and the current state-of-the-art technology of RNA-seq. Whereas the exons of most coding genes are unique to the entire genome, TEs can share sequences between multiple insertion copies. Because of technical limitations in sequencing the entire length of RNA transcripts, RNA-seq involves fragmenting RNA and then partially sequencing the ends of fragments. The RNA-seq reads are then bioinformatically mapped back to the genome to determine the originating gene, and expression levels are inferred by quantifying the number of reads. This quantification is more difficult, then, if many (if not all) of a TE transcript's reads are not mappable to a precise location in the genome.

Previous approaches to analyzing TE transcription have sidestepped such difficulties in one of three ways. Some have discarded all multi-mapping reads and inferred transcription only from RNA-seq reads to uniquely map to a TE locus. However, because TEs have varying divergences such that some TEs have more uniquely alignable sequence than others, this method cannot provide a quantitative picture of TE transcription without biasing against the representation of young TEs. Conversely, the TE software RepEnrich [43] and TEtranscripts [44] have discarded locus-specific information, instead aggregating TE read counts at the subfamily level. Despite allowing quantitative comparisons of TE expression, the details of TE expression are obscured without position information. Conclusions from using these software packages therefore cannot distinguish between

changes in TE transcription due to TE-specific regulation and changes in expression of TE-containing genes that are independent of TE sequence. A third approach used by TEtools [45] attempts to resolve alignment ambiguity in a multi-mapping TE-derived RNA-seq read by designating a single locus to which the read aligned. However, this approach can yield many false positive alignments, particularly for younger TEs with >99% similarity between copies. Furthermore, this underestimates the impacts of RNA-sequencing errors, single nucleotide polymorphism (SNP) variation among TE loci, and structural variations in non-reference TE polymorphisms that can misattribute a read to the incorrect locus.

We developed the pipeline SQuIRE (Software for Quantifying Interspersed Expression) to build on, extend, and improve these approaches to quantify TE expression at the locus level. Because of the varying uniquely aligning sequence content among TEs, SQuIRE's quantification algorithm normalizes for this, allowing for more accurate comparisons between different TEs. This improvement in accuracy distinguishes SQuIRE from RepEnrich and TEtranscripts, which used similar methods to assign multi-mapping reads to TE loci, but without normalization or the use of uniquely aligning reads. Furthermore, in determining the false positive rates for multi-mapping reads, we tested several aligners to find an optimal approach for TEs. Expecting that the aligner would be integral to accurate quantification of TEs, we incorporated the aligner and the optimal parameters into the pipeline for improved reproducibility across studies that use SQuIRE. In this work, we have demonstrated that the tools provided by SQuIRE are more accurate than past software. We have also gone to great lengths to make SQuIRE a complete, start-to-finish pipeline that is accessible to biologists and user-friendly.

Furthermore, unlike past approaches, SQuIRE does not assume that TE transcripts are bounded by their genomic annotation in RepeatMasker. Instead, SQuIRE's output indicates if the TE's transcript extends beyond its annotated borders or is shorter than annotated. Combining that information with neighboring expression data allows for a true picture of the composition of TE

transcripts. It allows us to detect spliced TE-containing RNAs and partially-expressed TEs, like those driven from non-canonical internal promoters. Indeed, when we applied SQuIRE to stranded, deeply sequenced RNA data from 31 low-passage, primary cell lines, we found that individual TE loci (ITL) transcripts are distinct from lncRNAs and mRNAs. In addition, characterizing TE-containing transcripts not only expanded our understanding of the transcript lengths of individual TE loci, but we also found that using SQuIRE updated the annotation of lncRNAs, identifying novel and longer-than-reported transcripts, and mRNAs, identifying downstream-of-gene transcription. We confirmed previous reports that lncRNAs are particularly enriched for long terminal repeat (LTR) retrotransposons, which include endogenous retroviruses[138]. This stresses the importance of an updated TE transcriptome annotation that can be integrated with current mRNA, lncRNA and retroviral annotations and nomenclature.

In studying ITL transcription in more depth, we found that cell lines from epithelial and neuronal origins were more permissive for ITL expression compared to muscle and connective tissue cell lines. Although our analysis of ITLs is limited and may not comprehensively include all independent TE transcription, in selecting only TEs that are not transcribed near other expressed TEs, genes or lncRNAs, we can be reasonably confident that our examples are not due to background transcription. Yet, in examining ITL expression levels, we found that the mean expression level of ITLs was lowest in neuronal cells. This seeming contradiction may be an illustration of the balance of TE transcription regulation by the host cell, to harness evolutionary beneficial functional domains, but to also mitigate individually harmful RNA intermediates and DNA insertions. We have demonstrated that TEs are expressed primarily as components of lncRNAs and pre-mRNA transcripts. Increased ITL transcription in neuronal and epithelial cells may be a byproduct of host-upregulated transcription from TE-derived promoter sequence. This may result in increased transcription of ITLs, lncRNAs, and pre-mRNAs sharing these promoter sequences. However, this increased TE permissiveness may

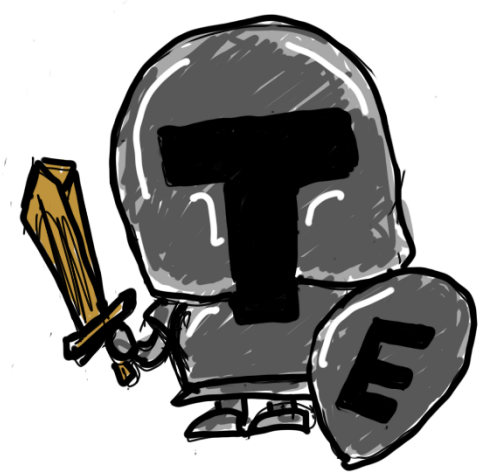
result in RNA intermediate sequences that trigger host responses to restrict harmful TE transcription and retrotransposition.

The disruption of this regulatory balance between TE expression and regulation may be critical to understanding TE-related disease processes. Indeed, our findings of ITL expression in mouse and human counters the dogma that TE expression is tightly regulated in healthy somatic cells. Instead, our findings suggest that the regulation of TE expression is TE type and locus-specific. We find that this sample-specific pattern persists for most ITLs in cancers. However, we observed that younger ITLs were more likely to be expressed in multiple cancer samples. Conversely, we determined that cancer samples that were more permissive and expressed greater ITLs per sample were more likely to be from older patients, suggesting that age is related to deregulation of TE expression. Further work needs to be done to investigate the sequence and contextual differences between TEs of the same family that have different expression patterns. This may be best done in disease settings such as cancer and other diseases featuring disrupted TE regulatory mechanisms (DICER[121], methylation [139], RNA editing[89]), as well as animal models with different patterns of TE expression. SQuIRE thus is a powerful tool that can enhance our understanding of the interplay between gene and TE expression in health and disease.

6. Appendices

Appendix A. SQuIRE website

Software for Quantifying Interspersed Repeat Expression



Installation

We recommend using Conda for SQuIRE installation.

Conda is a package manager that installs and runs packages and their dependencies. Conda also creates virtual environments and allows users to switch between those environments. The instructions below installs Conda and creates a virtual environment in which to install software required by SQuIRE. Following these instructions ensures that SQuIRE has the correct software versions and dependencies and prevents software conflicts.

1. Download Miniconda from https://conda.io/miniconda.html
    - wget -c https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh
    - Documentation will appear as the software downloads
2. Execute the installer and add to PATH in .bashrc
    - bash Miniconda3-latest-Linux-x86_64.sh
    - Press ENTER key to review the Miniconda license
    - Type yes to approve the license terms
    - Pres ENTER key to confirm install locatino (or enter a preferred location)
    - Type yes to add Miniconda2 into your PATH
3. Add PATH to .bash_profile as well
    - tail -n1 ~/.bashrc >> ~/.bash_profile
4. Restart shell
    - exec $SHELL
5. Create new virtual environment
    - conda create --name squire --override-channels -c iuc -c bioconda -c conda-forge -c defaults -c r python=2.7.13 bioconductor-deseq2=1.16.1 r-base=3.4.1 r-pheatmap bioconductor-vsn bioconductor-biocparallel=1.12.0 r-ggrepel star=2.5.3a bedtools=2.25.0 samtools=1.1 stringtie=1.3.3 igvtools=2.3.93 ucsc-genepredtobed ucsc-genepredtogtf ucsc-bedgraphtobigwig r-hexbin git=2.11.1
    - Type y to proceed.
6. Activate the virtual environment
    - source activate squire
    - **Enter this command each time you wish to use the SQuIRE pipeline**

    - The conda installation message may instruct the use of 'conda activate squire'. However, this is a newer and less stable usage than "source activate squire", which we recommend.
7. Install SQuIRE in the virtual environment
    - git clone https://github.com/wyang17/SQuIRE; cd SQuIRE; pip install -e .
    - The -e parameter for "pip install" automatially affects the current SQuIRE installation, so that there is no need to re-install SQuIRE with a new version.
    - To update SQuIRE, go to the SQuIRE folder and enter:
        - git pull

Notes

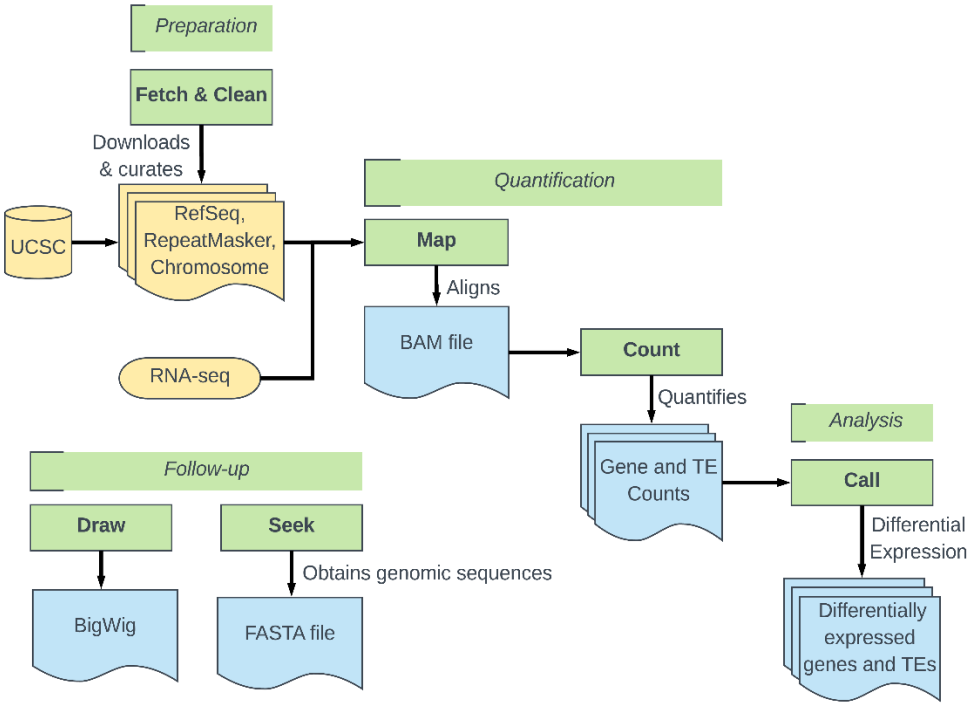
SQuIRE was written and tested with the following specific versions of software:

- STAR 2.5.3a
- bedtools 2.27.0
- samtools 1.1
- stringtie 1.3.3b
- DESeq2 1.16.1
- R 3.4.1
- Python 2.7

If installing these software with conda is unsuccessful, we recommend installing these versions with squire Build to ensure compatibility with SQuIRE.

- squire Build:
    - squire Build -s all

Pipeline Overview



Preparation Stage

1. Fetch: Downloads input files from RefGene and generates STAR index Only needs to be done once initially to acquire genomic input files or if a new build is desired.
2. Clean: Filters Repeatmasker file for Repeats of interest, collapses overlapping repeats, and returns as BED file.

**Optional: Incorporation of non-reference TE sequence* *

Quantification Stage

1. Map: Aligns RNAseq data
2. Count: Quantifies RNAseq reads aligning to TEs

Analysis Stage

1. Call: Compiles and outputs differential expression from multiple alignments

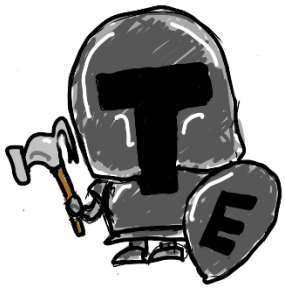
Follow-up Stage

1. Draw: Creates BEDgraphs from RNAseq data
2. Seek: Reports individual transposable element sequences

An example pipeline with sample scripts is described here.

Arguments for each step

squire Build
Use Build only if conda create does not successfully install software.



- Download and install required software (STAR, Bedtools, Samtools, and/or Stringtie)
- Adds software to PATH
- **usage** squire Build -o -s STAR,bedtools,samtools,stringtie -v

Arguments:	
-b, --folder	Destination folder for downloaded UCSC file(s). Optional; default='squire_build'
-s , --software	Install required SQuIRE software and add to PATH - specify 'all' or provide comma-separated list (no spaces) of: STAR,bedtools,samtools,stringtie. Optional; default = False
-v, --verbosity	Want messages and runtime printed to stderr. Optional.

Preparation Stage

squire Fetch

- Downloads required files from repeatmasker
- Only needs to be used the first time SQuIRE is used to transfer required genomic build references to your machine
- Outputs annotation files, chromosome fasta file(s) and STAR index
- **usage:** squire Fetch [-h] -b <build> [-o <folder>] [-f] [-c] [-r] [-g] [-x] [-p <int>] [-k] [-v]

Arguments	
-h, --help	show this help message and exit
-b , --build	UCSC designation for genome build, eg. 'hg38'
-o , --fetch_folder	Destination folder for downloaded UCSC file(s), default folder is 'squire_fetch'
-f, --fasta	Download chromosome fasta files for build chromosomes. Optional
-c, --chrom_info	Download chrom_info.txt file with chromosome lengths. Optional
-r, --rmsk	Download Repeatmasker file. Optional
-g, --gene	Download UCSC gene annotation. Optional
-x, --index	Create STAR index (WARNING: will take a lot of time and memory!), optional
-p , --pthreads	Launch parallel threads. Optional, default = 1
-k, --keep	Keep downloaded compressed files. Optional, default = False
-v, --verbosity	Print messages and runtime records to stderr. Optional; default = False

squire Clean

- Filters genomic coordinates of Repeats of interest from repeatmasker, collapses overlapping TEs, and returns BED file and count of subfamily copies.
- Only needs to be done at the first use of SQuIRE pipeline to clean up the index files
- Outputs .bed file of TE coordinates, strand and divergence
- **usage:** squire Clean [-h] [-r <rmsk.txt or file.out>] [-b <build>] [-o <folder>] [-c <classes>] [-f <subfamilies>] [-s <families>] [-e <file>] [-v]

Arguments	
-h, --help	show this help message and exit
-r , --rmsk	Repeatmasker file, default will search 'squire_fetch' folder for rmsk.txt or .out file. Optional
-b , --build	UCSC designation for genome build, eg. 'hg37'
-o , --clean_folder	Destination folder for output BED file, default folder is 'squire_clean'
-c , --repclass	Comma-separated list of desired repeat classes (AKA superfamilies), eg 'DNA,LTR'. Column 12 in repeatmasker file. Can use UNIX wildcard patterns. Optional
-f , --family	Comma-separated list of desired repeat families, eg 'ERV1,ERVK,ERVL'. Column 13 on repeatmasker file. Can use UNIX wildcard patterns. Optional
-s , --subfamily	Comma-separated list of desired repeat subfamilies, eg 'L1HS,AluYb'. Column 11 in repeatmasker file. Can use UNIX wildcard patterns. Optional
-e , --extra	Filepath of extra tab-delimited file containing non-reference repeat sequences. Columns should be chr, start, stop, strand, subfamily, and sequence. Optional; default = False
-v, --verbosity	Print messages and runtime records to stderr. Optional; default = False

Non-reference File Format

For known TE sequences that are not included in the reference genome, a tab-delimited file can be provided to SQuIRE to incorporate the non-reference TEs into the analysis. This file can be inputted into the Map and Clean steps with the --extra parameter.

The following information should be included in the file:

1. **Chromosome or Plasmid Identification**
- SQuIRE will add an identifier with an underscore "_" and the insertion type to distinguish the annotation from the reference genome.
2. **Insertion Start**
- 0-based numerical start location of the non-reference repeat.
3. **Insertion End**
- 0-based numerical end location. For chromosome insertions, this will only be one base different from Insertion Start.
4. **Strand**
- + or - Orientation of 'sense' strand of TE annotation.
5. **TE classification**
- Provide TE Subfamily, Family and Order, separated by colons ":".
6. **Insertion Type**
- Must be one of: polymorphic insertion, novel insertion, plasmid, or transgene.
7. **Left-Flank Sequence**
- Flanking sequence before the TE insertion.
8. **Right-Flank Sequence**
- Flanking sequence after the TE insertion.
9. **TE Sequence**
- Non-reference TE sequence.

Example File

Chromosome/ Vector	Insertion_start	Insertion_stop	Strand	Subfamily:Family:Order	Insertion_Type: Polymorphism,Novel, Plasmid, Transgene	Left-Flank_Seq	Right-Flank_Seq	TE_Seq
chr15	50070420	50070421	+	AluY:Alu:SINE	Polymorphism	TGATTTTTCCCTAGG GAACCTAACCCTGGC TCACTCTCAGAACAT TTGACTCCACTGGTG GTAGAATAACTCAA GAATGACTAAACAG GGTCTTCCATGACCT GGGCTATAATTCAAG GAGGAGCCTAGGAC CTATCCAGGTCAGTG AGACTCAATGGTAG GACCACTGGT	TCCACTGGGATTGATGTGCT GTGAGG	AGAAAGAGGCTCTCTTGGCCGGGCGCGGTG GCTCACGCCTGTAATCCCAGCACTTTGGGA GGCCGAGACGGGCGGATCACGAGGTCAGG AGATCGAGACCATCTTGGCTAACACGGTGA AACCCCGTTTCTACTAAAAATACAAAAAAT TAGCCGGGCGTGTTGGCGGGCGCCTGTAGT CCCAGCTACTTGGGAGGCTGAGGCAGGAGA ATGGCGTGAACCCGGGAGGCGGAGCTTGCA GTGAGCCGAGATTGCGCCACCGCACTCCAA CCTGGGAGACAGCGAGACTCCGTCTCAA AAAAAAAAAAAAAAAAAAAAAAAAAAAGA AAGAGGCTCTCTT
DA_L1RP	70	6087	+	L1HS:L1:LINE	Plasmid	CGTTTAGTGAACCGT CAGATCTCTAGAAGC TGGGTACCAGCTGCT AGCAAGCTTGCTAGC GGCCGCGGGG	ATCCAGACATGATAAGATAC ATTGATGAGTTTGGACAAAC CACAACTAGAATGCAGTGAA AAAAATGCTTTATTTGTGAA ATTTGTGATGCTATTGCTTTA TTTGTAACCATTATAAGCTGC AATAAACAAGTTAACAACAA CAATTGCATTCATTTTATGTT TCAGGTTCAGGGGGAGGTGT GGGAGGTTTTTTAAAGCAAG TAAAACC	GGAGGAGCCAAGATGGCCGAATAGGAACA GCTCCGGTCTACAGCTCCCAGCGTGAGCGA CGCAGAAGACGGTGATTTCTGCATTTCCAT CTGAGGTACCGGGTTCATCTCACTAGGGAG TGCCAGACAGTGGGCGCAGGCCAGTGTGTG TGCGCACCGTGCGCGAGCCGAAGCAGGGC GAGGCATTGCCTCACCTGGGAAGCGCAAGG GGTCAGGGAGTTCCCTTTCCGAGTCAAAGA AGGGGGTCACGGACGCACCTGGAAAATCG GGTCACTCCCACCCGAATATTGCGCTTTTCA GACCGGCTTAAGAAACGGCGCACCACGAG ACTATATCCCGCACCTGGCTCGGAGGGTCC TACGCCCACGGAATCTCGCTGATTGCTAGC ACAGCAGTCTGAGATCAAACTGCAAGGCGG CAACGAGGCTGGGGGAGGGGCGCCCGCCA TTGCCCAGGCTTGCTTAGGTAAACAAAGCA GCAGGGAAGCTCGAACTGGGTGGAGCCCA CCACAGCTCAAGGAGGCCTGCCTGCCTCTG TAGGCTCCACCTCTGGGGGCAGGGCACAGA CAAACAAAAAGACAGCAGTAACCTCTGCAG ACTTAAGTGTCCCTGTCTGACAGCTTTGAAG AGAGCAGTGGTTCTCCCAGCACGCAGCTGG AGATCTGAGAACGGGCAGACTGCCTCCTCA AGTGGGTCCCTGACCCCTGACCCCCGAGCA GCCTAACTGGGAGGCACCCCCCAGCAGGGG

Quantification Stage

squire Map

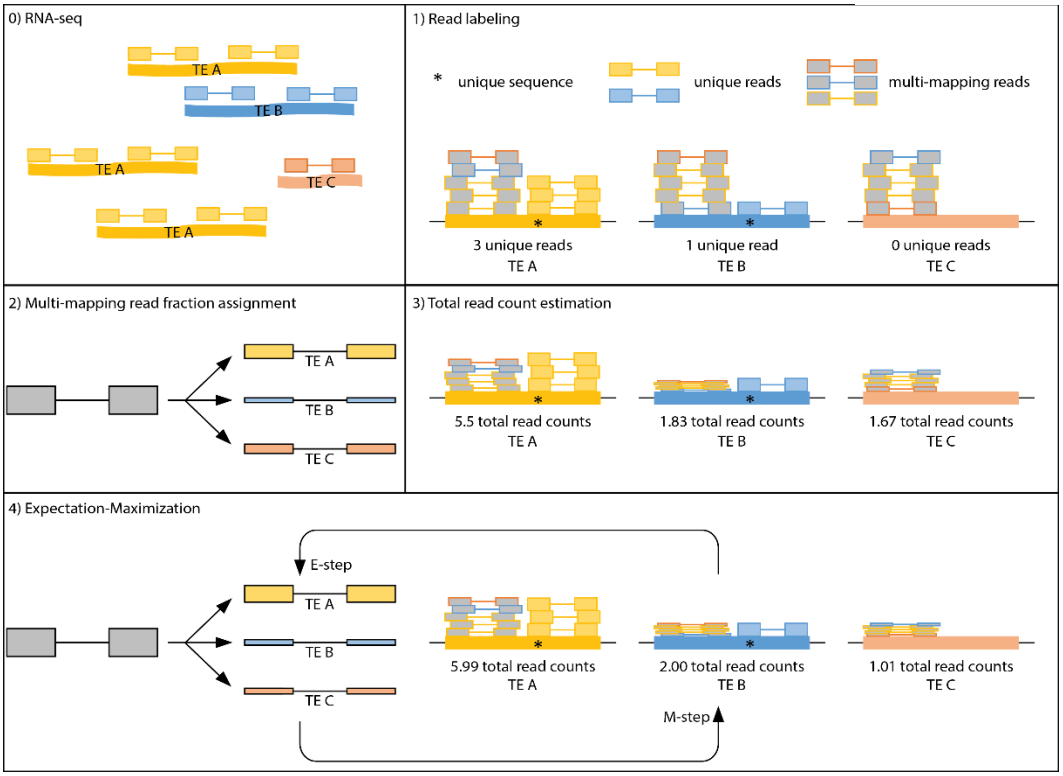
- Aligns RNAseq reads to STAR index allowing for multiple alignments
- Outputs .bam file
- **usage:** squire Map [-h] [-1 <file_1.fastq or file_1.fastq.gz>] [-2 <file_2.fastq or file_2.fastq.gz>] [-o <folder>][-f <folder>] -r <int> [-n <str>] [-3 <int>] [-e <file.txt>] [-b <build>] [-p <int>] [-v]

Arguments	
-h, --help	show this help message and exit
-1 , --read1	RNASeq data fastq file(s); read1 if providing paired end data. If more than one file, separate with commas, no spaces. Can be gzipped.
-2 , --read2	RNASeq data read2 fastq file(s). if more than one file, separate with commas, no spaces. Can be gzipped. Optional if unpaired data.
-o , --map_folder	Destination folder for output files. Optional, default = 'squire_map'
-f ,--fetch_folder	Folder location of outputs from SQuIRE Fetch (optional, default = 'squire_fetch'
-r , --read_length	Read length (if trim3 selected, after trimming; required)
-n , --name	Common basename for RNAseq input. Optional, default = basename of read1
-b ,--build , UCSC designation for genome build, eg. 'hg38' (required if more than 1 build in clean_folder)	
-3 , --trim3	Trim bases from right end of each read before alignment. Optional; default = 0
-e , --extra	Filepath of text file containing non-reference repeat sequence and genome information. Optional, default = False

Arguments	
-g , --gtf	Optional GTF of genome transcripts. For those interested in gene transcription
-p , --pthreads	Launch parallel threads. Optional, default = '1'
-v, --verbosity	Print messages and runtime records to stderr. Optional; default = False

squire Count

- Quantifies RNAseq reads aligning to TEs and genes
- Outputs counts for RefSeq genes and TEs at the locus and subfamily levels



- **usage:** squire Count [-h] [-m <folder>] [-c <folder>] [-o <folder>] [-t <folder>] [-f <folder>] -r <int> [-n <str>] [-b <build>] [-p <int>] [-s <int>] [-e EM] [-v]

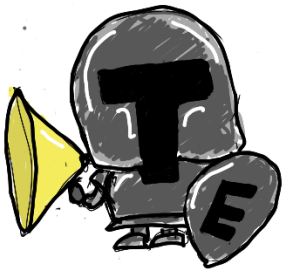
Arguments:	
-h, --help	show this help message and exit
-m , --map_folder	Folder location of outputs from SQuIRE Map (optional,default = 'squire_map')
-c , --clean_folder	Folder location of outputs from SQuIRE Clean (optional, default = 'squire_clean')
-o , --count_folder	Destination folder for output files(optional, default = 'squire_count')
-t , --tempfolder	Folder for tempfiles (optional; default=count_folder')

Arguments:	
-f , --fetch_folder	Folder location of outputs from SQuIRE Fetch (optional, default = 'squire_fetch')
-r , --read_length	Read length (if trim3 selected, after trimming; required).
-n , --name	Common basename for input files (required if more than one bam file in map_folder)
-b , --build	UCSC designation for genome build, eg. 'hg38' (required if more than 1 build in clean_folder)
-p , --pthreads	Launch parallel threads(optional; default='1')
-s , --strandedness	'0' if unstranded eg Standard Illumina, 1 if first- strand eg Illumina Truseq, dUTP, NSR, NNSR, 2 if second-strand, eg Ligation, Standard SOLiD (optional,default=0)
-e , --*EM*	Run estimation-maximization on TE counts given numberof times (optional, specify 0 if no EM desired; default=auto)
-v, --verbosity	Want messages and runtime printed to stderr (optional; default=False)

Analysis Stage

squire Call

- Performs differential expression analysis on TEs and genes
- Outputs DEseq2 output and plots
- **usage** squire Call [-h] -1 <str1,str2> or <*str*> -2 <str1,str2> or <*str*> -A -B [-o] [-s] [-p] [-N] [-f] [-v]



Arguments	
-h, --help	show this help message and exit
-1 <str1,str2> or <*str*>, --group1 <str1,str2> or <*str*>	List of basenames for group1 (Treatment) samples, can also provide string pattern common to all group1 basenames
-2 <str1,str2> or <*str*>, --group2 <str1,str2> or <*str*>	List of basenames for group2 (Control) samples, can also provide string pattern common to all group2 basenames
-A , --condition1	Name of condition for group1
-B , --condition2	Name of condition for group2
-o , --call_folder	Destination folder for output files (optional; default='squire_call')
-s, --subfamily	Compare TE counts by subfamily. Otherwise, compares TEs at locus level (optional; default=False)
-p , --pthreads	Launch parallel threads(optional; default='1')
-N , --projectname	Basename for project
-f , --output_format	Output figures as html or pdf
-v, --verbosity	Want messages and runtime printed to stderr (optional; default=False)

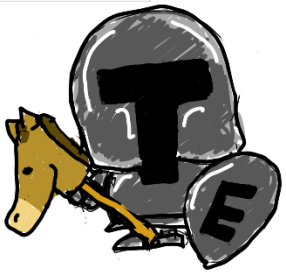
Follow-up Stage

squire Draw

- Creates bedgraphs and bigwigs from RNAseq data
- **usage** squire Draw [-h] [-f] [-m] [-o] [-n] [-s] -b [-l] [-p] [-v]

Arguments	
-h, --help	show this help message and exit
-f , --fetch_folder	Folder location of outputs from SQuIRE Fetch (optional, default = 'squire_fetch')
-m , --map_folder	Folder location of outputs from SQuIRE Map (optional, default = 'squire_map')
-o , --draw_folder	Destination folder for output files (optional; default='squire_draw')
-n , --name	Basename for bam file (required if more than one bam file in map_folder)
-s , --strandedness	'0' if unstranded, 1 if first-strand eg Illumina Truseq, dUTP, NSR, NNSR, 2 if second-strand, eg Ligation, Standard (optional,default=1)
-b , --build	UCSC designation for genome build, eg. 'hg38' (required)
-l, --normlib	Normalize bedgraphs by library size (optional; default=False)
-p , --pthreads	Launch parallel threads(optional; default='1')
-v, --verbosity	Want messages and runtime printed to stderr (optional; default=False)

squire Seek



- Retrieves transposable element sequences from chromosome fasta files
- Outputs sequences in FASTA format
- **usage** squire Seek [-h] -i <file.bed> -o <file.fa> -g <file.fa or folder.chromFa> [-v]

Arguments	
-h, --help	show this help message and exit
-i, --infile	Repeat genomic coordinates, can be TE_ID, bedfile, or gff
-o, --outfile	Repeat sequences output file (FASTA), can use "-" for stdout
-g, --genome	Genome build's fasta chromosomes - .fa file or .chromFa folder
-v, --verbosity	Print messages and runtime records to stderr. Optional; default = False

FAQs

How do I know if my data is stranded or not?

The RNA-seqlopedia by Cresko Lab at University of Oregon outlines strand specific data in section 3.7 Preparation of stranded libraries. You can verify the strand specificity with the researcher who collected the data, or use an outside program like infer-experiment.py in RSeQC or the libtype option in Salmon.

How much memory does each step require?

You can gauge how much vmem to assign to each job based on the number of reads in your datasets.

Can SQuIRE be used on ChIP or small RNA?

SQuIRE has not yet been tested with ChIP or small RNA sequencing data, so its compatibility has not yet been determined.

Example Pipeline

INSTRUCTIONS

1. Copy the sample_scripts folder to your project folder
    - mkdir <project folder>/scripts
    - cp SQuIRE/sample_scripts/* <project folder>/scripts
    - cd <project folder>/scripts
2. Fill out the arguments.sh file
3. Replace "squire@email.com" in the #$ -M squire@email.com line with your email address to get alert of script completion and memory usage
4. Submit jobs to SGE cluster (the -cwd option results in error and output files associated to stay in your current working directory)
    - qsub -cwd fetch.sh arguments.sh
    - qsub -cwd clean.sh arguments.sh
    - qsub -cwd loop_map.sh arguments.sh
    - qsub -cwd loop_count.sh arguments.sh
    - qsub -cwd call.sh arguments.sh
    - qsub -cwd loop_draw.sh arguments.sh
5. If a memory or segmentation fault error occurs, edit the #$ -l mem_free and #$ -l h_vmem lines to increase memory usage for the appropriate script.

Appendix B. SQuIRE Command-line Interface

```
#!/bin/env python

##################### MODULES ###################
import sys
import os
import shutil
import subprocess
from subprocess import *
import argparse #module that passes command-line arguments into script
from pkg_resources import get_distribution
__version__ = get_distribution("SQuIRE").version
script_folder=os.path.dirname(os.path.realpath(__file__))
currentWorkingDirectory = os.getcwd()
sys.path.append(currentWorkingDirectory)
sys.path.append(script_folder)
## import the processes to be called
import Build as s1
import Fetch as s2
import Clean as s3
import Map as s4
import Count as s5
import Call as s6
import Draw as s7
import Seek as s8

#from squire import __version__
#####################################

def main():
    ## create the top level parser
    parser = argparse.ArgumentParser()
    parser._positionals.title = "SQuIRE Steps"
    parser.add_argument('--version', action="version", version=__version__, help="print SQuIRE
version number")
    subparsers = parser.add_subparsers()
    # create subparser for Download Step b, "Build"
    parser1 = subparsers.add_parser("Build", help = "Installs required software")
    parser1._optionals.title = "Arguments"
    parser1.add_argument("-b","--build_folder", help = "Destination folder for downloaded UCSC
file(s) (optional; default='squire_build')", type=str, default="squire_build", metavar = "<folder>")
    parser1.add_argument("-s","--software", help = "Install required SQuIRE software and add to
PATH - specify 'all' or provide comma-separated list (no spaces) of:
STAR,bedtools,samtools,stringtie (optional; default = False)" , type=str, metavar = "<software>",
default=False)
    parser1.add_argument("-v","--verbosity", help = "Want messages and runtime printed to stderr
(optional; default=False)",  action = "store_true", default=False)
    parser1.set_defaults(func=s1.main)

    ## create subparser for Download Step 1, "Fetch"
```

```
        parser2 = subparsers.add_parser("Fetch", help ="Downloads input files from UCSC")
        parser2._optionals.title = "Arguments"
        parser2.add_argument("-b","--build", help = "UCSC designation for genome build, eg. 'hg37' (required)", type=str, required = True, metavar = "<build>")
        parser2.add_argument("-o","--fetch_folder", help = "Destination folder for downloaded UCSC file(s) (optional; default='squire_fetch')", type=str, default="squire_fetch", metavar = "<folder>")
        parser2.add_argument("-f","--fasta", help = "Download chromosome fasta files for build chromosomes (optional; default=False)", action = "store_true", default=False)
        parser2.add_argument("-c","--chrom_info", help = "Download chrom_info.txt file with lengths of each chromosome (optional; default=False)", action = "store_true", default=False)
        parser2.add_argument("-r","--rmsk", help = "Download Repeatmasker file (optional; default=False)", action = "store_true", default=False)
        parser2.add_argument("-g","--gene", help = "Download UCSC gene annotation(optional; default=False)", action = "store_true", default=False)
        parser2.add_argument("-x","--index", help = "Create STAR index, WARNING will take a lot of time and memory (optional; default=False)", action = "store_true", default=False)
        parser2.add_argument("-p","--pthreads", help = "Launch <int> parallel threads(optional; default='1')", type = int, metavar = "<int>", default=1)
        parser2.add_argument("-k","--keep", help = "Keep downloaded compressed files (optional; default=False)", action = "store_true", default=False)
        parser2.add_argument("-v","--verbosity", help = "Want messages and runtime printed to stderr (optional; default=False)",  action = "store_true", default=False)
        parser2.set_defaults(func=s2.main)

        ## create subparser for Step1, "Clean"
        parser3 = subparsers.add_parser("Clean", help = "Filters Repeatmasker file for Repeats of interest, collapses overlapping repeats, and returns as BED file.")
        parser3._optionals.title = "Arguments"
        parser3.add_argument("-r","--rmsk", help = "Repeatmasker file (optional; will search 'squire_fetch' folder for rmsk.txt or .out file by default)", type=str, metavar = "<rmsk.txt or file.out>")
        parser3.add_argument("-b","--build", help = "UCSC designation for genome build, eg. 'hg37' (optional; will be basename of rmsk.txt file by default)", type=str, metavar = "<build>")
        parser3.add_argument("-i","--fetch_folder", help = "Destination folder for downloaded UCSC file(s) (optional; default='squire_fetch')", type=str, default="squire_fetch", metavar = "<folder>")
        parser3.add_argument("-o","--clean_folder", help = "Destination folder for output BED file (optional; default = 'squire_clean')", type=str, default = "squire_clean", metavar = "<folder>")
        parser3.add_argument("-c","--repclass", help = "Comma-separated list of desired repeat class/classes, aka superfamily, eg DNA, LTR. Column 12 in repeatmasker file. Can use UNIX wildcard patterns. (optional; default=False)", type=str, metavar = "<classes>")
        parser3.add_argument("-f","--family", help = "Comma-separated list of desired repeat family/families, eg 'ERV1,ERVK,ERVL. Column 13 in repeatmasker file. Can use UNIX wildcard patterns.  (optional; default=False)", type=str, metavar = "<subfamilies>")
        parser3.add_argument("-s","--subfamily", help = "Comma-separated list of desired repeat subfamilies, eg 'L1HS,AluYb'. Column 11 in repeatmasker file. Can use UNIX wildcard patterns. (optional; default=False)", type=str, metavar = "<families>")
        parser3.add_argument("-e","--extra", help = "Filepath of extra file containing non-reference repeat sequences. Columns should be chr, start, stop, strand, subfamily, and sequence (optional)", type=str, metavar = "<file>", default=False)
        parser3.add_argument("-v","--verbosity", help = "Want messages and runtime printed to stderr (optional; default=False)",  action = "store_true", default = False)
        parser3.set_defaults(func=s3.main)
```

```
        ## create subparser for Step2, 'Map'
        parser4 = subparsers.add_parser('Map', help='Aligns RNAseq reads to STAR index allowing for multiple alignments')
        parser4._optionals.title = "Arguments"
        parser4.add_argument("-1","--read1", help = "RNASeq data fastq file(s); read1 if providing paired end data. If more than one file, separate with commas, no spaces. Can be gzipped.", type = str, metavar = "<file_1.fastq or file_1.fastq.gz>")
        parser4.add_argument("-2","--read2", help = "RNASeq data read2 fastq file(s). if more than one file, separate with commas, no spaces. Can be gzipped.  (optional, can skip or enter 'False' if data is unpaired)", type = str, metavar = "<file_2.fastq or file_2.fastq.gz>")
        parser4.add_argument("-o","--map_folder", help = "Location of SQuIRE Map outputs (optional, default = 'squire_map')", type = str, metavar = "<folder>", default = "squire_map")
        parser4.add_argument("-f","--fetch_folder", help = "Folder location of outputs from SQuIRE Fetch (optional, default = 'squire_fetch'",type = str, metavar = "<folder>",default="squire_fetch")
        parser4.add_argument("-r","--read_length", help = "Read length (if trim3 selected, after trimming; required).", type = int, metavar = "<int>", required=True)
        parser4.add_argument("-n","--name", help = "Common basename for input files (optional; uses basename of read1 as default)", type = str, metavar = "<str>",default=False)
        parser4.add_argument("-3","--trim3", help = "Trim <int> bases from right end of each read before alignment (optional; default=0).", type = int, default = 0, metavar = "<int>")
        parser4.add_argument("-e","--extra", help = "Filepath of text file containing non-reference repeat sequence and genome information", type=str, metavar = "<file.txt>")
        parser4.add_argument("-b","--build", help = "UCSC designation for genome build, eg. 'hg38' (required if more than 1 build in clean_folder)", type=str, metavar = "<build>",default=False)
        # parser.add_argument("-m","--mask", help = "Separate reads from bamfile that map to plasmid or transgene into another file (optional; default=False)", action = "store_true", default = False)
        parser4.add_argument("-p","--pthreads", help = "Launch <int> parallel threads(optional; default='1')", type = int, metavar = "<int>", default=1)
        parser4.add_argument("-v","--verbosity", help = "Want messages and runtime printed to stderr (optional; default=False)", action = "store_true", default = False)

        parser4.set_defaults(func=s4.main)

        ## create subparser for Step3, 'Count'
        parser5 = subparsers.add_parser('Count', help = "Quantifies RNAseq reads aligning to TEs and genes")
        parser5._optionals.title = "Arguments"
        parser5.add_argument("-m","--map_folder", help = "Folder location of outputs from SQuIRE Map (optional, default = 'squire_map')", type = str, metavar = "<folder>",default="squire_map")
        parser5.add_argument("-c","--clean_folder", help = "Folder location of outputs from SQuIRE Clean (optional, default = 'squire_clean')", type = str, metavar = "<folder>",default = "squire_clean")
        parser5.add_argument("-o","--count_folder", help = "Destination folder for output files(optional, default = 'squire_count')", type = str, metavar = "<folder>", default="squire_count")
        parser5.add_argument("-t","--tempfolder", help = "Folder for tempfiles (optional; default=count_folder')", type = str, metavar = "<folder>", default=False)
        parser5.add_argument("-f","--fetch_folder", help = "Folder location of outputs from SQuIRE Fetch (optional, default = 'squire_fetch)'",type = str, metavar = "<folder>",default="squire_fetch")
        parser5.add_argument("-r","--read_length", help = "Read length (if trim3 selected, after trimming; required).", type = int, metavar = "<int>", required=True)
```

```
        parser5.add_argument("-n","--name", help = "Common basename for input files (required if more than one bam file in map_folder)", type = str, metavar = "<str>",default=False)
        parser5.add_argument("-b","--build", help = "UCSC designation for genome build, eg. 'hg38' (required if more than 1 build in clean_folder)", type=str, metavar = "<build>",default=False)
        parser5.add_argument("-p","--pthreads", help = "Launch <int> parallel threads(optional; default='1')", type = int, metavar = "<int>", default=1)
        parser5.add_argument("-s","--strandedness", help = " '0' if unstranded eg Standard Illumina, 1 if first-strand eg Illumina Truseq, dUTP, NSR, NNSR, 2 if second-strand, eg Ligation, Standard SOLiD (optional,default=0)", type = int, metavar = "<int>", default = 0)
        parser5.add_argument("-e","--EM", help = "Run estimation-maximization on TE counts given number of times (optional, specify 0 if no EM desired; default=auto)", type=str, default = "auto")
        parser5.add_argument("-v","--verbosity", help = "Want messages and runtime printed to stderr (optional; default=False)", action = "store_true", default = False)

    ## set which program to be associated with this parser
        parser5.set_defaults(func=s5.main)

        parser6 = subparsers.add_parser("Call",help = """Performs differential expression analysis on TEs and genes""")
        parser6._optionals.title = "Arguments"
        parser6.add_argument("-1","--group1", help = "List of basenames for group1 (Treatment) samples, can also provide string pattern common to all group1 basenames",required = True, type = str, metavar = "<str1,str2> or <*str*>")
        parser6.add_argument("-2","--group2", help = "List of basenames for group2 (Control) samples, can also provide string pattern common to all group2 basenames",required = True, type = str, metavar = "<str1,str2> or <*str*>")
        parser6.add_argument("-A","--condition1", help = "Name of condition for group1",required = True, type = str, metavar = "<str>")
        parser6.add_argument("-B","--condition2", help = "Name of condition for group2",required = True, type = str, metavar = "<str>")
        parser6.add_argument("-i","--count_folder", help = "Folder location of outputs from SQuIRE Count (optional, default = 'squire_count')", type = str, metavar = "<folder>",default="squire_count")
        parser6.add_argument("-o","--call_folder", help = "Destination folder for output files (optional; default='squire_call')", type = str, metavar = "<folder>", default="squire_call")
        parser6.add_argument("-s","--subfamily", help = "Compare TE counts by subfamily. Otherwise, compares TEs at locus level (optional; default=False)", action = "store_true", default = False)
        parser6.add_argument("-p","--pthreads", help = "Launch <int> parallel threads(optional; default='1')", type = int, metavar = "<int>", default=1)
        parser6.add_argument("-N","--projectname", help = "Basename for project, default='SQuIRE'",type = str, metavar = "<str>",default="SQuIRE")
        parser6.add_argument("-f","--output_format", help = "Output figures as html or pdf", type = str, metavar = "<str>",default="html")
        parser6.add_argument("-t","--table_only", help = "Output count table only, don't want to perform differential expression with DESeq2", action = "store_true", default = False)
        parser6.add_argument("-v","--verbosity", help = "Want messages and runtime printed to stderr (optional; default=False)", action = "store_true", default = False)

        parser6.set_defaults(func=s6.main)
```

```
        parser7 = subparsers.add_parser('Draw', help  = """Makes bedgraphs and bedwigs from
RNAseq data""")
        parser7._optionals.title = "Arguments"
        parser7.add_argument("-f","--fetch_folder", help = "Folder location of outputs from SQuIRE
Fetch (optional, default = 'squire_fetch')",type = str, metavar = "<folder>",default="squire_fetch")
        parser7.add_argument("-m","--map_folder", help = "Folder location of outputs from SQuIRE
Map (optional, default = 'squire_map')", type = str, metavar = "<folder>", default="squire_map")
        parser7.add_argument("-o","--draw_folder", help = "Destination folder for output files
(optional; default='squire_draw')", type = str, metavar = "<folder>", default="squire_draw")
        parser7.add_argument("-n","--name", help = "Basename for bam file (required if more than
one bam file in map_folder)", type = str, metavar = "<str>",default=False)
        parser7.add_argument("-s","--strandedness", help = " '0' if unstranded, 1 if first-strand eg
Illumina Truseq, dUTP, NSR, NNSR, 2 if second-strand, eg Ligation, Standard
(optional,default=1)", type = int, metavar = "<int>", default = False)
        parser7.add_argument("-b","--build", help = "UCSC designation for genome build, eg. 'hg38'
(required)", type=str, metavar = "<build>",default=False,required=True)
        parser7.add_argument("-l","--normlib", help = "Normalize bedgraphs by library size (optional;
default=False)", action = "store_true", default = False)
        parser7.add_argument("-p","--pthreads", help = "Launch <int> parallel threads(optional;
default='1')", type = int, metavar = "<int>", default=1)
        parser7.add_argument("-v","--verbosity", help = "Want messages and runtime printed to stderr
(optional; default=False)", action = "store_true", default = False)
        parser7.set_defaults(func=s7.main)

        parser8 = subparsers.add_parser("Seek", help = """Retrieves sequences from chromosome
fasta files designated by BED file coordinates""")
        parser8._optionals.title = "Arguments"
        parser8.add_argument("-i","--infile", help = """Repeat genomic coordinates, can be TE_ID,
bedfile, or gff (required)""", type=argparse.FileType('r'), metavar = "<file.bed>", required=True)
        parser8.add_argument("-o","--outfile", help = """Repeat sequences output file (FASTA), can
use "-" for stdout (required)""", type = argparse.FileType('w'), metavar = "<file.fa>", required=True)
        parser8.add_argument("-g","--genome", help = "Genome build's fasta chromosomes - .fa file or
.chromFa folder (required)", type = str, metavar="<file.fa or folder.chromFa>", required=True)
        parser8.add_argument("-v","--verbosity", help = "Want messages and runtime printed to stderr
(optional; default=False)",  action = "store_true", default = False)

        parser8.set_defaults(func=s8.main)

        ## parse the args and call the specific program
        subargs,extra_args = parser.parse_known_args()
        subargs.func(args = subargs)

        # print help usage if no arguments are supplied
        if len(sys.argv)==1 and not ext_args:
            parser.print_help()
            sys.exit(1)

    if __name__=="__main__":
        main()
```

Appendix C. SQuIRE Fetch

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
##################### MODULES ###################
from __future__ import print_function
import sys
import os
import errno
import argparse #module that passes command-line arguments into script
import subprocess
import glob
import urllib
import urllib2
import tarfile
import gzip
from datetime import datetime
import subprocess as sp
import zipfile
from urllib2 import urlopen
import re
import shutil
import tempfile
import pkg_resources
import warnings
#########################################################
def make_dir(path):
    try:
        original_umask = os.umask(0)
        os.makedirs(path, 0770)
    except OSError as exception:
        if exception.errno != errno.EEXIST:
            raise
    finally:
        os.umask(original_umask)

def decompress(compressed, decompressed):     #Function for decompressing gzip files
    inF = gzip.open(compressed, 'rb')
    outF = file(decompressed, 'wb')
    for line in inF:
        outF.write(line)
    outF.close()

def unzip(compressed, decompressed): #unzip .zip files
    zip_ref = zipfile.ZipFile(compressed, 'r')
    zip_ref.extractall(decompressed)
    zip_ref.close()

def failed_dl(filepath):          # If the path created by previous steps is empty, break
    with open(filepath) as downloadedfile:
        for i,line in enumerate(downloadedfile):
```

```
            if "not found" in line.lower():
                return True
                break
            elif i>10: #if past line 10
                return False
                break

def gtf_to_bed(gtf,bed):
    #convert gtf to genepred
    genepred=gtf.replace(".gtf",".genepred")
    gtftogenepredcommand_list = ["gtfToGenePred",gtf,genepred]
    gtftogenepredcommand=" ".join(gtftogenepredcommand_list)
    sp.check_call(["/bin/sh", "-c", gtftogenepredcommand])
    #convert genepred to bed
    genepredtobedcommand_list = ["genePredToBed ",genepred,bed]
    genepredtobedcommand=" ".join(genepredtobedcommand_list)
    sp.check_call(["/bin/sh", "-c", genepredtobedcommand])

def genepred_to_bed(genepred,bed,outfolder):
    refGene_temp=make_tempfile("refGenebed",outfolder)
    #convert genepred to bed
    genepredtobedcommand_list = ["genePredToBed ",genepred,refGene_temp]
    genepredtobedcommand=" ".join(genepredtobedcommand_list)
    sp.check_call(["/bin/sh", "-c", genepredtobedcommand])

    sort_commandlist = ["sort","-k1,1", "-k2,2n",genepred,refGene_temp, ">", bed]
    sort_command = " ".join(sort_commandlist)
    sp.check_call(["/bin/sh", "-c", sort_command])

    os.unlink(refGene_temp)

def genepred_to_gtf(genepred,gtf,outfolder):
    refGene_temp=make_tempfile("refGene",outfolder)
    refGene_temp2=make_tempfile("refGene2",outfolder)
    refGene_temp3=make_tempfile("refGene3",outfolder)

    genePredToGtf_commandlist = ["genePredToGtf","file",genepred,refGene_temp]
    genePredToGtf_command = " ".join(genePredToGtf_commandlist)
    sp.check_call(["/bin/sh", "-c", genePredToGtf_command])

    replace_command_list = ["awk","-v", "OFS='\\t'", """'{ gsub("stdin","hg38_refGene",$2); print $0
}'""", refGene_temp, ">", refGene_temp2]
    replace_command = " ".join(replace_command_list)
    sp.check_call(["/bin/sh","-c",replace_command])

    sort_commandlist = ["sort","-k1,1", "-k4,4n", refGene_temp2, ">", refGene_temp3]
    sort_command = " ".join(sort_commandlist)
    sp.check_call(["/bin/sh", "-c", sort_command])

    fix_gtf(refGene_temp3, gtf)
```

```
    os.remove(refGene_temp)
    os.remove(refGene_temp2)
    os.remove(refGene_temp3)


def make_tempfile(step, outfolder):
    tmpfile = tempfile.NamedTemporaryFile(delete=False, dir = outfolder, prefix= step +  ".tmp")
    tmpname = tmpfile.name
    tmpfile.close()
    return tmpname

def find_files(folder,pattern, wildpos):
    if wildpos == 1:
        file_list=glob.glob(folder + "/" + "*" + pattern)
    elif wildpos ==2:
        file_list=glob.glob(folder + "/" + pattern + "*")
    if len(file_list) == 0:
        raise Exception("No files found in folder; please give specific " + pattern + " file")
    else:
        return file_list

def fix_gtf(infile,outfile):
    outgtf=open(outfile,'w')
    with open(infile,'r') as gtf:
        for line in gtf:
            line = line.rstrip()
            line=line.split()
            attributes=line[8:]
            attribute_col = " ".join(attributes)
            gtf_cols = "\t".join(line[:8])
            outgtf.writelines(gtf_cols + "\t" + attribute_col + "\n")

    outgtf.close()

def sort_coord(infile, outfile,chrcol,startcol):
    chrfieldsort = "-k" + str(chrcol) + "," + str(chrcol)
    startfieldsort = "-k" + str(startcol) + "," + str(startcol) + "n"
    sort_command_list = ["sort",chrfieldsort,startfieldsort, infile, ">", outfile]
    sort_command = " ".join(sort_command_list)
    sp.check_call(["/bin/sh", "-c", sort_command])

def get_basename(filepath):
        filename = os.path.basename(filepath)
        filebase = os.path.splitext(filename)[0]
        return filebase


def get_script_path():
    return os.path.dirname(os.path.realpath(sys.argv[0]))

def main(**kwargs):
```

```
    ######## ARGUMENTS ############
    #check if already args is provided, i.e. main() is called from the top level script
    args = kwargs.get('args', None)  # if no arguments, the below parser statements will be printed
    if args is None: ## i.e. standalone script called from command line in normal way
        parser = argparse.ArgumentParser(description = "Downloads input files from UCSC")
        parser._optionals.title = "Arguments"
        parser.add_argument("-b","--build", help = "UCSC designation for genome build, eg. 'hg38'
(required)", type=str, required = True, metavar = "<build>")
        parser.add_argument("-o","--fetch_folder", help = "Destination folder for downloaded UCSC
file(s) (optional; default='squire_fetch')", type=str, default="squire_fetch", metavar = "<folder>")
        parser.add_argument("-f","--fasta", help = "Download chromosome fasta files for build
chromosomes (optional; default=False)", action = "store_true", default=False)
        parser.add_argument("-c","--chrom_info", help = "Download chrom_info.txt file with lengths of
each chromosome (optional; default=False)", action = "store_true", default=False)
        parser.add_argument("-r","--rmsk", help = "Download Repeatmasker file (optional;
default=False)", action = "store_true", default=False)
        parser.add_argument("-g","--gene", help = "Download UCSC gene annotation(optional;
default=False)", action = "store_true", default=False)
        parser.add_argument("-x","--index", help = "Create STAR index, WARNING will take a lot of
time and memory (optional; default=False)", action = "store_true", default=False)
        parser.add_argument("-p","--pthreads", help = "Launch <int> parallel threads(optional;
default='1')", type = int, metavar = "<int>", default=1)
        parser.add_argument("-k","--keep", help = "Keep downloaded compressed files (optional;
default=False)", action = "store_true", default=False)
        parser.add_argument("-v","--verbosity", help = "Want messages and runtime printed to stderr
(optional; default=False)",  action = "store_true", default=False)
        args,extra_args = parser.parse_known_args()

    ###### I/O #############
    build=args.build
    outfolder=args.fetch_folder
    fasta = args.fasta
    chrom_info=args.chrom_info
    rmsk = args.rmsk
    keep = args.keep
    gene = args.gene
    index=args.index
    pthreads=args.pthreads
    verbosity = args.verbosity

    ######### START TIMING SCRIPT #############
    if verbosity:
        startTime = datetime.now()
        print("start time is:" + str(startTime) + '\n', file = sys.stderr)# Prints start time
        print(os.path.basename(__file__) + '\n', file = sys.stderr) #prints script name to std err
        print("Script Arguments" + '\n' + "==================", file = sys.stderr)  #
        args_dict = vars(args)
        for option,arg in args_dict.iteritems():
            print(str(option) + "=" + str(arg), file = sys.stderr) #prints all arguments to std err
        print("\n", file = sys.stderr)
```

```
    ######## CHECK IF FOLDER DOESN'T EXIST, OTHERWISE CREATE #########

    make_dir(outfolder)

    ####### DOWNLOAD CHROMOSOME FASTA FILES #########
    if fasta:
        if verbosity:
            print("Downloading Compressed Chromosome files..." + "\n", file = sys.stderr)

        chrom_loc1 = "http://hgdownload.cse.ucsc.edu/goldenPath"  + "/" + build + "/" + "bigZips" + "/"
+ "chromFa.tar.gz"  # Different file types depending on size/format of chromosome data
        chrom_loc2 = "http://hgdownload.cse.ucsc.edu/goldenPath"  + "/" + build + "/" + "bigZips" + "/"
+ build + ".chromFa.tar.gz"
        chrom_loc3 = "http://hgdownload.cse.ucsc.edu/goldenPath"  + "/" + build + "/" + "bigZips" + "/"
+ build + ".fa.gz"
        chrom_loc4 = "http://hgdownload.cse.ucsc.edu/goldenPath"  + "/" + build + "/" + "bigZips" + "/"
+ "chromFa.zip"
        chrom_basename = outfolder + "/" + build
        chrom_outfolder= chrom_basename + ".chromFa"
        #Download chromosome fasta files

        chrom_name_compressed = chrom_basename + "chromFa.tar.gz"
        urllib.urlretrieve(chrom_loc1, filename=chrom_name_compressed)
        df_fail1=failed_dl(chrom_name_compressed)
        if df_fail1:
            os.unlink(chrom_name_compressed)
            chrom_name_compressed = chrom_basename + "chromFa.tar.gz"
            urllib.urlretrieve(chrom_loc2, filename=chrom_name_compressed)
            df_fail2=failed_dl(chrom_name_compressed)
            if df_fail2:
                os.unlink(chrom_name_compressed)
                chrom_name_compressed = chrom_basename + ".fa.gz"
                urllib.urlretrieve(chrom_loc3, filename=chrom_name_compressed)
                df_fail3=failed_dl(chrom_name_compressed)
                if df_fail3:
                    os.unlink(chrom_name_compressed)
                    chrom_name_compressed = outfolder + "/" + "chromFa.zip"
                    urllib.urlretrieve(chrom_loc4, filename=chrom_name_compressed)
                    df_fail4=failed_dl(chrom_name_compressed)
                    if df_fail4:
                        os.unlink(chrom_name_compressed)
                        raise Exception("Was not able to download chromosome file from UCSC" + "\n", file
= sys.stderr)


        if verbosity:
            print("Finished Downloading Compressed Chromosome folder, Decompressing..." + "\n", file
= sys.stderr)
```

```
    #Unzip
    if "tar.gz" in chrom_name_compressed:
       chrom_name = chrom_outfolder
       with tarfile.TarFile.open(chrom_name_compressed, 'r') as tarredgzippedFile:
          tarredgzippedFile.extractall(path=chrom_name)
    elif "fa.gz" in chrom_name_compressed:
       chrom_name = chrom_outfolder + "/" + build + ".fa"
       decompress(compressed = chrom_name_compressed, decompressed = chrom_name)
    elif "chromFa.zip" in chrom_name_compressed:
       chrom_name = chrom_outfolder
       unzip(chrom_name_compressed,chrom_name)

    if verbosity:
       print("Finished Decompressing Chromosome folder" + "\n", file = sys.stderr)

    #Removes compressed file
    if keep == False:
       if verbosity:
         print("Deleting Compressed Chromosome folder", file=sys.stderr)
       os.remove(chrom_name_compressed)
    #filter for fasta files and filter out unwanted chromosomes
    if os.path.isdir(chrom_name): # if genome is folder and not file
       fasta_folder = chrom_name + "/" + "chroms"
       if not os.path.isdir(fasta_folder): #if chromFa folder does not have "chroms" subdirectory
          fasta_folder = chrom_name #then fasta files are in chromFa folder

       unwanted_folder = chrom_name + "/" + "unwanted"  # create unwanted folder
       make_dir(unwanted_folder)

       file_list=os.listdir(fasta_folder) #list all files in unwanted folder (previously fasta folder)

       unwantedChr = ["hap", "M", "alt"]

       for i in file_list:    # Cleans up unwanted characters from the files before
          i=i.rstrip()
          i_file = fasta_folder + "/" + i
          wanted_file = chrom_outfolder + "/" + i
          unwanted_file = unwanted_folder + "/" + i
          basename = os.path.splitext(i)[0]
          extension = os.path.splitext(i)[1]

          #Filter out folders, non-fasta files, unwanted chromosome fasta files
          if any(x in basename for x in unwantedChr):
             os.rename(i_file,unwanted_file) # move unwanted chromosome files to
chrom.Fa/unwanted folder
             continue
          if os.path.isdir(i):
             continue
          if i_file != wanted_file:
             os.rename(i_file,wanted_file) # move wanted chromosome files to chromFa folder
```

```
        if "chroms" in fasta_folder:
            os.rmdir(fasta_folder)

        if verbosity:
            print("Chromosome fasta files are in" + chrom_outfolder + "\n", file = sys.stderr)

    ####### DOWNLOAD CHROM_INFO FILE ##########
    if chrom_info:
        if verbosity:
            print("Downloading Chrom_info file..." + "\n", file = sys.stderr)

        chrom_info_loc = "http://hgdownload.cse.ucsc.edu/goldenPath"   + "/" + build  + "/" +
"database" + "/"+ "chromInfo.txt.gz"
        chrom_info_name = outfolder + "/" + build + "_chromInfo.txt"
        chrom_info_name_compressed = chrom_info_name + ".gz"
        #Downloads Chromosome info file
        urllib.urlretrieve(chrom_info_loc, filename=chrom_info_name_compressed)

        if verbosity:
            print("Finished Downloading Chrom_info file, Decompressing..." + "\n", file = sys.stderr)

        #Decompresses chromosome info file
        decompress(compressed = chrom_info_name_compressed, decompressed = chrom_info_name)

        if verbosity:
            print("Finished Decompressing Chrom_info file: "  + "\t" +  chrom_info_name + "\n", file =
sys.stderr)
        #Deletes compressed chromosome info file
        if keep == False:
            if verbosity:
              print("Deleting Compressed Chrom_info file" + "\n", file=sys.stderr)
            os.remove(chrom_info_name_compressed)

    ###### DOWNLOAD REPEATMASKER FILE ##################
    if rmsk:
        if verbosity:
            print("Downloading Repeatmasker file..." + "\n", file = sys.stderr)
        rmsk_file=outfolder + "/" + build + "_rmsk.txt"
        rmsk_list=set()
        rmsk_loc="http://hgdownload.cse.ucsc.edu/goldenPath"   + "/" + build  + "/" + "database" + "/"

        urlpath = urlopen(rmsk_loc)
        string = urlpath.read().decode('utf-8')

        pattern = re.compile('\\brmsk.txt.gz\\b')
        filelist = pattern.findall(string)
        for filename in filelist:
            rmsk_list.add(filename)

        pattern = re.compile('chr[0-9][0-9]*_rmsk.txt.gz')
        filelist = pattern.findall(string)
```

```
        for filename in filelist:
            rmsk_list.add(filename)

        pattern = re.compile('chr[A-Z]_rmsk.txt.gz')
        filelist = pattern.findall(string)
        for filename in filelist:
            rmsk_list.add(filename)

        if len(rmsk_list) > 1:
            if verbosity:
                print("Multiple Repeatmasker files found, Downloading, Decompressing and combining
into a single file..." + "\n", file = sys.stderr)
            with open(rmsk_file,'wb') as outfile:
                for filename in rmsk_list:
                    remotefile=urllib.urlretrieve(rmsk_loc + filename, filename=outfolder +"/" + filename)
                    if verbosity:
                        print("Downloading Compressed Repeatmasker file" + " " + filename + "\n",
file=sys.stderr)
                    newfilename=filename.replace(".gz","")
                    decompress(compressed=outfolder + "/" + filename, decompressed=outfolder + "/" +
newfilename)
                    with open(outfolder + "/" + newfilename, 'rb') as inrmsk:
                        shutil.copyfileobj(inrmsk, outfile)
                        if verbosity:
                            print("Adding to Repeatmasker file" + " " + rmsk_file + "\n", file=sys.stderr)
                     #Deletes decompressed repeatmasker file
                    if keep == False:
                        if verbosity:
                            print("Deleting Compressed Repeatmasker file" + " " + filename + "\n",
file=sys.stderr)
                            os.remove(outfolder +"/" + filename)
                        if verbosity:
                            print("Deleting Decompressed Repeatmasker file" + " " + newfilename + "\n",
file=sys.stderr)
                            os.remove(outfolder + "/" + newfilename)

        elif len(rmsk_list) == 1:
            rmsk_list=list(rmsk_list)
            filename=rmsk_list[0]
            remotefile=urllib.urlretrieve(rmsk_loc + filename, filename=outfolder +"/" + filename)
            if verbosity:
                print("Finished Downloading Repeatmasker file, Decompressing..." + "\n", file = sys.stderr)
            decompress(compressed=outfolder + "/" + filename, decompressed=rmsk_file)
            if keep == False:
                if verbosity:
                    print("Deleting Compressed Repeatmasker file" + "\n", file=sys.stderr)
                    os.remove(outfolder +"/" + filename)
        elif not rmsk_list:
            raise Exception("Was not able to download rmsk file from UCSC" + "\n", file = sys.stderr)

        if verbosity:
```

```
        print("Finished with Repeatmasker download step" + "\n", file = sys.stderr)


    ####### DOWNLOAD GENE ANNOTATIONS ##########
    if gene:
        if verbosity:
            print("Downloading RefGene file..." + "\n", file = sys.stderr)

        refGene_loc = "http://hgdownload.cse.ucsc.edu/goldenPath"   + "/" + build  + "/" + "database" +
"/"+ "refGene.txt.gz"
        refGene_name = outfolder + "/" + build + "_refGene.txt"
        refGene_name_compressed = refGene_name + ".gz"
        #Downloads Chromosome info file
        urllib.urlretrieve(refGene_loc, filename=refGene_name_compressed)

        if verbosity:
            print("Finished Downloading refGene file, Decompressing..." + "\n", file = sys.stderr)

        #Decompresses chromosome info file
        decompress(compressed = refGene_name_compressed, decompressed = refGene_name)

        if verbosity:
            print("Finished Decompressing refGene file: "  + "\t" +  refGene_name + "\n", file =
sys.stderr)
        #Deletes compressed chromosome info file
        if keep == False:
            if verbosity:
                print("Deleting Compressed refGene file" + "\n", file=sys.stderr)
            os.remove(refGene_name_compressed)

        #remove first column
        refGene_genepred=outfolder + "/" + build + "_refGene.genepred"
        removecolumn_commandlist = ["cut","-f2-",refGene_name,">",refGene_genepred]
        removecolumn_command = " ".join(removecolumn_commandlist)
        sp.check_call(["/bin/sh", "-c", removecolumn_command])
        os.unlink(refGene_name)

        if verbosity:
            print("Converting RefGene file to GTF ..." + "\n", file = sys.stderr)
        refGene_gtf=outfolder + "/" + build + "_refGene.gtf"
        genepred_to_gtf(refGene_genepred,refGene_gtf,outfolder)

        if verbosity:
            print("Finished converting RefGene file to GTF ..." + "\n", file = sys.stderr)

        if verbosity:
            print("Converting RefGene file to Bed ..." + "\n", file = sys.stderr)
        refGene_Bed=outfolder + "/" + build + "_refGene.bed"
        genepred_to_bed(refGene_genepred,refGene_Bed,outfolder)

        if verbosity:
```

```
        print("Finished converting RefGene file to Bed ..." + "\n", file = sys.stderr)

    ####### CREATE STAR INDEX ##########
    if index:
        chrom_folder = outfolder + "/" + build + ".chromFa"
        if not os.path.isdir(chrom_folder):
            raise Exception(str(chrom_folder) +  "not found" + "\n", file = sys.stderr)
        fasta_list=find_files(chrom_folder,".fa",1)
        genome_filepath = " ".join(fasta_list)
        index_name = outfolder + "/" + build + "_STAR"
        make_dir(index_name)
        STAR_build_commandlist = ["STAR","""--runThreadN""", str(pthreads), """--runMode
genomeGenerate""","""--genomeFastaFiles""",genome_filepath,"""--genomeDir""",index_name]
        STAR_build_command = " ".join(STAR_build_commandlist)
        if verbosity:
            print("Building STAR index" + "\n", file = sys.stderr)
            print(STAR_build_command,file=sys.stderr)
        sp.check_call(["/bin/sh", "-c", STAR_build_command])
    ####### STOP TIMING SCRIPT ########################
    if verbosity:
        endTime = datetime.now()
        print('end time is: '+ str(endTime) + "\n", file = sys.stderr)   # print end time
        print('it took: ' + str(endTime-startTime) + "\n", file = sys.stderr)   # print total time


############################################################
if __name__ == "__main__":
    main()
```

Appendix D. SQuIRE Map

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
############MODULES##########################
from __future__ import print_function,division
import sys
import os
import errno
import argparse #module that passes command-line arguments into script
from datetime import datetime
import operator #for doing operations on tuple
from operator import itemgetter
import subprocess as sp
from subprocess import Popen, PIPE,STDOUT
import io
import tempfile
#for creating interval from start
from collections import defaultdict #for dictionary
import glob
import re
from six import itervalues
import textwrap
import shutil

####### FUNCTIONS ##############################
def isempty(filepath):
    if os.path.getsize(filepath) == 0:
        raise Exception(filepath + " is empty")

def make_dir(path):
    try:
        original_umask = os.umask(0)
        os.makedirs(path, 0770)
    except OSError as exception:
        if exception.errno != errno.EEXIST:
            raise
    finally:
        os.umask(original_umask)

def get_basename(filepath):
        filename = os.path.basename(filepath)
        filebase = os.path.splitext(filename)[0]
        return filebase

def make_tempfile(basename,step,outfolder):
    tmpfile = tempfile.NamedTemporaryFile(delete=False, dir = outfolder, prefix= basename + "_" +
step +  ".tmp")
    tmpname = tmpfile.name
```

```
    tmpfile.close()
    return tmpname

def rev_comp(sequence):
    rev_seq = sequence[::-1]
    new_seq=""
    for base in rev_seq:
        if base == "A":
            newbase = "T"
        elif base == "T":
            newbase = "A"
        elif base == "C":
            newbase = "G"
        elif base == "G":
            newbase = "C"
        new_seq += newbase
    return new_seq
def rename_file(oldname,newname):
    shutil.move(oldname, newname)


def combine_files(file1,file2,outfile,debug):
    catcommand_list = ["cat", file1, file2, ">", outfile] #combines multi_aligned reads
    catcommand = " ".join(catcommand_list)
    sp.check_call(["/bin/sh","-c",catcommand])

    if not debug:
        os.unlink(file1)
        os.unlink(file2)

def find_file(folder,pattern,base, wildpos, needed):
    foundfile=False
    if wildpos == 1:
        file_list=glob.glob(folder + "/" + "*" + pattern)
    elif wildpos ==2:
        file_list=glob.glob(folder + "/" + pattern + "*")
    if len(file_list)>1: #if more than one file in folder
        if not base:
            raise Exception("More than 1 " + pattern + " file")
        for i in file_list:
            if base in i:
                foundfile = i
    elif len(file_list) == 0:
        foundfile = False
    else:
        foundfile = file_list[0]
    if not foundfile:
        if needed:
            raise Exception("No " + pattern + " file")
        else:
            foundfile = False
```

```
    return foundfile


def align_paired(fastq1,fastq2,pthreads,trim3,index,outfile,gtf,gzip,prefix,read_length,extra_fa):
        ##### ALIGN FASTQ FILE(S) TO GENOME OR REPCHR ########
        #-p16: allows hyperhreading over 16 cores
        #-t: outputs time of alignment
        #--tryhard Puts in maximal effort in finding valid alignments for paired end reads
        #-a:  reports all valid alignments for reads
        #-3 trim3: trims user-specified bases from 3' end of FASTQ sequences (useful for if sequencing
read > subsequence length)
        gtf_option = []
        gzip_option = []
        extra_option = []
        if gtf:
            gtf_option = ["--sjdbGTFfile", gtf, "--sjdbOverhang",str(read_length-1), "--twopassMode",
"Basic"]
        if gzip:
            gzip_option = ["""--readFilesCommand""", "zcat"]
        if extra_fa:
            extra_option=["""--genomeFastaFiles""",extra_fa]


        add_options = gtf_option + gzip_option + extra_option

        multi_align = ["""--outFilterMultimapNmax""", "100", """--winAnchorMultimapNmax""",
"100", "--alignEndsType","EndToEnd" ,"--alignEndsProtrude","100 DiscordantPair"]
        trim = ["""--clip3pNbases""", str(trim3)]
        single_reads = ["""--outFilterScoreMinOverLread""", "0.4", """--
outFilterMatchNminOverLread""", """0.4"""]
        #single_reads=[]
        discordant = ["--chimSegmentMin", str(read_length)]
        #discordant = []
        inputs = ["""--genomeDir""", index,"""--readFilesIn""",fastq1, fastq2]
        outputs = [ """--outFileNamePrefix""", prefix, """--outSAMtype""", "BAM Unsorted", "--
outSAMattributes", "All","--outSAMstrandField", "intronMotif", "--outSAMattrIHstart", "0"]

        STARcommand_list = ["STAR","""--runThreadN""",str(pthreads)] + trim + multi_align +
single_reads + discordant + inputs + outputs + add_options
        STARcommand=" ".join(STARcommand_list)
        sp.check_call(["/bin/sh", "-c", STARcommand])

        STAR_output = prefix + "Aligned.out.bam"

        sortcommand_list = ["samtools", "sort", "-@",str(pthreads), STAR_output, prefix]
        sortcommand = " ".join(sortcommand_list)
        sp.check_call(["/bin/sh", "-c", sortcommand])

        indexcommand_list = ["samtools", "index", outfile]
        indexcommand = " ".join(indexcommand_list)
        sp.check_call(["/bin/sh", "-c", indexcommand])
```

```
    os.unlink(STAR_output)
    os.unlink(prefix + "Log.out")
    os.unlink(prefix + "Log.progress.out")
    rename_file(prefix + "Log.final.out",prefix +".log")

def align_unpaired(fastq,pthreads,trim3,index,outfile,gtf,gzip,prefix,read_length,extra_fa):
    ##### ALIGN FASTQ FILE(S) TO GENOME OR REPCHR ########
    #-p16: allows hyperhreading over 16 cores
    #-t: outputs time of alignment
    #-v3: allows maximum of 3 mismatches to account for population variants, increases stringency
of Tag finding
    #-a -m1:  reports all valid alignments for reads with only 1 reportable alignment
    #-3 trim3: trims user-specified bases from 3' end of FASTQ sequences (useful for if sequencing
read > subsequence length)

    gtf_option = []
    gzip_option = []
    extra_option = []
    if gtf:
       gtf_option = ["--sjdbGTFfile", gtf, "--sjdbOverhang",str(read_length-1), "--twopassMode",
"Basic"]
    if gzip:
       gzip_option = ["""--readFilesCommand""", "zcat"]
    if extra_fa:
       extra_option=["""--genomeFastaFiles""",extra_fa]
    add_options = gtf_option + gzip_option + extra_option

    multi_align = ["""--outFilterMultimapNmax""", "100", """--winAnchorMultimapNmax""",
"100"]
    trim = ["""--clip3pNbases""", str(trim3)]
    inputs = ["""--genomeDir""", index,"""--readFilesIn""",fastq]
    outputs = [ """--outFileNamePrefix""", prefix, """--outSAMtype""", "BAM Unsorted", "--
outSAMattributes", "All","--outSAMstrandField", "intronMotif", "--outSAMattrIHstart", "0"]

    STARcommand_list = ["STAR","""--runThreadN""",str(pthreads)] + trim + multi_align + inputs
+ outputs + add_options
    STARcommand=" ".join(STARcommand_list)
    sp.check_call(["/bin/sh", "-c", STARcommand])
    STAR_output = prefix + "Aligned.out.bam"

    sortcommand_list = ["samtools", "sort", "-@",str(pthreads), STAR_output, prefix]
    sortcommand = " ".join(sortcommand_list)
    sp.check_call(["/bin/sh", "-c", sortcommand])


    indexcommand_list = ["samtools", "index", outfile]
    indexcommand = " ".join(indexcommand_list)
    sp.check_call(["/bin/sh", "-c", indexcommand])

    os.unlink(STAR_output)
```

```
        # os.unlink(prefix + "Log.out")
        # os.unlink(prefix + "Log.progress.out")
        rename_file(prefix + "Log.final.out",prefix +".log")

def get_header(bamfile,headerfile):
    samtoolscommand_list = ["samtools","view","-H", bamfile, ">",headerfile]
    samtoolscommand = " ".join(samtoolscommand_list)
    sp.check_call(["/bin/sh", "-c", samtoolscommand])

def mask_reads(infile,extra,chrom_list,basename,outfolder,pthreads,debug):
    read_dict={}
    sam_temp=make_tempfile(basename,"sam_temp",outfolder)
    ectopic_alignments = make_tempfile(basename,"ectopic",outfolder)
    ectopic_reads = infile.replace(".bam","_ectopic.bam")
    nonectopic_reads = infile.replace(".bam","_masked.bam")
    with open(extra,'r') as nonreftable:
        for line in nonreftable:
            line=line.rstrip()
            line=line.split("\t")
            chrom=line[0]
            strand=line[3]
            TE_type=line[5].lower()
            if strand=="Strand":
                continue
            if TE_type=="plasmid":
                chrom_list.append(chrom)
            elif TE_type=="transgene":
                chrom_list.append(chrom)

        get_header(infile,ectopic_reads)
        get_header(infile,nonectopic_reads)

    for chrom in chrom_list:
        search="""""""+ '$3 ~ /' + chrom + '/' + """""""
        dupe_command_list = ["samtools","view",infile,chrom, ">", ectopic_alignments] #skips lines if
the read has already appeared in the file
        dupe_command = " ".join(dupe_command_list)
        sp.check_call(["/bin/sh", "-c", dupe_command])

        awkcommand_list = ["samtools","view",  infile,  ">",sam_temp] #writes lines in
combined_tempfile that are not in unique_tempfile2 -> duplicates
        awkcommand = " ".join(awkcommand_list)
        sp.check_call(["/bin/sh","-c",awkcommand])


        awkcommand_list = ["awk", """'FNR==NR{a[$1]++;next}a[$1]'""", ectopic_alignments,
sam_temp,  ">>", ectopic_reads] #writes lines in combined_tempfile that are not in unique_tempfile2
-> duplicates
        awkcommand = " ".join(awkcommand_list)
        sp.check_call(["/bin/sh","-c",awkcommand])
```

```
        awkcommand_list = ["awk", """'FNR==NR{a[$1]++;next}!a[$1]'""",  ectopic_alignments,
sam_temp,  ">", nonectopic_reads] #writes lines in combined_tempfile that are not in
unique_tempfile2 -> duplicates
        awkcommand = " ".join(awkcommand_list)
        sp.check_call(["/bin/sh","-c",awkcommand])
        if not debug:
            os.unlink(ectopic_alignments)

def main(**kwargs):
    ######## ARGUMENTS ###########
    #check if already args is provided, i.e. main() is called from the top level script
    args = kwargs.get('args', None)
    if args is None: ## i.e. standalone script called from command line in normal way
        parser = argparse.ArgumentParser(description = """Aligns RNAseq reads to STAR index
allowing for multiple alignments""")
        parser._optionals.title = "Arguments"
        parser.add_argument("-1","--read1", help = "RNASeq data fastq file; read1 if providing paired
end data. If more than one file, separate with commas, no spaces. Can be gzipped. (Required for
single-end data; optional for paired-end)", type = str, metavar = "<file_1.fastq or file_1.fastq.gz>")
        parser.add_argument("-2","--read2", help = "RNASeq data read2 fastq file. if more than one file,
separate with commas, no spaces. Can be gzipped.  (optional, can skip or enter 'False' if data is
unpaired)", type = str, metavar = "<file_2.fastq or file_2.fastq.gz>")
        parser.add_argument("-o","--map_folder", help = "Location of SQuIRE Map outputs (optional,
default = 'squire_map')", type = str, metavar = "<folder>", default = "squire_map")
        parser.add_argument("-f","--fetch_folder", help = "Folder location of outputs from SQuIRE
Fetch (optional, default = 'squire_fetch'",type = str, metavar = "<folder>",default="squire_fetch")
        parser.add_argument("-r","--read_length", help = "Read length (if trim3 selected, after trimming;
required).", type = int, metavar = "<int>", required=True)
        parser.add_argument("-n","--name", help = "Common basename for input files (optional; uses
basename of read1 as default)", type = str, metavar = "<str>",default=False)
        parser.add_argument("-3","--trim3", help = "Trim <int> bases from right end of each read before
alignment (optional; default=0).", type = int, default = 0, metavar = "<int>")
        parser.add_argument("-e","--extra", help = "Filepath of text file containing non-reference repeat
sequence and genome information", type=str, metavar = "<file.txt>")
        parser.add_argument("-b","--build", help = "UCSC designation for genome build, eg. 'hg38'
(required if more than 1 build in clean_folder)", type=str, metavar = "<build>",default=False)
        parser.add_argument("-p","--pthreads", help = "Launch <int> parallel threads(optional;
default='1')", type = int, metavar = "<int>", default=1)
        parser.add_argument("-v","--verbosity", help = "Want messages and runtime printed to stderr
(optional; default=False)", action = "store_true", default = False)
        args,extra_args = parser.parse_known_args()

########## I/O #########
    ###### ARGUMENTS ######
    read1=args.read1
    read2=args.read2
    outfolder = args.map_folder
    read_length = args.read_length
    fetch_folder=args.fetch_folder
    #index = args.index
    basename = args.name
```

```
trim3 = args.trim3
extra=args.extra
build=args.build
#gtf=args.gtf
# mask=args.mask
pthreads = args.pthreads
verbosity=args.verbosity

######### START TIMING SCRIPT #############
if verbosity:
   startTime = datetime.now()
   print("start time is:" + str(startTime) + '\n', file = sys.stderr)# Prints start time
   print(os.path.basename(__file__) + '\n', file = sys.stderr) #prints script name to std err
   print("Script Arguments" + '\n' + "==================", file = sys.stderr)
   args_dict = vars(args)
   for option,arg in args_dict.iteritems():
      print(str(option) + "=" + str(arg), file = sys.stderr) #prints all arguments to std err
   print("\n", file = sys.stderr)

#### SET DEFAULTS #####

if not read1 and not read2:
   raise Exception("read1 or read2 must be provided")
if read2:
   if read2.lower()=="false":
      read2=False
debug=True

### CHECK INPUTS#####
index = find_file(fetch_folder,"_STAR",build, 1,True)

gtf = find_file("squire_fetch","_refGene.gtf",build, 1,True)

if not basename:
   basename = get_basename(read1)
make_dir(outfolder)
outfile = outfolder + "/" + basename + ".bam"

prefix = outfolder + "/" + basename
if ".gz" in read1:
   gzip=True
else:
   gzip = False

extra_fapath=False

if extra:
   extra_fapath = outfolder + "/" + get_basename(extra) +  ".fa"
   extra_fa=open(extra_fapath,'wb')
   if verbosity:
      print("Making fasta file from extra file" + "\n", file = sys.stderr)
```

```
    previous_chrom=0 #This is needed to avoid reopening chromosome sequence files, which would make the script run time a lot longer.
    buffer_sequence = "N" * 200
    chrom_dict = {}
    seq_dict=defaultdict(str)
    maskchrom_list=[]

    with open(extra,'r') as extra_file:
      nonref_types=["polymorphism","novel","plasmid","transgene"]
      for line in extra_file:
        line = line.rstrip()
        line=line.split("\t")
        chrom=line[0]
        start = line[1]
        stop = line[2]
        strand = line[3]
        if strand.lower()=="strand":
          continue
        taxo = line[4]
        TE_type=line[5].lower()
        if TE_type not in nonref_types:
          raise Exception('TE type needs to be "polymorphism","novel","plasmid",or "transgene"')
        chrom = chrom + "_" + TE_type
        if not chrom.startswith("chr"):
          chrom="chr"+chrom
        #chrom=chrom + "_" + TE_type
        if "plasmid" in TE_type: #if plasmid
          score = "999"
          maskchrom_list.append(chrom)
        elif "transgene" in TE_type:
          score="999"
          maskchrom_list.append(chrom)
        else: #if insertion polymorphism
          score = "1000"
        left_flankseq = line[6]
        right_flankseq = line[7]
        TEsequence = line[8]

        sequence=left_flankseq + TEsequence + right_flankseq

        seq_dict[chrom] += sequence + buffer_sequence

    for chrom, sequence in seq_dict.iteritems():
      extra_fa.writelines(">" + chrom + "\n")
      new_chromseq = textwrap.fill(seq_dict[chrom],50)
      extra_fa.writelines(new_chromseq + "\n")

    extra_fa.close()
  else:
    extra_fa=None
```

```
    if read1 and not read2: #if single-end
        if read1.endswith(","):
            read1=read1[:-1]
        if verbosity:
            print("Aligning FastQ files " + str(datetime.now()) + "\n",file = sys.stderr)
        align_unpaired(read1,pthreads,trim3,index,outfile,gtf,gzip,prefix, read_length,extra_fapath)

    if read1 and read2:
        if read1.endswith(","):
            read1=read1[:-1]
        if read2.endswith(","):
            read2=read2[:-1]
        if verbosity:
            print("Aligning FastQ files for Read1 and Read2 " + str(datetime.now()) + "\n",file =
sys.stderr)
        align_paired(read1,read2,pthreads,trim3,index,outfile,gtf,gzip,prefix, read_length,extra_fapath)


    # if mask:
    #     mask_reads(outfile,chrom_list,basename,outfolder,pthreads,debug)


    ####### STOP TIMING SCRIPT ########################
    if verbosity:
        print("finished writing outputs" + "\n",file = sys.stderr)

        endTime = datetime.now()
        print('end time is: '+ str(endTime) + "\n", file = sys.stderr)
        print('it took: ' + str(endTime-startTime) + "\n", file = sys.stderr)

####################
if __name__ == "__main__":
    main()
```

Appendix E. SQuIRE Count

```
#!/usr/bin/env python

############MODULES############################
from __future__ import print_function,division
import sys
import os
import errno
import re
import argparse #module that passes command-line arguments into script
from datetime import datetime
import operator #for doing operations on tuple
from operator import itemgetter
import subprocess as sp
from subprocess import Popen, PIPE,STDOUT
import io
import tempfile
#for creating interval from start
from collections import defaultdict #for dictionary
import glob
import re
from six import itervalues
import shutil

#########################################################
#########INITIATE VARIABLES
#########################################################

RepCalc_dict = {}
subF_reads = defaultdict(int)

####### FUNCTIONS ###############################
def isempty(filepath):
    if os.path.getsize(filepath) == 0:
            raise Exception(filepath + " is empty")

def get_basename(filepath):
            filename = os.path.basename(filepath)
            filebase = os.path.splitext(filename)[0]
            return filebase

def make_dir(path):
    try:
            original_umask = os.umask(0)
            os.makedirs(path, 0770)
    except OSError as exception:
            if exception.errno != errno.EEXIST:
                    raise
    finally:
            os.umask(original_umask)
```

```
    def find_file(folder,pattern,base, wildpos, needed):
        foundfile=False
        if wildpos == 1:
            file_list=glob.glob(folder + "/" + "*" + pattern)
        elif wildpos ==2:
            file_list=glob.glob(folder + "/" + pattern + "*")
        if len(file_list)>1: #if more than one file in folder
            if not base:
                raise Exception("More than 1 " + pattern + " file")
            for i in file_list:
                if base in i:
                    foundfile = i
        elif len(file_list) == 0:
            foundfile = False
        else:
            foundfile = file_list[0]
        if not foundfile:
            if needed:
                raise Exception("No " + pattern + " file")
            else:
                foundfile = False
        return foundfile


    def rename_file(oldname,newname):
            shutil.move(oldname, newname)

            ####create tempfiles ###
    def make_tempfile(basename, step, outfolder):
            tmpfile = tempfile.NamedTemporaryFile(delete=False, dir = outfolder, prefix= basename +
"_" + step +  ".tmp")
            tmpname = tmpfile.name
            tmpfile.close()
            return tmpname

    def getlibsize(logfile, infile,multi_bed,uniq_bed,paired_end,debug):
            if logfile:
                    STAR_logfile=open(logfile,'r')
                    for line in STAR_logfile:
                            line = line.strip()
                            unique_string = """Uniquely mapped reads number"""
                            multi_string = """Number of reads mapped to multiple loci"""
                            if unique_string in line:
                                    unique_libsize  = int(re.search("\d+",line).group(0))
                            elif multi_string in line:
                                    multi_libsize =int(re.search("\d+",line).group(0))
                    libsize = (unique_libsize + multi_libsize)/2
                    STAR_logfile.close()
            else:
                    count_temp = infile + "libsize"
```

```
                linecountcommandlist = ["samtools", "view", infile, "|", "cut", "-f1", "|", "sort", "-k1,1", "|" , "uniq","|", "wc -l", ">", count_temp]
                linecountcommand = " ".join(linecountcommandlist)
                sp.check_call(["/bin/sh","-c",linecountcommand])
                with open(count_temp, 'r') as count_file:
                        first_line = count_file.readline()
                        first_line_split = first_line.split()
                        libsize = int(first_line_split[0])
                if paired_end:
                        libsize = libsize/2
                if not debug:
                        os.unlink(count_temp)
        return libsize


    def getlinecount(first_file,name):
        count_temp = first_file +"_" + name +  ".libsize"
        linecountcommandlist = ["wc","-l",first_file,">", count_temp]
        linecountcommand = " ".join(linecountcommandlist)
        sp.check_call(["/bin/sh","-c",linecountcommand])

        with open(count_temp, 'r') as count_file:
                first_line = count_file.readline()
                first_line_split = first_line.split()
                libsize = first_line_split[0]
                return int(libsize)

        # os.unlink(count_temp)


    def Stringtie(bamfile,outfolder,basename,strandedness,pthreads,gtf, verbosity,outgtf):
      ###Stringtie parameters
      extra_files=True
      if strandedness ==1:
        stringtie_strand = "--rf"
      elif strandedness == 2:
        stringtie_strand = "--fr"
      else:
        stringtie_strand = ""
      if gtf:
        inputs = ["-G", gtf, bamfile]
        pct_max_fpkm=0.1
        flanklength = 10
        flankdepth = 1
        read_gap = 50
        min_tx_length=200
        max_multi_pct = .95
        min_coverage = 2.5
        TEoptions = [stringtie_strand, "-f",str(pct_max_fpkm),"-m", str(min_tx_length), "-a", str(flanklength), "-j", str(flankdepth), "-g", str(read_gap), "-M", str(max_multi_pct), "-c", str(min_coverage), "-e"]
```

```
        else:
            inputs = [bamfile]
            pct_max_fpkm=0.1
            flanklength = 10
            flankdepth = .1
            read_gap = 50
            min_tx_length=200
            max_multi_pct = 1.0
            min_coverage = 1.5
            TEoptions = [stringtie_strand,"-l",basename, "-f",str(pct_max_fpkm),"-m",
str(min_tx_length), "-a", str(flanklength), "-j", str(flankdepth), "-g", str(read_gap), "-M",
str(max_multi_pct), "-c", str(min_coverage), "-t"]
        runoptions = ["-p", str(pthreads), ]
        if verbosity:
            if gtf:
                print("Running Guided Stringtie on each bamfile " + basename + " " + str(datetime.now())
+ "\n",file = sys.stderr)
            else:
                print("Running Unguided Stringtie on each bamfile " + basename + " " +
str(datetime.now()) + "\n",file = sys.stderr)
        outputs=["-o", outgtf]
        if extra_files:
            out_abund = outgtf.replace("outgtf","outabund")
            outputs= outputs + ["-A", out_abund]
        StringTiecommand_list = ["stringtie"] + runoptions + TEoptions + outputs + inputs
        StringTiecommand=" ".join(StringTiecommand_list)
        sp.check_call(["/bin/sh", "-c", StringTiecommand])


    class gtfline(object):
        def __init__(self,line):
                self.line=line
                self.chrom = line[0]
                self.source=line[1]
                self.category=line[2]
                self.start = (int(line[3])-1)
                self.stop = int(line[4])
                self.score=(line[5])
                self.strand  = line[6]
                self.frame=line[7]
                self.attributes=line[8].split("; ")
                for attribute_pair in self.attributes:
                        self.attribute = attribute_pair.replace(" ","").split("")
                        if self.attribute[0]=="FPKM":
                                self.fpkm=float(self.attribute[1])
                        elif self.attribute[0]=="TPM":
                                self.tpm=float(self.attribute[1])
                        elif self.attribute[0]=="gene_id":
                                self.Gene_ID=self.attribute[1]
                        elif self.attribute[0]=="cov" :
                                self.coverage=float(self.attribute[1])
```

```
                elif self.attribute[0]== "transcript_id":
                        self.transcript_id=self.attribute[1]
        def replace_geneid(self,newgeneid):
                newgeneid=[str(x) for x in newgeneid]
                newgeneid=",".join(newgeneid)
                self.attributes[0] = "gene_id" + " " + "" + newgeneid + ""
                attributesout = self.attributes= "; ".join(self.attributes)
                gtfout =
[self.chrom,self.source,self.category,self.start+1,self.stop,self.score,self.strand,self.frame,attributesout
]
                self.gtfout = [str(i) for i in gtfout]

    def filter_tx(infile,gene_dict,read_length,genecounts):
        with open(infile,'r') as filterin:
                header=filterin.readline()
                for line in filterin:
                        if line.startswith("#"):
                                continue
                        line = line.rstrip()
                        line = line.split("\t")
                        gtf_line = gtfline(line[0:9])
                        if len(line) == 9:
                                if gtf_line.category=="exon":
                                        transcribed_length=int(gtf_line.stop) - int(gtf_line.start)
                                        counts =
gtf_line.coverage*transcribed_length/int(read_length)
                                        if counts > 0:

        gene_dict[(gtf_line.Gene_ID,gtf_line.strand)].add_counts(counts)

        gene_dict[(gtf_line.Gene_ID,gtf_line.strand)].add_tx(gtf_line.transcript_id)
                        else:
                                ref_line=gtfline(line[9:18])

        gene_dict[(ref_line.Gene_ID,ref_line.strand)].add_tx(gtf_line.transcript_id)
        with open(genecounts,'w') as outfile:
                for genestrand,geneinfo in gene_dict.iteritems():
                        outline="\t".join(geneinfo.countsout)
                        outfile.writelines(outline+"\n")

    class gene_info(object):
        def __init__(self,line):
                self.Gene_ID = line[0]
                self.Gene_name = line[1]
                self.chrom = line[2]
                self.strand  = line[3]
                self.start = str(int(line[4])-1) #changes from 1 base to 0-base
                self.stop = int(line[5])
                self.coverage = float(line[6])
                self.fpkm = float(line[7])
                self.tpm = float(line[8])
```

```
                self.counts=0
                self.tx_IDs=set()
                self.tx_ID_string=",".join(self.tx_IDs)
                self.flagout=[self.Gene_ID,self.fpkm,self.counts]

        self.countsout=[self.chrom,self.start,self.stop,self.Gene_ID,self.fpkm,self.strand,int(round(sel
f.counts)),self.tx_ID_string]
                self.countsout = [str(i) for i in self.countsout]
        def add_counts(self,counts):
                self.counts += counts
        def add_tx(self,txID):
                self.tx_IDs.add(txID)
                self.tx_ID_string=",".join(self.tx_IDs)
                self.flagout = [self.Gene_ID,self.fpkm,self.counts]
                self.countsout =
[self.chrom,self.start,self.stop,self.Gene_ID,self.fpkm,self.strand,int(round(self.counts)),self.tx_ID_st
ring]
                self.countsout = [str(i) for i in self.countsout]

    def filter_abund(infile,gene_dict,notinref_dict):
        with open(infile,'r') as filterin:
                for line in filterin:
                        line = line.rstrip()
                        line = line.split("\t")
                        if "Gene" in line[0] and "TPM" in line[-1]:
                                continue
                        gene_data=gene_info(line)
                        if not notinref_dict:
                                gene_dict[(gene_data.Gene_ID,gene_data.strand)] = gene_data
                        else:
                                if gene_data.Gene_ID in notinref_dict:
                                        gene_dict[(gene_data.Gene_ID,gene_data.strand)] =
gene_data


    def intersect(bamfile,bedfile,out_bed):
        ######## INTERSECT WITH BED FILE ##########################
        intersect_list = ["bedtools", "intersect", "-a",bamfile,"-b",bedfile,"-wo", "-bed",">",out_bed]
        intersect_command = " ".join(intersect_list)
        sp.check_call(["/bin/sh", "-c", intersect_command])

    def intersect_flank(bamfile,bedfile,out_bed,debug):
        ######## INTERSECT WITH BED FILE ##########################
        #keep read if 50% of read overlaps with TE range
        intersect_list = ["bedtools", "intersect", "-a",bamfile,"-b",bedfile,"-wo", "-bed","-f",
".5",">",out_bed]
        intersect_command = " ".join(intersect_list)
        sp.check_call(["/bin/sh", "-c", intersect_command])

    def label_files(file_in,file_out, string,debug):
        command = "'{print $0," + "" + string + "" + "}'"
```

```
        pastecommandlist = ["awk", "-v", "OFS='\\t'",command,file_in, ">", file_out]
        pastecommand = " ".join(pastecommandlist)
        sp.check_call(["/bin/sh","-c",pastecommand])
        if not debug:
                os.unlink(file_in)
    def combine_files(file1,file2,outfile,debug):
        catcommand_list = ["cat", file1, file2, ">", outfile] #combines multi_aligned reads
        catcommand = " ".join(catcommand_list)
        sp.check_call(["/bin/sh","-c",catcommand])

        if not debug:
                os.unlink(file1)
                os.unlink(file2)

    def sort_temp(tempfile, field,sorted_tempfile,debug):
        field_command = str(field) + "," + str(field)
        sort_command_list = ["sort","-k",field_command, tempfile, ">", sorted_tempfile]
        sort_command = " ".join(sort_command_list)
        sp.check_call(["/bin/sh", "-c", sort_command])
        if not debug:
                os.unlink(tempfile)

    def get_header(bamfile,headerfile):
        samtoolscommand_list = ["samtools","view","-H", bamfile, ">",headerfile]
        samtoolscommand = " ".join(samtoolscommand_list)
        sp.check_call(["/bin/sh", "-c", samtoolscommand])

    def is_paired(bamfile,basename,tempfolder,debug):
        bam_temp = make_tempfile(basename,"bam_header",tempfolder)
        get_header(bamfile,bam_temp)
        with open(bam_temp,'r') as header:
                for line in header:
                        if line.startswith("@CO"):
                                fastq=re.search("--readFilesIn(.+)--
outFileNamePrefix",line).group(1)
                                fastq_list = fastq.split()
                                if len(fastq_list) > 1:
                                        paired = True
                                else:
                                        paired = False
        if not debug:
                os.unlink(bam_temp)
        return paired


    def find_properpair(paired_bam, proper,nonproper):
        ##### FILTER INTO CONCORDANT AND DISCORDANT/SINGLE READS ####
        #-b: output in BAM format
        #-h: keep header
        #-S: input is SAM File
        #-F4: skip unmapped reads (bit flag = 4)
```

```
        #-f2 = keep proper pair
        #-F2 = discard proper pair
        samtoolscommand_list = ["samtools","view","-bf2", "-o", proper, paired_bam]
        samtoolscommand = " ".join(samtoolscommand_list)
        sp.check_call(["/bin/sh", "-c", samtoolscommand])
        samtoolscommand_list = ["samtools","view","-bF2", "-o", nonproper, paired_bam]
        samtoolscommand = " ".join(samtoolscommand_list)
        sp.check_call(["/bin/sh", "-c", samtoolscommand])

    def split_paired(paired_bed, paired_bed1, paired_bed2,debug):
        #separate read 1 and read2 into separate files
        awkcommand_list = ["awk","'$4 ~ v'","v='/1'", paired_bed,">", paired_bed1]
        awkcommand = " ".join(awkcommand_list)
        sp.check_call(["/bin/sh", "-c", awkcommand])
        awkcommand_list = ["awk","'$4 ~ v'","v='/2'", paired_bed,">", paired_bed2]
        awkcommand = " ".join(awkcommand_list)
        sp.check_call(["/bin/sh", "-c", awkcommand])
        if not debug:
                os.unlink(paired_bed)

    def reduce_reads(read_file,new_readfile,debug):
        #Find reads aligned to same position but different TE_IDs (overlapping flanks) and merge
        prev = False
        with open(read_file,'r') as infile:
                with open(new_readfile,'w') as outfile:
                        for line in infile:
                                if not prev:
                                        prev=bedline(line)
                                        prev.TE_ID = prev.line_split[15]
                                        prev_TE_ID = prev.TE_ID
                                        continue
                                else:
                                        current = bedline(line)
                                        current.TE_ID = current.line_split[15]
                                        if current.Read_ID == prev.Read_ID and current.Read_chr
== prev.Read_chr and current.Read_geno_start==prev.Read_geno_start and current.Read_geno_stop
== prev.Read_geno_stop and current.Read_strand == prev.Read_strand:
                                                if current.TE_ID != prev.TE_ID:
                                                        prev_TE_ID = prev_TE_ID + "&" +
current.TE_ID
                                        else:
                                                prev.line_split[15]  = prev_TE_ID
                                                prev.line = "\t".join(prev.line_split)
                                                outfile.writelines(prev.line + "\n")
                                                prev= current
                                                prev_TE_ID = current.TE_ID
        #end of loop
                        prev.line_split[15]  = prev_TE_ID
                        prev.line = "\t".join(prev.line_split)
                        outfile.writelines(prev.line + "\n")
        if not debug:
```

```
                os.unlink(read_file)


    def get_coords(file_in,read_end,strandedness, file_out,debug):
        ####Get genomic coordinates from bed file
        temp_file_coords = file_in + "_temp_coords"
        temp_file_chr = file_in + "_temp_chr"
        temp_file_plus = file_in + "_temp_plus"
        temp_file_minus = file_in + "_temp_minus"
        temp_file_new = file_in + "_temp_new"
        coords_commandlist = ["awk", "-v", "OFS='\\t'",'''{print $1 OFS $19-$14+$2 OFS $19-
$14+$3 OFS $4 OFS $5 OFS "orig_"$6 OFS $16 OFS $23}''',file_in, ">", temp_file_coords]
        coords_command = " ".join(coords_commandlist)
        sp.check_call(["/bin/sh","-c",coords_command])
        remove_underscore_command_list = ["awk","-v", "OFS='\\t'", '''{
gsub(/_polymorphism/,"",$1); gsub(/_novel/,"",$1);print $0 }''', temp_file_coords, ">",
temp_file_chr]
        remove_underscore_command = " ".join(remove_underscore_command_list)
        sp.check_call(["/bin/sh","-c",remove_underscore_command])
        if not debug:
                os.unlink(temp_file_coords)
                os.unlink(file_in)
        if strandedness==0:
                strandedness=1 #change strandedness just so paired-end reads are switched to the
same strand
        if strandedness == read_end: #switch strand
                plus_command_list = ["awk","-v", "OFS='\\t'", '''{ gsub(/orig_\+/,"new_-",$6);
print $0 }''', temp_file_chr, ">", temp_file_plus]
                plus_command = " ".join(plus_command_list)
                sp.check_call(["/bin/sh","-c",plus_command])
                minus_command_list = ["awk", "-v", "OFS='\\t'",'''{ gsub(/orig_\-/,"new_+",$6);
print $0 }''', temp_file_plus, ">", temp_file_minus]
                minus_command = " ".join(minus_command_list)
                sp.check_call(["/bin/sh","-c",minus_command])
                new_command_list = ["awk", "-v", "OFS='\\t'",'''{ gsub("new_","",$6); print $0
}''', temp_file_minus, ">", file_out]
                new_command = " ".join(new_command_list)
                sp.check_call(["/bin/sh","-c",new_command])
                if not debug:
                        os.unlink(temp_file_chr)
                        os.unlink(temp_file_plus)
                        os.unlink(temp_file_minus)
        else: #keep strand
                new_command_list = ["awk","-v", "OFS='\\t'", '''{ gsub("orig_","",$6); print $0
}''', temp_file_chr, ">", file_out]
                new_command = " ".join(new_command_list)
                sp.check_call(["/bin/sh","-c",new_command])
                if not debug:
                        os.unlink(temp_file_chr)

    def fix_paired(file1,file2,fixed_file1,fixed_file2, debug): #remove "/1" or "/2"
```

```
        remove1_command_list = ["sed", """s@/1@@g""", file1, ">", fixed_file1]
        remove1_command = " ".join(remove1_command_list)
        sp.check_call(["/bin/sh","-c",remove1_command])
        remove2_command_list = ["sed", """s@/2@@g""", file2, ">", fixed_file2]
        remove2_command = " ".join(remove2_command_list)
        sp.check_call(["/bin/sh","-c",remove2_command])
        if not debug:
                os.unlink(file1)
                os.unlink(file2)

    def find_uniq(combined_tempfile, first_tempfile,unique_tempfile, multi_tempfile,debug):
        ##### SEPARATE UNIQUELY ALIGNED AND MULTI-ALIGNED READS ########
        dupe_tempfile = combined_tempfile + "_dupe"
        dupe_tempfile1 = combined_tempfile + "_dupe1"
        dupe_command_list = ["awk","'!a[$4]++'", combined_tempfile, ">", first_tempfile] #skips
lines if the read has already appeared in the file
        dupe_command = " ".join(dupe_command_list)
        sp.check_call(["/bin/sh", "-c", dupe_command])
        awkcommand_list = ["awk", """FNR==NR{a[$0]++;next}!a[$0]""", first_tempfile,
combined_tempfile,  ">", dupe_tempfile] #writes lines in combined_tempfile that are not in
unique_tempfile2 -> duplicates
        awkcommand = " ".join(awkcommand_list)
        sp.check_call(["/bin/sh","-c",awkcommand])
        awkcommand_list = ["awk", """FNR==NR{a[$4]++;next}!a[$4]{print $0}""",
dupe_tempfile, first_tempfile,  ">", unique_tempfile] #writes lines where read is in unique2 but not
multi file -> truly unique
        awkcommand = " ".join(awkcommand_list)
        sp.check_call(["/bin/sh","-c",awkcommand])
        awkcommand_list = ["awk", """FNR==NR{a[$4]++;next}a[$4]{print $0}""",
dupe_tempfile, first_tempfile,  ">", dupe_tempfile1] #writes lines in read is in unique2 and multi file
-> gets first appearance of multi-aligned reads
        awkcommand = " ".join(awkcommand_list)
        sp.check_call(["/bin/sh","-c",awkcommand])
        #delete unneeded tempfiles
        catcommand_list = ["cat", dupe_tempfile, dupe_tempfile1, ">", multi_tempfile ] #combines
multi_aligned reads
        catcommand = " ".join(catcommand_list)
        sp.check_call(["/bin/sh","-c",catcommand])
        if not debug:
                os.unlink(dupe_tempfile)
                os.unlink(dupe_tempfile1)
                os.unlink(combined_tempfile)
                os.unlink(first_tempfile)

    def match_reads(R1, R2, strandedness, matched_file,unmatched_file1,unmatched_file2,debug):
        #match  read1 and read2 if within 2kb of each other on same strand
        #add rough location to read_ID to reduce combinations for join
        rounded_1_v1 = R1 + "_rounded_v1"
        rounded_2_v1 = R2 + "_rounded_v1"
        newread_1_v1 = R1 + "_newread_v1"
        newread_2_v1 = R2 + "_newread_v1"
```

```
        rounded_1_v2 = R1 + "_rounded_v2"
        rounded_2_v2 = R2 + "_rounded_v2"
        newread_1_v2 = R1 + "_newread_v2"
        newread_2_v2 = R2 + "_newread_v2"
        matched_file_v1 = matched_file + "_v1"
        matched_file_v2 = matched_file + "_v2"
        matched_file_10k_v1 = matched_file + "_10k_v1"
        matched_file_10k_v2 = matched_file + "_10k_v2"
        unmatched_file1_v1 = unmatched_file1 + "_v1"
        unmatched_file2_v1 = unmatched_file2 + "_v1"
        roundcommand_list = ["awk", "-v", "OFS='\\t'","-v", "FS='\\t'", """{print $0, $2/10000}""",
"""OFMT='%.f"""", R1, ">", rounded_1_v1]
        roundcommand=" ".join(roundcommand_list)
        sp.check_call(["/bin/sh","-c",roundcommand])
        roundcommand_list = ["awk", "-v", "OFS='\\t'","-v", "FS='\\t'", """{print $0, $2/10000}""",
"""OFMT='%.f"""", R2, ">", rounded_2_v1]
        roundcommand=" ".join(roundcommand_list)
        sp.check_call(["/bin/sh","-c",roundcommand])
        #create new read to join on that is read/chro
        newreadcommand_list = ["awk", "-v", "OFS='\\t'","-v", "FS='\\t'", """{print $0, $4 "/" $1 "/"
$11 "/" $6}""", rounded_1_v1,"|", "sort -k12", ">", newread_1_v1]
        newreadcommand=" ".join(newreadcommand_list)
        sp.check_call(["/bin/sh","-c",newreadcommand])
        newreadcommand_list = ["awk", "-v", "OFS='\\t'","-v", "FS='\\t'", """{print $0, $4 "/" $1 "/"
$11 "/" $6}""",  rounded_2_v1,"|", "sort -k12", ">", newread_2_v1]
        newreadcommand=" ".join(newreadcommand_list)
        sp.check_call(["/bin/sh","-c",newreadcommand])
        #use join not awk because awk only takes 1st hit with shared value to find match
        joincommand_list = ["join", "-j", "12", "-t", "$'\\t'", "-o",
"1.1,1.2,1.3,1.4,1.5,1.6,1.7,1.8,1.9,1.10,2.1,2.2,2.3,2.4,2.5,2.6,2.7,2.8,2.9,2.10", newread_1_v1,
newread_2_v1, ">" , matched_file_10k_v1]
        joincommand=" ".join(joincommand_list)
        sp.check_call(["/bin/sh","-c",joincommand])
        pos_strand_2 =  """($3 -$12 <= 500 && $3 -$12 >= 0 && $2 >= $12 && $6=="+" &&
$5!=1000 && $15!=1000)""" #insert size < 500 & end of read1 will be after beginning of read 2 &
start of read1 will be after beginning of read2
        minus_strand_2 = """($13 -  $2 <= 500 && $13 -  $2 >= 0 && $12 >= $2 && $6=="-" &&
$5!=1000 && $15!=1000)""" #insert size < 500 & end of read2 will be after beginning of read 1 &
start of read2 will be after beginning of read1
        poly_pos_2 = """($3 -$12 <= 500 && $3 -$12 >= 0 && $2 >= $12 && $6=="+" &&
$5==1000 && $15 !=1000) || ($13 -  $2 <= 500 && $13-  $2 >= 0 && $12 >= $2 && $6=="+" &&
$5!=1000 && $15==1000)||($3 -$12 <= 500 && $3 -$12 >= 0 && $2 >= $12 && $6=="+" &&
$5==1000 && $15==1000)"""
        poly_minus_2 =  """($3 -$12 <= 500 && $3 -$12 >= 0 && $2 >= $12 && $6=="-" &&
$5!=1000 && $15 ==1000) || ($13 -  $2 <= 500 && $13 -  $2 >= 0 && $12 >= $2 && $6=="-" &&
$5==1000 && $15!=1000)||($13 -  $2 <= 500 && $13 -  $2 >= 0 && $12 >= $2 && $6=="-"  &&
$5==1000 && $15==1000)"""#if plus strand: pos_strand 2 before insert, minus_strand 2 after; if
minus strand: minus strand_2 before insert, pos strand 2 after
        pos_strand_1 =  """($13 -  $2 <= 500 && $13 -  $2 >= 0 && $12 >= $2 && $6=="+" &&
$5!=1000)""" #insert size < 500 & end of read2 will be after beginning of read 1 & start of read2 will
be after beginning of read1
```

```
        minus_strand_1 = """($3 -$12 <= 500 && $3 -$12 >= 0 && $2 >= $12 && $6=="-" &&
$5!=1000)""" #insert size < 500 & end of read1 will be after beginning of read 2 & start of read1 will
be after beginning of read2
        poly_minus_1 = """($3 -$12 <= 500 && $3 -$12 >= 0 && $2 >= $12 && $6=="-" &&
$5==1000 && $15 !=1000) || ($13 -  $2 <= 500 && $13 -  $2 >= 0 && $12 >= $2 && $6=="-" &&
$5!=1000 && $15==1000)||($3 -$12 <= 500 && $3 -$12 >= 0 && $2 >= $12 && $6=="-" &&
$5==1000 && $15==1000)"""
        poly_pos_1 =  """($3 -$12 <= 500 && $3 -$12 >= 0 && $2 >= $12 && $6=="+" &&
$5!=1000 && $15 ==1000) || ($13 -  $2 <= 500 && $13 -  $2 >= 0 && $12 >= $2 && $6=="+" &&
$5==1000 && $15!=1000)||($13 -  $2 <= 500 && $13 -  $2 >= 0 && $12 >= $2 && $6=="+"  &&
$5==1000 && $15==1000)"""#if plus strand: pos_strand 2 before insert, minus_strand 2 after; if
minus strand: minus strand_2 before insert, pos strand 2 after
        unstranded = """($13 -  $2 <= 500 && $13 -  $2 >= 0 && $12 >= $2 ) || ($3 -$12 <= 500 &&
$3 -$12 >= 0 && $2 >= $12)"""
        awk_inout_v1 = [matched_file_10k_v1,  ">",  matched_file_v1]
        if strandedness==1: #first strand RNA synthesis (Illumina, dUTP, NSR, NNSR)
                awkcommand_list2 = ["awk", "-v", "OFS='\\t'","-v",
"FS='\\t'","""("""",pos_strand_2,"""||""",minus_strand_2,"||",poly_pos_2,"||",poly_minus_2,"){print
$0}"""]  #find pairs that match TE_ID and strand
                awkcommand2 = " ".join(awkcommand_list2 + awk_inout_v1)
                sp.check_call(["/bin/sh","-c",awkcommand2])
        if strandedness==2: #second strand (Ligation, standard Solid)
                awkcommand_list1 = ["awk", "-v", "OFS='\\t'","-v",
"FS='\\t'","""("""",pos_strand_1,"""||""", minus_strand_1 ,"||",poly_pos_1,"||",poly_minus_1,""")
{print $0}"""] #find pairs that match TE_ID and strand
                awkcommand1 = " ".join(awkcommand_list1 + awk_inout_v1)
                sp.check_call(["/bin/sh","-c",awkcommand1])
        if strandedness==0:
                awkcommand_list0 = ["awk", "-v", "OFS='\\t'","-v", "FS='\\t'","""("""",
unstranded,""") {print $0}"""] #find pairs that match TE_ID and strand
                awkcommand0 = " ".join(awkcommand_list0 + awk_inout_v1)
                sp.check_call(["/bin/sh","-c",awkcommand0])
        awkcommand_list = ["awk","-v", "OFS='\\t'","-v", "FS='\\t'",
"""'FNR==NR{a[$4]++;next}!a[$4]{print $0}'""", matched_file_v1,R1,   ">", unmatched_file1_v1]
#writes lines in read1 that is not in matched file -> unmatched
        awkcommand = " ".join(awkcommand_list)
        sp.check_call(["/bin/sh","-c",awkcommand])
        awkcommand_list = ["awk", "-v", "OFS='\\t'","-v",
"FS='\\t'","""'FNR==NR{a[$4]++;next}!a[$4]{print $0}'""", matched_file_v1, R2,  ">",
unmatched_file2_v1] #writes lines in read2 that is not in matched file -> unmatched
        awkcommand = " ".join(awkcommand_list)
        sp.check_call(["/bin/sh","-c",awkcommand])
        roundcommand_list = ["awk", "-v", "OFS='\\t'","-v", "FS='\\t'", """{print $0, $2/100000}""",
"""OFMT='%.f'""", unmatched_file1_v1, ">", rounded_1_v2]
        roundcommand=" ".join(roundcommand_list)
        sp.check_call(["/bin/sh","-c",roundcommand])
        roundcommand_list = ["awk", "-v", "OFS='\\t'","-v", "FS='\\t'", """{print $0, $2/100000}""",
"""OFMT='%.f'""", unmatched_file2_v1, ">", rounded_2_v2]
        roundcommand=" ".join(roundcommand_list)
        sp.check_call(["/bin/sh","-c",roundcommand])
```

```
        newreadcommand_list = ["awk", "-v", "OFS='\\t'","-v", "FS='\\t'", """{print $0, $4 "/" $1 "/"
$11 "/" $6}""", rounded_1_v2,"|", "sort -k12", ">", newread_1_v2]
        newreadcommand=" ".join(newreadcommand_list)
        sp.check_call(["/bin/sh","-c",newreadcommand])
        newreadcommand_list = ["awk", "-v", "OFS='\\t'","-v", "FS='\\t'", """{print $0, $4 "/" $1 "/"
$11 "/" $6}""",  rounded_2_v2,"|", "sort -k12", ">", newread_2_v2]
        newreadcommand=" ".join(newreadcommand_list)
        sp.check_call(["/bin/sh","-c",newreadcommand])
        #use join not awk because awk only takes 1st hit with shared value to find match
        joincommand_list = ["join", "-j", "12", "-t", "$'\\t'", "-o",
"1.1,1.2,1.3,1.4,1.5,1.6,1.7,1.8,1.9,1.10,2.1,2.2,2.3,2.4,2.5,2.6,2.7,2.8,2.9,2.10", newread_1_v2,
newread_2_v2, ">" , matched_file_10k_v2]
        joincommand=" ".join(joincommand_list)
        sp.check_call(["/bin/sh","-c",joincommand])
        awk_inout_v2 = [matched_file_10k_v2,  ">",  matched_file_v2]
        if strandedness==1:
                awkcommand2 = " ".join(awkcommand_list2+ awk_inout_v2)
                sp.check_call(["/bin/sh","-c",awkcommand2])
        if strandedness==2:
                awkcommand1 = " ".join(awkcommand_list1+ awk_inout_v2)
                sp.check_call(["/bin/sh","-c",awkcommand1])
        if strandedness==0:
                awkcommand0 = " ".join(awkcommand_list0+ awk_inout_v2)
                sp.check_call(["/bin/sh","-c",awkcommand0])
        catcommand_list = ["cat", matched_file_v1, matched_file_v2, ">", matched_file ] #combines
multi_aligned reads
        catcommand = " ".join(catcommand_list)
        sp.check_call(["/bin/sh","-c",catcommand])
        awkcommand_list = ["awk","-v", "OFS='\\t'","-v", "FS='\\t'",
"""FNR==NR{a[$4]++;next}!a[$4]{print $0}""", matched_file,R1,   ">", unmatched_file1] #writes
lines in read1 that is not in matched file -> unmatched
        awkcommand = " ".join(awkcommand_list)
        sp.check_call(["/bin/sh","-c",awkcommand])
        awkcommand_list = ["awk", "-v", "OFS='\\t'","-v",
"FS='\\t'","""FNR==NR{a[$4]++;next}!a[$4]{print $0}""", matched_file, R2,  ">",
unmatched_file2] #writes lines in read2 that is not in matched file -> unmatched
        awkcommand = " ".join(awkcommand_list)
        sp.check_call(["/bin/sh","-c",awkcommand])
        if not debug:
                os.unlink(rounded_1_v1)
                os.unlink(rounded_2_v1)
                os.unlink(newread_1_v1)
                os.unlink(newread_2_v1)
                os.unlink(rounded_1_v2)
                os.unlink(rounded_2_v2)
                os.unlink(newread_1_v2)
                os.unlink(newread_2_v2)
                os.unlink(matched_file_10k_v1)
                os.unlink(matched_file_10k_v2)
                os.unlink(matched_file_v1)
                os.unlink(matched_file_v2)
```

```
                os.unlink(unmatched_file1_v1)
                os.unlink(unmatched_file2_v1)
                os.unlink(R1)
                os.unlink(R2)



    def merge_coords(paired_file,merged_paired,debug): #combine coordinates for paired reads
        outfile = open(merged_paired,'w')
        with open(paired_file,'r') as infile:
                for line in infile:
                        line = line.rstrip()
                        line_split = line.split("\t")
                        chrom = line_split[0]
                        R1_start = line_split[1]
                        R1_end = line_split[2]
                        R1_score = line_split[4]
                        R2_score = line_split[14]
                        R2_start = line_split[11]
                        R2_end = line_split[12]
                        new_read = "paired"
                        R1_proper = line_split[7]
                        R2_proper = line_split[17]
                        R1_uniq = line_split[9]
                        R2_uniq = line_split[19]
                        new_start = str(min(int(R1_start), int(R2_start)))
                        new_end = str(max(int(R1_end),int(R2_end)))
                        read_ID = line_split[3]
                        TE_ID_1 = line_split[6]
                        TE_ID_2 = line_split[16]
                        if TE_ID_1 != TE_ID_2:
                                new_TE_ID = TE_ID_1 + "&" + TE_ID_2
                                new_score=R1_score + "&" + R2_score
                        else:
                                new_TE_ID = TE_ID_1
                                new_score = R1_score
                        strand = line_split[5]
                        new_uniq="R1" + "_" + R1_start + "_" + R1_uniq + ":" + "R2" + "_" +
R2_start + "_" + R2_uniq
                        new_proper = "R1" + "_" + R1_proper + ":" + "R2" + "_" + R2_proper
                        insert_size=abs(int(new_end) - int(new_start))
                        new_line = "\t".join([chrom,
new_start,new_end,read_ID,new_score,strand,new_TE_ID,new_proper,new_read,new_uniq])
                        outfile.writelines(new_line + "\n")
        outfile.close()
        infile.close()
        if not debug:
                os.unlink(paired_file)


    def find_proper(single_bed,nonproper_bed, proper_bed,debug):
```

```
        #separate read 1 and read2 into separate files
        awkcommand_list = ["awk","'$8 ~ v'","v='nonproper'", single_bed,">", nonproper_bed]
        awkcommand = " ".join(awkcommand_list)
        sp.check_call(["/bin/sh", "-c", awkcommand])
        awkcommand_list = ["awk", """FNR==NR{a[$0]++;next}!a[$0]{print $0}""",
nonproper_bed, single_bed, ">", proper_bed] #writes lines that are in single_bed and not in
nonproper bed (all proper alignments)
        awkcommand = " ".join(awkcommand_list)
        sp.check_call(["/bin/sh","-c",awkcommand])
        if not debug:
                os.unlink(single_bed)

    def remove_repeat_reads(paired_bed,unpaired_bed,new_unpaired_bed, debug): #removes reads
from unpaired files that are already present in paired files
        removecommandlist = ["awk", "-v", "OFS='\\t'","'FNR==NR{a[$4]++;next}!a[$4]{print
$0}'",paired_bed, unpaired_bed, ">", new_unpaired_bed]
        removecommand = " ".join(removecommandlist)
        sp.check_call(["/bin/sh","-c",removecommand])
        if not debug:
                os.unlink(unpaired_bed)

    def find_paired_uniq(multi_tempfile,paired_uniq_tempfile,new_multi_tempfile,
new_uniq_tempfile,debug):
        ##### Find reads which exclusively align as paired + uniq -> means one read is unique and
the other is multi-aligned to nearby TEs
        paired_uniq_tempfile_1 = paired_uniq_tempfile + "_1"
        new_multi_tempfile_1 = new_multi_tempfile + "_0"
        awkcommand_list = ["awk", """$9=="paired" && $10 ~/uniq/""", multi_tempfile,  ">",
paired_uniq_tempfile_1] #writes lines labeled with "paired" and uniq
        awkcommand = " ".join(awkcommand_list)
        sp.check_call(["/bin/sh","-c",awkcommand])
        awkcommand_list = ["awk", """FNR==NR{a[$0]++;next}!a[$0]{print $0}""",
paired_uniq_tempfile_1, multi_tempfile, ">", new_multi_tempfile_1] #writes lines that are not paired
and uniq
        awkcommand = " ".join(awkcommand_list)
        sp.check_call(["/bin/sh","-c",awkcommand])
        awkcommand_list = ["awk", """FNR==NR{a[$4]++;next}!a[$4]{print $0}""",
new_multi_tempfile_1, paired_uniq_tempfile_1,  ">", paired_uniq_tempfile] #writes lines in read is
in unique2 and multi file -> gets first appearance of multi-aligned reads
        awkcommand = " ".join(awkcommand_list)
        sp.check_call(["/bin/sh","-c",awkcommand])
        awkcommand_list = ["awk", """FNR==NR{a[$0]++;next}!a[$0]{print $0}""",
paired_uniq_tempfile, multi_tempfile, ">", new_multi_tempfile] #writes lines that are not paired and
uniq
        awkcommand = " ".join(awkcommand_list)
        sp.check_call(["/bin/sh","-c",awkcommand])
        new_multi_bed = open(new_multi_tempfile, 'a')
        new_uniq_bed = open(new_uniq_tempfile, 'a')

        def eval_paired_uniq(startlist,stoplist,beddict,uniqfile,multifile):
                if len(beddict)==1: #if all alignments of read are to same TE_ID
```

```
                        TE_ID=beddict.keys()[0]
                        newbedline = beddict[TE_ID]
                        newbedline.Read_geno_start =str(max(start_list))
                        newbedline.Read_geno_stop = str(min(stop_list))
                        newbedline.write_bedline(uniqfile)
                else:
                        for TE_ID, newbedline in beddict.iteritems():
                                newbedline.write_bedline(multifile)
        paired_uniq_bed = open(paired_uniq_tempfile,'r')
        paired_uniq_bed.seek(0)
        prev_ID = False
        bed_dict={}
        start_list=[]
        stop_list=[]
        for line in paired_uniq_bed:
                bed_line = bedline(line)
                if not prev_ID: #if first line
                        prev_ID = bed_line.Read_ID
                        bed_dict={bed_line.TE_ID:bed_line}
                        start_list = [int(bed_line.Read_geno_start)]
                        stop_list = [int(bed_line.Read_geno_stop)]
                elif prev_ID == bed_line.Read_ID:
                        bed_dict[bed_line.TE_ID] = bed_line
                        start_list.append(int(bed_line.Read_geno_start))
                        stop_list.append(int(bed_line.Read_geno_stop))
                else: #if new read, evaluate previous read and replace info
                        eval_paired_uniq(start_list,stop_list,bed_dict,new_uniq_bed,new_multi_bed)
                        #start new ID
                        prev_ID = bed_line.Read_ID
                        bed_dict={bed_line.TE_ID:bed_line}
                        start_list = [int(bed_line.Read_geno_start)]
                        stop_list = [int(bed_line.Read_geno_stop)]
        #End of loop
        eval_paired_uniq(start_list,stop_list,bed_dict,new_uniq_bed,new_multi_bed)
        if not debug: #delete unneeded tempfiles
                os.unlink(paired_uniq_tempfile_1)
                os.unlink(new_multi_tempfile_1)
                os.unlink(multi_tempfile)
                os.unlink(paired_uniq_tempfile)

def get_subF(TE_ID):
        TE_ID_components = TE_ID.split("|")
        TE_ID_subfamily = TE_ID_components[3]
        return TE_ID_subfamily

def get_strand(TE_ID):
        TE_ID_components = TE_ID.split("|")
        TE_ID_strand = TE_ID_components[5]
        return TE_ID_strand

def get_chr(TE_ID):
```

```
    TE_ID_components = TE_ID.split("|")
    TE_ID_strand = TE_ID_components[0]
    return TE_ID_strand

def split_subF(subF):
    if "overlap" in subF:
            subfamily_string=subF.replace("overlap:","") #remove overlap
            subfamily_string=subfamily_string.replace(".","") #remove period
            subfamily_string=subfamily_string.replace("+","") #remove strand
            subfamily_string=subfamily_string.replace("-","") #remove strand
            subfamily_list = subfamily_string.split(",")
            subfamily_list = set(subfamily_list)
            return subfamily_list
    else:
            return [subF]

def get_length(TE_ID):
    TE_ID_components = TE_ID.split("|")
    TE_ID_stop = TE_ID_components[2]
    TE_ID_start = TE_ID_components[1]
    TE_length = int(TE_ID_stop) - int(TE_ID_start)
    return TE_length

#### DEFINE CLASS TO CALCULATE COUNTS ###############

class RepCalc(object):
    def __init__(self, TE_ID):
            self.TE_ID = TE_ID
            self.uniqcounts = 0
            self.uniq_plus = 0
            self.uniq_minus = 0
            self.multilist = []
            self.multi_plus=0
            self.multi_minus = 0
            self.multi_u_plus=0
            self.multi_u_minus = 0
            self.multimax_plus=0
            self.multimax_minus = 0
            self.counts_plus = 0
            self.counts_minus = 0
            self.multi_counts = 0
            self.counts_tot = 0
            self.total_reads_tot = 0
            self.uniq_reads_plus = 0
            self.uniq_reads_minus = 0
            self.total_1_plus = 0
            self.total_1_minus = 0
            self.multi_u_reads_plus = 0
            self.multi_u_reads_minus = 0
            self.multi_reads_plus = 0
            self.multi_reads_minus = 0
```

```
        self.multi_max_reads_plus = 0
        self.multi_max_reads_minus = 0
        self.total_reads_plus = 0
        self.total_reads_minus = 0
        self.uniq_fragment_plus = 0
        self.uniq_fragment_minus = 0
        self.uniq_plus_perTagkb = 0
        self.uniq_minus_perTagkb = 0
        self.start_plus=False
        self.stop_plus=False
        self.start_minus=False
        self.stop_minus=False
        self.start_tot=False
        self.stop_tot=False
        self.efflength_plus=0
        self.efflength_minus=0
        self.length_plus=0
        self.length_minus=0
        self.length_tot=0
        self.fpkm_tot=0
        self.uniq_starts_plus=set()
        self.uniq_starts_minus=set()
        self.TEstrand = get_strand(TE_ID)
        self.read_list_plus = []
        self.read_list_minus = []
        self.total_fragment_plus = 0
        self.total_fragment_minus = 0
        self.min_fragment_plus = False
        self.min_fragment_minus = False
        self.new_counts_plus = 0
        self.new_counts_minus = 0
        self.fpkm=0
        # self.same_strandcount=0
        # self.opp_strandcount = 0
def add_uniq(self,strand,count):
                if strand == "+":
                        self.uniq_plus += count
                        # self.uniq_reads_plus += count

                elif strand == "-":
                        self.uniq_minus += count
                        # self.uniq_reads_minus += count
def add_multi(self,strand,read_fraction):
        count = read_fraction
        readcount=1
        if strand == "+":
                self.multi_plus += count
                # self.multi_reads_plus += readcount
        else:
                self.multi_minus += count
                # self.multi_reads_minus += readcount
```

```
	def add_multi_u(self,strand,read_fraction):
		count = read_fraction
		readcount=1
		if strand == "+":
			self.multi_u_plus += count
			# self.multi_u_reads_plus += readcount
		else:
			self.multi_u_minus += count
			# self.multi_u_reads_minus += readcount
	def add_read(self,read_ID,strand):
		if strand == "+":
			self.read_list_plus.append(read_ID)
		elif strand == "-":
			self.read_list_minus.append(read_ID)
	def get_fragment(self,start,stop, strand):
		start=int(start)
		stop = int(stop)
		length = stop - start
		if strand == "+":
			self.total_fragment_plus += length
			if not self.start_plus or start < self.start_plus:
				self.start_plus = start
			if not self.stop_plus or stop > self.stop_plus:
				self.stop_plus = stop
			if not self.min_fragment_plus:
				self.min_fragment_plus = length
			else:
				self.min_fragment_plus = min(length,self.min_fragment_plus)
			self.total_reads_plus +=1
			self.avg_fraglength_plus  = self.total_fragment_plus/self.total_reads_plus
		else:
			self.total_fragment_minus += length
			if not self.start_minus or start < self.start_minus:
				self.start_minus = start
			if not self.stop_minus or stop > self.stop_minus:
				self.stop_minus = stop
			if not self.min_fragment_minus:
				self.min_fragment_minus = length
			else:
				self.min_fragment_minus = min(length,self.min_fragment_minus)
			self.total_reads_minus +=1
			self.avg_fraglength_minus =
self.total_fragment_minus/self.total_reads_minus
		if self.start_plus and self.start_minus: #minimum comparing False and a number
gives False
			self.start_tot = min(self.start_plus, self.start_minus)
		elif self.start_plus:
			self.start_tot = self.start_plus
		elif self.start_minus:
			self.start_tot = self.start_minus
```

```
                if self.stop_plus and self.stop_minus: #minimum comparing False and a number
gives False
                        self.stop_tot = max(self.stop_plus, self.stop_minus)
                        self.avg_fraglength_tot = (self.avg_fraglength_plus +
self.avg_fraglength_minus)/2
                elif self.stop_plus:
                        self.stop_tot = self.stop_plus
                        self.avg_fraglength_tot = self.avg_fraglength_plus
                elif self.stop_minus:
                        self.stop_tot = self.stop_minus
                        self.avg_fraglength_tot = self.avg_fraglength_minus


        def get_uniqfragment(self,start,stop, strand, read_end, read_uniq):
                start=int(start)
                stop = int(stop)
                if read_end != "paired":
                        if strand == "+":
                                self.uniq_fragment_plus += 1
                                if start not in self.uniq_starts_plus:
                                        self.uniq_starts_plus.add(start)
                        else:
                                self.uniq_fragment_minus += 1   #add fragment length to count
                                if start not in self.uniq_starts_minus:
                                                self.uniq_starts_minus.add(start)
                else: #if reads are paired
                        uniq_list = read_uniq.split(":")
                        R1_uniq_list = uniq_list[0].split("_")
                        R2_uniq_list = uniq_list[1].split("_")
                        R1_start = R1_uniq_list[1]
                        R2_start = R2_uniq_list[1]
                        R1_uniq=R1_uniq_list[2]
                        R2_uniq=R2_uniq_list[2]
                        # Effective uniquely alignable length
                        if R1_uniq == "uniq":
                                if strand == "+":
                                        self.uniq_fragment_plus += 1   #add to fragment count
                                        self.uniq_starts_plus.add(R1_start)
                                if strand == "-":
                                        self.uniq_fragment_minus += 1  #add to fragment count
                                        self.uniq_starts_minus.add(R1_start)
                        if R2_uniq == "uniq":
                                if strand == "+":
                                        self.uniq_fragment_plus += 1   #add to fragment count
                                        self.uniq_starts_plus.add(R2_start)
                                if strand == "-":
                                        self.uniq_fragment_minus += 1  #add to fragment count
                                        self.uniq_starts_minus.add(R2_start)
                ### Transcript start stop
                if strand == "+":
                        if not self.start_plus or start < self.start_plus:
```

```
                                        self.start_plus = start
                                if not self.stop_plus or stop > self.stop_plus:
                                        self.stop_plus = stop
                        else:
                                if not self.start_minus or start < self.start_minus:
                                        self.start_minus = start
                                if not self.stop_minus or stop > self.stop_minus:
                                        self.stop_minus = stop
                        if self.start_plus and self.start_minus: #minimum comparing False and a number
gives False
                                self.start_tot = min(self.start_plus, self.start_minus)
                        elif self.start_plus:
                                self.start_tot = self.start_plus
                        elif self.start_minus:
                                self.start_tot = self.start_minus
                        self.stop_tot = max(self.stop_plus, self.stop_minus)
                def calcuniqRep(self):
                        self.efflength_plus = len(self.uniq_starts_plus)
                        self.efflength_minus = len(self.uniq_starts_minus)
                        if self.efflength_plus:
                                self.uniq_plus_perTagkb =
self.uniq_fragment_plus/(int(self.efflength_plus)/1000)
                        if self.efflength_minus:
                                self.uniq_minus_perTagkb =
self.uniq_fragment_minus/(int(self.efflength_minus)/1000)
                        self.uniqcounts=self.uniq_plus + self.uniq_minus
                def calcmultiRep(self,iteration):
                        self.length_plus = self.stop_plus - self.start_plus
                        self.length_minus = self.stop_minus - self.start_minus
                        self.length_tot = self.stop_tot - self.start_tot
                        if iteration ==1:
                                self.total_1_plus = self.uniq_plus + self.multi_plus + self.multi_u_plus
                                self.total_1_minus = self.uniq_minus + self.multi_minus +
self.multi_u_minus
                                self.counts_tot = self.total_1_plus + self.total_1_minus
                        tot_change = 0
                        changed_strands =0
                        pct_change = 0
                        ###old  likelihood
                        if self.multi_plus:
                                avg_fraglength  = self.avg_fraglength_plus
                                min_fraglength = self.min_fragment_plus
                                if iteration > 1:
                                        self.old_counts_plus = self.uniq_plus + self.multi_plus +
self.multi_u_plus
                                        self.oldmulti_plus_perkb = self.old_counts_plus/((self.length_plus -
avg_fraglength +1)/1000)
                                        if self.oldmulti_plus_perkb <0:
                                                self.oldmulti_plus_perkb =
self.old_counts_plus/((self.length_plus - min_fraglength +1)/1000)
                                        self.multi_plus = self.multimax_plus
```

```
                                        self.new_counts_plus = self.uniq_plus + self.multimax_plus +
self.multi_u_plus
                                        if self.new_counts_plus < 1:
                                                self.newmulti_plus_perkb = 0
                                        else:
                                                self.newmulti_plus_perkb =
self.new_counts_plus/((self.length_plus - avg_fraglength +1)/1000)
                                                if self.newmulti_plus_perkb  < 0:
                                                        self.newmulti_plus_perkb =
self.new_counts_plus/((self.length_plus - min_fraglength +1)/1000)
                                        tot_change += abs(self.old_counts_plus - self.new_counts_plus)
                                        changed_strands +=1
                                        pct_change = tot_change/self.old_counts_plus *100
                                else:
                                        self.oldmulti_plus_perkb = self.uniq_plus_perTagkb
                                        self.old_counts_plus = self.uniq_plus + self.multi_plus +
self.multi_u_plus
                                        if self.old_counts_plus < 1:
                                                self.newmulti_plus_perkb = 0
                                        else:
                                                self.newmulti_plus_perkb =
self.old_counts_plus/((self.length_plus - avg_fraglength +1)/1000)
                                                if self.newmulti_plus_perkb  < 0:
                                                        self.newmulti_plus_perkb =
self.old_counts_plus/((self.length_plus - min_fraglength +1)/1000)
                        if self.multi_minus:
                                avg_fraglength = self.avg_fraglength_minus
                                min_fraglength = self.min_fragment_minus
                                if iteration > 1:
                                        self.old_counts_minus =self.uniq_minus + self.multi_minus +
self.multi_u_minus
                                        self.oldmulti_minus_perkb =
self.old_counts_minus/((self.length_minus - avg_fraglength +1)/1000)
                                        if self.oldmulti_minus_perkb <0:
                                                self.oldmulti_minus_perkb =
self.old_counts_minus/((self.length_minus - min_fraglength +1)/1000)
                                        self.multi_minus = self.multimax_minus
                                        self.new_counts_minus = self.uniq_minus + self.multimax_minus +
self.multi_u_minus
                                        if self.new_counts_minus < 1:
                                                self.newmulti_minus_perkb = 0
                                        else:
                                                self.newmulti_minus_perkb =
self.new_counts_minus/((self.length_minus - avg_fraglength +1)/1000)
                                                if self.newmulti_minus_perkb  < 0:
                                                        self.newmulti_minus_perkb =
self.new_counts_minus/((self.length_minus - min_fraglength +1)/1000)
                                        tot_change +=abs(self.old_counts_minus - self.new_counts_minus)
                                        pct_change = tot_change/self.old_counts_minus*100
                                        changed_strands +=1
                                else:
```

```
                                        self.oldmulti_minus_perkb = self.uniq_minus_perTagkb
                                        self.old_counts_minus = self.uniq_minus + self.multi_minus +
self.multi_u_minus
                                        if self.old_counts_minus < 1:
                                                self.newmulti_minus_perkb = 0
                                        else:
                                                self.newmulti_minus_perkb =
self.old_counts_minus/((self.length_minus - avg_fraglength +1)/1000)
                                                if self.newmulti_minus_perkb  < 0:
                                                        self.newmulti_minus_perkb =
self.old_counts_minus/((self.length_minus - min_fraglength +1)/1000)
                        #reset multimax
                        self.multimax_plus = 0
                        self.multimax_minus = 0
                        if iteration > 1:
                                self.counts_tot = self.new_counts_plus + self.new_counts_minus
                        if changed_strands > 0:
                                return (tot_change/changed_strands)
                        else:
                                return 0

                def add_multimax(self,strand,read_fraction):
                        count = read_fraction
                        if strand == "+":
                                self.multimax_plus += count
                                # self.multi_max_reads_plus +=1
                        else:
                                self.multimax_minus += count
                                # self.multi_max_reads_minus +=1
                def calc_transcript_coords(self,read_locdict):
                        for read in self.read_list_plus:
                                start=int(read_locdict[read][(self.TE_ID,"+")][1])
                                stop = int(read_locdict[read][(self.TE_ID,"+")][2])
                                if not self.start_plus or start < self.start_plus:
                                        self.start_plus = start
                                if not self.stop_plus or stop > self.stop_plus:
                                        self.stop_plus = stop
                        for read in self.read_list_minus:
                                start=int(read_locdict[read][(self.TE_ID,"-")][1])
                                stop = int(read_locdict[read][(self.TE_ID,"-")][2])
                                if not self.start_minus or start < self.start_minus:
                                        self.start_minus = start
                                if not self.stop_minus or stop > self.stop_minus:
                                        self.stop_minus = stop
                        if self.start_plus and self.start_minus: #minimum comparing False and a number
gives False
                                self.start_tot = min(self.start_plus, self.start_minus)
                        elif self.start_plus:
                                self.start_tot = self.start_plus
                        elif self.start_minus:
                                self.start_tot = self.start_minus
```

```
                if self.stop_plus and self.stop_minus: #minimum comparing False and a number
gives False
                        self.stop_tot = max(self.stop_plus, self.stop_minus)
                elif self.stop_plus:
                        self.stop_tot = self.stop_plus
                elif self.stop_minus:
                        self.stop_tot = self.stop_minus
        def calc_total_reads(self):
                self.total_reads_plus = len(self.read_list_plus)
                self.total_reads_minus = len(self.read_list_minus)
        def writeRep(self,aligned_libsize, counts_file,basename,strandedness,iteration):
                if iteration ==0:
                        self.total_1_plus = self.uniq_plus + self.multi_plus + self.multi_u_plus
                        self.total_1_minus = self.uniq_minus + self.multi_minus +
self.multi_u_minus
                self.length_plus = self.stop_plus - self.start_plus
                self.length_minus = self.stop_minus - self.start_minus
                self.length_tot = self.stop_tot - self.start_tot
                self.counts_plus = self.uniq_plus + self.multi_plus + self.multi_u_plus
                self.counts_minus =self.uniq_minus + self.multi_minus + self.multi_u_minus
                self.total_1 = self.total_1_plus + self.total_1_minus
                self.uniq_tot = self.uniq_plus + self.uniq_minus
                self.counts_tot = self.counts_plus + self.counts_minus
                self.total_reads_tot = self.total_reads_plus + self.total_reads_minus
                self.TE_ID_tab = self.TE_ID.split("|")
                self.TE_chr =self.TE_ID_tab[0]
                if strandedness> 0:
                        if self.counts_plus > 0 and self.total_reads_plus > 0 and self.length_plus > 0:
                                outline = squire_bed(self.TE_chr,self.start_plus,self.stop_plus,
self.avg_fraglength_plus,"+",self.TE_ID,self.uniq_plus,self.counts_plus,self.total_reads_plus,basena
me,aligned_libsize)
                                counts_file.writelines(outline.out_line + "\n")
                                self.fpkm += outline.fpkm

                        if self.counts_minus > 0 and self.total_reads_minus > 0 and
self.length_minus > 0:
                                outline = squire_bed(self.TE_chr,self.start_minus,self.stop_minus,
self.avg_fraglength_minus,"-
",self.TE_ID,self.uniq_minus,self.counts_minus,self.total_reads_minus,basename,aligned_libsize)
                                counts_file.writelines(outline.out_line + "\n")
                                self.fpkm += outline.fpkm

                else:
                        if self.counts_tot > 0 and self.total_reads_tot > 0 and self.length_tot > 0:
                                outline =
squire_bed(self.TE_chr,self.start_tot,self.stop_tot,self.avg_fraglength_tot,
".",self.TE_ID,self.uniq_tot, self.counts_tot,self.total_reads_tot,basename,aligned_libsize)
                                counts_file.writelines(outline.out_line + "\n")
                                self.fpkm += outline.fpkm
```

```
    class squire_bed(object):
        def __init__(self,chrom, start, stop, avg_fraglength,strand, TE_ID,uniq_counts,total_counts,
reads, basename, aligned_libsize):
                self.seqname = chrom
                self.source = "SQuIRE"
                self.feature = "TE"
                self.start = str(start)
                self.end = str(stop)
                self.length = stop - start
                self.score = TE_ID.split("|")[4]
                self.bed = TE_ID.split("|")
                self.strand = strand
                self.uniq_counts = str(uniq_counts)
                self.total_counts = "{0:.2f}".format(total_counts)
                self.reads = str(reads)
                self.conf =  "{0:.2f}".format(total_counts/reads * 100)
                self.fpkm =  (total_counts/((self.length /1000)*(int(aligned_libsize)/1000000)))
                self.aligned_libsize = str(aligned_libsize)
                self.chrom = self.bed[0]
                self.out_list = [self.chrom, self.start, self.end,
TE_ID,"{0:.2f}".format(self.fpkm),strand, basename, self.aligned_libsize] + self.bed +
[self.uniq_counts,self.total_counts, self.reads, self.conf ]
                self.out_line = "\t".join(self.out_list)

    class subfamily(object):
        def __init__(self,subF,multi_reads):
                self.subF = subF
                self.uniq = 0
                self.unique_copies = 0
                self.total_counts_pre = 0
                self.total_counts = 0
                self.total_reads = 0
                self.multi_reads=multi_reads
                self.fpkm=0
                # self.same=0
                # self.opp = 0
        def add_TE_count(self,RepClass,strandedness):
                self.uniq +=RepClass.uniqcounts
                self.total_counts+=RepClass.counts_tot
                self.total_counts_pre +=RepClass.total_1
                self.fpkm += RepClass.fpkm
                # if strandedness > 0:
                #       self.same +=RepClass.same_strandcount
                #       self.opp += RepClass.opp_strandcount
                # else:
                #       self.same= "NA"
                #       self.opp="NA"
        def add_copy_info(self,line):
                self.line_copies = line[1]
                self.line_length = line[2]
```

```
        def write_subfamily(self,outfile,basename,aligned_libsize,iteration):
                self.total_reads = self.multi_reads + self.uniq
                if self.total_reads >  0:
                        self.conf = "{0:.2f}".format(self.total_counts/self.total_reads * 100)
#confidence = total counts assigned to subfamily divided by total reads
                else:
                        self.conf = "NA"
                outfile.writelines(basename + "\t" + str(aligned_libsize) + "\t" + self.subF + "\t" +
self.line_copies + "\t" + str(self.fpkm)  + "\t" + str(self.uniq)  + "\t" +
"{0:.2f}".format(self.total_counts) + "\t" + str(self.total_reads) + "\t" + str(self.conf) + "\n")




    class bedline(object):
        def __init__(self,line):
                self.line = line.rstrip() #removes white space at end of line
                self.line_split = self.line.split('\t')  # returns list of items that were separated by tab in
original file
                ### Read variables #####
                col_no = len(self.line_split)
                self.Read_chr = self.line_split[0]
                self.Read_geno_start = self.line_split[1]
                self.Read_geno_stop = self.line_split[2]
                self.Read_name = self.line_split[3]
                self.Read_ID = re.split("[ #/]", self.Read_name)
                self.Read_ID=self.Read_ID[0]
                self.Read_score = self.line_split[4]
                self.Read_strand = self.line_split[5]
                if col_no >=7:
                        self.TE_ID = self.line_split[6]
                        self.TE_ID_list = self.TE_ID.split('&')
                else:
                        self.TE_ID = False
                if col_no >=8:
                        self.proper = self.line_split[7]
                else:
                        self.proper = False
                if col_no >=9:
                        self.Read_end = self.line_split[8]
                else:
                        self.Read_end = False
                if col_no >=10:
                        self.uniq = self.line_split[9]
                else: self.uniq = False
                        #### TE variables #####
                self.Read_length = int(self.Read_geno_stop) - int(self.Read_geno_start)


        def write_bedline(self,outfile):
```

```
                outline = [self.Read_chr, self.Read_geno_start, self.Read_geno_stop,
self.Read_name, self.Read_score,self.Read_strand]
                if self.TE_ID:
                        outline.append(self.TE_ID)
                if self.proper:
                        outline.append(self.proper)
                if self.Read_end:
                        outline.append(self.Read_end)
                if self.uniq:
                        outline.append(self.uniq)
                outline = "\t".join(outline)
                outfile.writelines(outline + "\n")

    def uniquecount(tempBED,RepCalc_dict,read_locdict):
        unique_fragsum=0
        tempBED.seek(0)
        unique_linecount=0
        for line in tempBED:
                bed_line = bedline(line)
                if bed_line.Read_length < 25:
                        continue
                unique_fragsum += bed_line.Read_length
                unique_linecount+=1
                if len(bed_line.TE_ID_list) == 1:
                ### Convert Read coordinates from Rep chrom coordinates to genomic coordinates
##
                #if RNAseq data was aligned to whole genome, Read_start-Seq_start=0, so the result
will be the same
                        TE_id_list = bed_line.TE_ID.split('|')
                        TE_start = TE_id_list[1]
                        TE_stop = TE_id_list[2]
                        if bed_line.TE_ID not in RepCalc_dict:
                                RepCalc_dict[bed_line.TE_ID] = RepCalc(bed_line.TE_ID)
                        if "uniq" in bed_line.uniq:
                                RepCalc_dict[bed_line.TE_ID].add_uniq(bed_line.Read_strand,1)

        RepCalc_dict[bed_line.TE_ID].get_uniqfragment(bed_line.Read_geno_start,bed_line.Read_
geno_stop,bed_line.Read_strand,bed_line.Read_end, bed_line.uniq)

        RepCalc_dict[bed_line.TE_ID].get_fragment(bed_line.Read_geno_start,bed_line.Read_geno
_stop,bed_line.Read_strand)

        RepCalc_dict[bed_line.TE_ID].add_read(bed_line.Read_ID,bed_line.Read_strand)
                        else:

        RepCalc_dict[bed_line.TE_ID].add_multi_u(bed_line.Read_strand,1)

        RepCalc_dict[bed_line.TE_ID].get_fragment(bed_line.Read_geno_start,bed_line.Read_geno
_stop,bed_line.Read_strand)

        RepCalc_dict[bed_line.TE_ID].add_read(bed_line.Read_ID,bed_line.Read_strand)
```

```
                else: #if two TE_IDs
                        for TE_ID in bed_line.TE_ID_list:
                                TE_id_list = TE_ID.split('|')
                                TE_start = TE_id_list[1]
                                TE_stop = TE_id_list[2]
                                if TE_ID not in RepCalc_dict:
                                        RepCalc_dict[TE_ID] = RepCalc(TE_ID)
                                if "uniq" in bed_line.uniq: #if read is unique when aligning single
end, otherwise only uniquely aligned because paired
                                        RepCalc_dict[TE_ID].add_uniq(bed_line.Read_strand,1)

        RepCalc_dict[TE_ID].get_uniqfragment(bed_line.Read_geno_start,bed_line.Read_geno_stop
,bed_line.Read_strand,bed_line.Read_end, bed_line.uniq)

        RepCalc_dict[TE_ID].get_fragment(bed_line.Read_geno_start,bed_line.Read_geno_stop,bed
_line.Read_strand)

        RepCalc_dict[TE_ID].add_read(bed_line.Read_ID,bed_line.Read_strand)
                                else:
                                        RepCalc_dict[TE_ID].add_multi_u(bed_line.Read_strand,1)

        RepCalc_dict[TE_ID].get_fragment(bed_line.Read_geno_start,bed_line.Read_geno_stop,bed
_line.Read_strand)

        RepCalc_dict[TE_ID].add_read(bed_line.Read_ID,bed_line.Read_strand)
        unique_fragavg=unique_fragsum/int(unique_linecount)
        return unique_fragavg

    def multicount(tempBED,RepCalc_dict, multidict,read_locdict):
        tempBED.seek(0)
        for line in tempBED:
                adj = False
                bed_line = bedline(line)
                if bed_line.Read_length < 25:
                        continue
                if len(bed_line.TE_ID_list) == 1: #if read not aligned to more than one TE_ID at
same position
                        if bed_line.TE_ID not in RepCalc_dict:
                                RepCalc_dict[bed_line.TE_ID] = RepCalc(bed_line.TE_ID) #Initiate
Repeat class object, Add RepeatTagNo, Repeat Total Tag Length to Repeat class object
                        if bed_line.Read_ID not in multidict: #if TE_ID not in multi dictionary:

        multidict[bed_line.Read_ID]={bed_line.TE_ID:bed_line.Read_strand} #For each Read_ID,
include TE it's aligned to and strand it's on
                                # locdict[bed_line.Read_ID] =
{bed_line.TE_ID:[bed_line.Read_chr,str(bed_line.Read_geno_start),str(bed_line.Read_geno_stop),ad
j]} #For each Read_Id, include location of alignment

        RepCalc_dict[bed_line.TE_ID].get_fragment(bed_line.Read_geno_start,bed_line.Read_geno
_stop,bed_line.Read_strand)
```

```
        RepCalc_dict[bed_line.TE_ID].add_read(bed_line.Read_ID,bed_line.Read_strand)
                        else: #if TE_ID in dictionary:
                                multidict[bed_line.Read_ID][bed_line.TE_ID] =
bed_line.Read_strand
                                # locdict[bed_line.Read_ID][bed_line.TE_ID] =
[bed_line.Read_chr,str(bed_line.Read_geno_start),str(bed_line.Read_geno_stop),adj]

        RepCalc_dict[bed_line.TE_ID].get_fragment(bed_line.Read_geno_start,bed_line.Read_geno
_stop,bed_line.Read_strand)

        RepCalc_dict[bed_line.TE_ID].add_read(bed_line.Read_ID,bed_line.Read_strand)
                else: ##if read aligned to more than one TE_ID at same position
                        adj= True
                        for TE_ID in bed_line.TE_ID_list:
                                TE_id_list = TE_ID.split('|')
                                TE_start = TE_id_list[1]
                                TE_stop = TE_id_list[2]

                                if TE_ID not in RepCalc_dict:
                                        RepCalc_dict[TE_ID] = RepCalc(TE_ID)
                                if bed_line.Read_ID not in multidict: #if TE_ID not in dictionary:

        multidict[bed_line.Read_ID]={TE_ID:bed_line.Read_strand} #Initiate Repeat class object,
Add RepeatTagNo, Repeat Total Tag Length to Repeat class object
                                        # locdict[bed_line.Read_ID] =
{TE_ID:[bed_line.Read_chr,str(bed_line.Read_geno_start),str(bed_line.Read_geno_stop),adj]}

        RepCalc_dict[TE_ID].get_fragment(bed_line.Read_geno_start,bed_line.Read_geno_stop,bed
_line.Read_strand)

        RepCalc_dict[TE_ID].add_read(bed_line.Read_ID,bed_line.Read_strand)
                        else: #if TE_ID in dictionary:
                                multidict[bed_line.Read_ID][TE_ID] =
bed_line.Read_strand
                                # locdict[bed_line.Read_ID][TE_ID] =
[bed_line.Read_chr,str(bed_line.Read_geno_start),str(bed_line.Read_geno_stop),adj]

        RepCalc_dict[TE_ID].add_read(bed_line.Read_ID,bed_line.Read_strand)

        RepCalc_dict[TE_ID].get_fragment(bed_line.Read_geno_start,bed_line.Read_geno_stop,bed
_line.Read_strand)


    def comparedict(read_multidict, RepCalc_dict):
        for Read_ID,TE_dict in read_multidict.iteritems():
                setfraction(Read_ID,TE_dict,RepCalc_dict)


    def setfraction(Read_ID,TE_dict,RepCalc_dict): #compare multi with unique
        ID_TagKb_dict = {}
```

```
	UnTagged_TEs = 0
	read_subF = []
	read_sum=0
	### Use count from  unique count based on strand
	for TE_ID,strand in TE_dict.iteritems():
		#adj=locdict[TE_ID][3]
		#if TE_ID is untagged, would not have any unique reads
		if strand == "+":
			if RepCalc_dict[TE_ID].uniq_plus_perTagkb==0:
				read_fraction = 1/len(TE_dict)
				RepCalc_dict[TE_ID].add_multi(strand,read_fraction)

	#RepCalc_dict[TE_ID].get_fragment(locdict[TE_ID][1],locdict[TE_ID][2],strand)
				UnTagged_TEs +=1
				read_subF.append(get_subF(TE_ID))
				read_sum+=read_fraction
			elif RepCalc_dict[TE_ID].uniq_plus_perTagkb:
		#if TE is tagged, evaluate likelihood of contribution by length of uniq Tags on
appropriate strand per Tagkb
				ID_TagKb_dict[TE_ID] =
RepCalc_dict[TE_ID].uniq_plus_perTagkb
		if strand == "-": #if TE_ID is untagged, would not have any unique reads
			if RepCalc_dict[TE_ID].uniq_minus_perTagkb==0:
				read_fraction = 1/len(TE_dict)
				read_sum+=read_fraction
				RepCalc_dict[TE_ID].add_multi(strand,read_fraction)

	#RepCalc_dict[TE_ID].get_fragment(locdict[TE_ID][1],locdict[TE_ID][2],strand)
				UnTagged_TEs +=1
				read_subF.append(get_subF(TE_ID))
			elif RepCalc_dict[TE_ID].uniq_minus_perTagkb:
		#if TE is tagged, evaluate likelihood of contribution by length of uniq Tags on
appropriate strand per Tagkb
				ID_TagKb_dict[TE_ID] =
RepCalc_dict[TE_ID].uniq_minus_perTagkb
	TagKb_sum = sum(itervalues(ID_TagKb_dict))
	#print("TagKb_sum " + Read_ID + " " + str(TagKb_sum),file = sys.stderr)
	remainder_fraction = (len(TE_dict) - UnTagged_TEs)/len(TE_dict) #fraction of TEs in
TE_dict that are tagged
	if TagKb_sum > 0: #if some unique elements can be assigned
		for TE_ID, uniq_sum in ID_TagKb_dict.iteritems():
			# adj=locdict[TE_ID][3]
			strand = TE_dict[TE_ID]
			#print(TE_ID + " " + str(uniq_sum),file = sys.stderr)
			read_fraction = (uniq_sum/TagKb_sum) * remainder_fraction #defines
read_fraction by TE_IDs contribution to total uniqperTagKb_sum
			read_sum+=read_fraction
			RepCalc_dict[TE_ID].add_multi(strand,read_fraction)
			#
RepCalc_dict[TE_ID].get_fragment(locdict[TE_ID][1],locdict[TE_ID][2],strand)
			read_subF.append(get_subF(TE_ID))
```

```
		### If no unique counts for any element in new dict, read fraction is 1/number of elements per read
		else:
			for TE_ID, uniq_sum in ID_TagKb_dict.iteritems():
				strand = TE_dict[TE_ID]
				#adj=locdict[TE_ID][3]
				read_fraction = 1/len(TE_dict) #defines read_fraction by TE_IDs contribution to total uniqperTagKb_sum
				read_sum+=read_fraction
				RepCalc_dict[TE_ID].add_multi(strand,read_fraction)
				#RepCalc_dict[TE_ID].get_fragment(locdict[TE_ID][1],locdict[TE_ID][2],strand)
				read_subF.append(get_subF(TE_ID))
		unique_subF = set(read_subF) #add 1 read for each uniquely represented subfamily
		if read_sum > 1.01:
			print("read_sum is greater than 1 for " + Read_ID + "\n",file = sys.stderr)
		for subfamily in unique_subF:
			subF_list=split_subF(subfamily)
			for subF in subF_list:
				subF_reads[subF] +=1

	def estdict(read_multidict, RepCalc_dict):
		changed_likelihood_sum = 0
		read_no=0
		changed_reads = 0
		for Read_ID,TE_dict in read_multidict.iteritems():
			changed_likelihood = maxfraction(Read_ID,TE_dict,RepCalc_dict)
			changed_likelihood_sum +=changed_likelihood
			read_no +=1
			if changed_likelihood >= 0.1:
				changed_reads +=1
		if read_no > 0:
			avg_changed_likelihood = changed_likelihood_sum/read_no
		else:
			avg_changed_likelihood = 0
		print("Expectation maximization changed the average likelihood by: " + str(avg_changed_likelihood) + " " + str(datetime.now())  + "\n",file = sys.stderr)
		print("Number of reads with average TE likelihoods changed by at least 0.1: " + str(changed_reads) + " " + str(datetime.now())  + "\n",file = sys.stderr)
		return avg_changed_likelihood

	def maxfraction(Read_ID,TE_dict,RepCalc_dict): #calculate new likelihoods and compare with previous
		ID_TagKb_dict = {}
		UnTagged_TEs = 0
		read_subF = []
		oldTagKb_sum=0
		newTagKb_sum=0
		changed_likelihood=0
		changed_TEs = 0
		read_sum=0
```

```
	TEcount=0
	### Use count from  unique count based on strand
	for TE_ID,strand in TE_dict.iteritems():
		if strand == "+":
			ID_TagKb_dict[(TE_ID,strand)] =
(RepCalc_dict[TE_ID].oldmulti_plus_perkb,RepCalc_dict[TE_ID].newmulti_plus_perkb)
			oldTagKb_sum += RepCalc_dict[TE_ID].oldmulti_plus_perkb
			newTagKb_sum += RepCalc_dict[TE_ID].newmulti_plus_perkb
			if RepCalc_dict[TE_ID].newmulti_plus_perkb > 0:
				TEcount+=1
		elif strand == "-":
			ID_TagKb_dict[(TE_ID,strand)] =
(RepCalc_dict[TE_ID].oldmulti_minus_perkb,RepCalc_dict[TE_ID].newmulti_minus_perkb)
			oldTagKb_sum += RepCalc_dict[TE_ID].oldmulti_minus_perkb
			newTagKb_sum += RepCalc_dict[TE_ID].newmulti_minus_perkb
			if RepCalc_dict[TE_ID].newmulti_minus_perkb > 0:
				TEcount+=1

	for TE_ID_tuple, multi_sum_tuple in ID_TagKb_dict.iteritems():
		TE_ID = TE_ID_tuple[0]
		strand = TE_ID_tuple[1]
		old_multi = multi_sum_tuple[0]
		new_multi = multi_sum_tuple[1]
		 #defines read_fraction by TE_IDs contribution to total multipekb_sum
		if newTagKb_sum == 0:
			newread_fraction = 1/len(TE_dict)
		else:
			newread_fraction = (new_multi/newTagKb_sum)
			if newread_fraction > 1:
				raise Exception("Fraction is greater than 1:" + TE_ID + " " +
Read_ID + " " + str(TEcount) + " " + str(len(TE_dict)) + " " + str(newread_fraction) + " " +
str(new_multi) + " " + str(newTagKb_sum) + '\n')
		read_sum+=newread_fraction
		RepCalc_dict[TE_ID].add_multimax(strand,newread_fraction)
		if oldTagKb_sum == 0:
			oldread_fraction =  1/len(TE_dict)
		else:
			oldread_fraction = (old_multi/oldTagKb_sum)

		changed_likelihood += abs(newread_fraction-oldread_fraction)
		changed_TEs +=1

	if read_sum > 1.01:
		print("read_sum is greater than 1 for " + Read_ID + "\n",file = sys.stderr)
		print("newTagKb_sum is " + str(newTagKb_sum) + "\n",file = sys.stderr)
		print("oldTagKb_sum is " + str(oldTagKb_sum) + "\n",file = sys.stderr)
		print("read_sum is " + str(read_sum) + "\n",file = sys.stderr)
	if changed_TEs == 0:
		avg_change = 0
	else:
		avg_change = changed_likelihood/changed_TEs
```

```
        return avg_change

    def sort_coord(infile, outfile,chrcol,startcol,debug):
        chrfieldsort = "-k" + str(chrcol) + "," + str(chrcol)
        startfieldsort = "-k" + str(startcol) + "," + str(startcol) + "n"
        sort_command_list = ["sort",chrfieldsort,startfieldsort, infile, ">", outfile]
        sort_command = " ".join(sort_command_list)
        sp.check_call(["/bin/sh", "-c", sort_command])
        if not debug:
                os.unlink(infile)

    def sort_counts(tempfile,headerfile,countsfile, field,debug):
        sorted_countsfile = tempfile + ".sorted"
        field_command = str(field) + "," + str(field) + "rn"
        sort_command_list = ["sort","-k",field_command, tempfile, ">", sorted_countsfile]
        sort_command = " ".join(sort_command_list)
        sp.check_call(["/bin/sh", "-c", sort_command])
        catcommand_list = ["cat", headerfile, sorted_countsfile, ">",countsfile ] #combines
multi_aligned reads
        catcommand = " ".join(catcommand_list)
        sp.check_call(["/bin/sh","-c",catcommand])
        if not debug:
                os.unlink(sorted_countsfile)
                os.unlink(tempfile)
                os.unlink(headerfile)

    def bedgraph(infile,strandedness,outfolder,basename):
        if strandedness==1:
                stranded_yesno= "Stranded"
                plus_bedgraph_unique=outfolder + "/" + basename + "Signal.Unique.str2.out.bg"
                minus_bedgraph_unique = outfolder + "/" + basename + "Signal.Unique.str1.out.bg"
                plus_bedgraph_multi=outfolder + "/" + basename +
"Signal.UniqueMultiple.str2.out.bg"
                minus_bedgraph_multi = outfolder + "/" + basename +
"Signal.UniqueMultiple.str1.out.bg"
        elif strandedness==2:
                stranded_yesno= "Stranded"
                plus_bedgraph_multi=outfolder + "/" + basename +
"Signal.UniqueMultiple.str1.out.bg"
                minus_bedgraph_multi = outfolder + "/" + basename +
"Signal.UniqueMultiple.str2.out.bg"
                plus_bedgraph_unique=outfolder + "/" + basename + "Signal.Unique.str1.out.bg"
                minus_bedgraph_unique = outfolder + "/" + basename + "Signal.Unique.str2.out.bg"
        else:
                stranded_yesno="Unstranded"
                bedgraph_unique = outfolder + "/" + basename + "Signal.Unique.str1.out.bg"
                bedgraph_multi = outfolder + "/" + basename + "Signal.UniqueMultiple.str1.out.bg"
        inputs = ["""--inputBAMfile""", infile]
        outputs = ["""--outWigType""", "bedGraph", """--outWigStrand""", stranded_yesno, """--
outFileNamePrefix""", outfolder + "/" + basename]
        normalization=["""--outWigNorm""", "None"]
```

```
        STARcommand_list = ["STAR","""--runMode""","inputAlignmentsFromBAM"] + inputs +
outputs + normalization
        STARcommand=" ".join(STARcommand_list)
        sp.check_call(["/bin/sh", "-c", STARcommand])
        if strandedness !=0:
                rename_file(plus_bedgraph_unique,outfolder + "/" + basename +
"_plus_unique.bedgraph")
                rename_file(minus_bedgraph_unique,outfolder + "/" + basename +
"_minus_unique.bedgraph")
                rename_file(plus_bedgraph_multi,outfolder + "/" + basename +
"_plus_multi.bedgraph")
                rename_file(minus_bedgraph_multi,outfolder + "/" + basename +
"_minus_multi.bedgraph")
        else:
                rename_file(bedgraph_unique,outfolder + "/" + basename + "_unique.bedgraph")
                rename_file(bedgraph_multi,outfolder + "/" + basename + "_multi.bedgraph")

    def main(**kwargs):
        ######## ARGUMENTS ###########
        #check if already args is provided, i.e. main() is called from the top level script
        args = kwargs.get('args', None)
        if args is None: ## i.e. standalone script called from command line in normal way
                parser = argparse.ArgumentParser(description = """Quantifies RNAseq reads
aligning to TEs. Outputs TE count file and subfamily count file""")
                parser._optionals.title = "Arguments"
                parser.add_argument("-m","--map_folder", help = "Folder location of outputs from
SQuIRE Map (optional, default = 'squire_map')", type = str, metavar =
"<folder>",default="squire_map")
                parser.add_argument("-c","--clean_folder", help = "Folder location of outputs from
SQuIRE Clean (optional, default = 'squire_clean')", type = str, metavar = "<folder>",default =
"squire_clean")
                parser.add_argument("-o","--count_folder", help = "Destination folder for output
files(optional, default = 'squire_count')", type = str, metavar = "<folder>", default="squire_count")
                parser.add_argument("-t","--tempfolder", help = "Folder for tempfiles (optional;
default=count_folder')", type = str, metavar = "<folder>", default=False)
                parser.add_argument("-f","--fetch_folder", help = "Folder location of outputs from
SQuIRE Fetch (optional, default = 'squire_fetch')",type = str, metavar =
"<folder>",default="squire_fetch")
                parser.add_argument("-r","--read_length", help = "Read length (if trim3 selected,
after trimming; required).", type = int, metavar = "<int>", required=True)
                parser.add_argument("-n","--name", help = "Common basename for input files
(required if more than one bam file in map_folder)", type = str, metavar = "<str>",default=False)
                parser.add_argument("-b","--build", help = "UCSC designation for genome build, eg.
'hg38' (required if more than 1 build in clean_folder)", type=str, metavar = "<build>",default=False)
                parser.add_argument("-p","--pthreads", help = "Launch <int> parallel
threads(optional; default='1')", type = int, metavar = "<int>", default=1)
                parser.add_argument("-s","--strandedness", help = " '0' if unstranded eg Standard
Illumina, 1 if first-strand eg Illumina Truseq, dUTP, NSR, NNSR, 2 if second-strand, eg Ligation,
Standard SOLiD (optional,default=0)", type = int, metavar = "<int>", default = 0)
```

```
		parser.add_argument("-e","--EM", help = "Run estimation-maximization on TE counts given number of times (optional, specify 0 if no EM desired; default=auto)", type=str, default = "auto")
		parser.add_argument("-v","--verbosity", help = "Want messages and runtime printed to stderr (optional; default=False)", action = "store_true", default = False)
		args = parser.parse_args()
	########## I/O #########
		###### ARGUMENTS ######
		map_folder = args.map_folder
		clean_folder = args.clean_folder
		count_folder=args.count_folder
		fetch_folder=args.fetch_folder
		read_length=args.read_length
		tempfolder=args.tempfolder
		basename = args.name
		pthreads=args.pthreads
		strandedness=args.strandedness
		EM=args.EM
		build = args.build
		verbosity=args.verbosity

		######### START TIMING SCRIPT #############
		if verbosity:
			startTime = datetime.now()
			print("start time is:" + str(startTime) + '\n', file = sys.stderr)# Prints start time
			print(os.path.basename(__file__) + '\n', file = sys.stderr) #prints script name to std err
			print("Script Arguments" + '\n' + "==================", file = sys.stderr)
			args_dict = vars(args)
			for option,arg in args_dict.iteritems():
				print(str(option) + "=" + str(arg), file = sys.stderr) #prints all arguments to std err
			print("\n", file = sys.stderr)

		debug = False
		######## FIND INPUTS #############
		outfolder = count_folder
		make_dir(outfolder) #Create outfolder if doesn't exist
		if not os.path.isdir(clean_folder):
			raise Exception(clean_folder + " is not a folder" )
		if not os.path.isdir(map_folder):
			raise Exception(map_folder + " is not a folder" )

		logfile = find_file(map_folder,".log",basename, 1,False)
		bamfile = find_file(map_folder,".bam",basename,1,True)
		if not bamfile:
			if basename:
				raise Exception("Cannot find bamfile matching " + basename )
			else:
				raise Exception("Cannot find bamfile in map_folder" )
		if not basename:
			basename = get_basename(bamfile)
```

```
	rmsk_bed=find_file(clean_folder,".bed",build,1,True)
	copies = find_file(clean_folder,"_copies.txt",build,1,True)
	if not rmsk_bed:
		if build:
			raise Exception("Cannot find bedfile matching " + build )
		else:
			raise Exception("Cannot find bedfile in clean_folder" )
	if not copies:
		if build:
			raise Exception("Cannot find copies.txt file matching " + build )
		else:
			raise Exception("Cannot find copies.txt file in clean_folder")

	if not tempfolder:
		tempfolder = outfolder

	paired_end = is_paired(bamfile,basename,tempfolder,debug)

	if verbosity:
		print("Quantifying Gene expression "+ str(datetime.now())  + "\n",file = sys.stderr)
	gtf = find_file(fetch_folder,"_refGene.gtf",build,1,True)
	outgtf_ref =outfolder + "/" + basename + ".gtf"
	abund_ref =outgtf_ref.replace(".gtf","_abund.txt")
	outgtf_ref_temp =  make_tempfile(basename, "outgtf_ref", tempfolder)
	abund_ref_temp = outgtf_ref_temp.replace("outgtf","outabund")
	Stringtie(bamfile,outfolder,basename,strandedness,pthreads,gtf, verbosity,outgtf_ref_temp)
	sort_coord(outgtf_ref_temp,outgtf_ref,1,4,debug)
	sort_coord(abund_ref_temp,abund_ref,3,5,debug)
	gene_dict={}
	filter_abund(abund_ref,gene_dict,False)
	genecounts=outfolder + "/" + basename + "_refGenecounts.txt"
	filter_tx(outgtf_ref, gene_dict,read_length,genecounts)


	#### OPEN OUTPUTS & WRITE HEADER INFORMATION#############
	if verbosity:
		print("Creating temporary files"+ str(datetime.now())  + "\n",file = sys.stderr)
	counts_temp = tempfile.NamedTemporaryFile(delete=False, dir = tempfolder, prefix="count"
+  ".tmp")
	countsfilepath = outfolder + "/" + basename + "_TEcounts.txt"
	counts_file_header = open(countsfilepath +".header",'w')

	counts_file_header.writelines("tx_chr" + "\t"  + "tx_start"  + "\t" + "tx_stop"  + "\t" +
"TE_ID" + "\t" + "fpkm"  + "\t" + "tx_strand" + "\t" + "Sample" + "\t" + "alignedsize" + "\t" +
"TE_chr" + "\t" + "TE_start" + "\t" + "TE_stop" + "\t" + "TE_name" + "\t" + "milliDiv" + "\t" +
"TE_strand" + "\t" + "uniq_counts" + "\t" + "tot_counts"  + "\t" + "tot_reads" +"\t" + "score" + "\n" )

	counts_file_header.close()


#####CREATE TEMPFILES #######
```

```
	if verbosity:
		print("Creating unique and multiple alignment bedfiles "+ str(datetime.now())  +
"\n",file = sys.stderr)

	if not paired_end:
		single_bam = bamfile
		if verbosity:
			print("Intersecting bam file with TE bedfile "+ str(datetime.now())  +
"\n",file = sys.stderr)
		#intersect bam files with TE bed files
		single_bed_tempfile1 = make_tempfile(basename,"single_bed_1", tempfolder)
		intersect_flank(single_bam, rmsk_bed, single_bed_tempfile1,debug)
		if verbosity:
			print("Combining adjacent TEs with same read alignment "+
str(datetime.now())  + "\n",file = sys.stderr)
		#reduce reads   #Find reads aligned to same position but different TE_IDs
(overlapping flanks) and merge
		single_reduced_tempfile1 = make_tempfile(basename,"single_reduced_1",
tempfolder)
		single_reduced_tempfile1_sorted =single_reduced_tempfile1  + "_sorted"
		sort_coord(single_bed_tempfile1,single_reduced_tempfile1_sorted,1,2,debug)
		reduce_reads(single_reduced_tempfile1_sorted, single_reduced_tempfile1,debug)
		if verbosity:
			print("Getting genomic coordinates of read"+ str(datetime.now())  + "\n",file
= sys.stderr)
		#get genomic coordinates and RNA strand for all alignments
		single_coords_tempfile1= make_tempfile(basename,"single_coords_1", tempfolder)

	get_coords(single_reduced_tempfile1,1,strandedness,single_coords_tempfile1,debug)

		# os.unlink(single_bed_tempfile1)

		if verbosity:
			print("Identifying and labeling unique and multi reads"+ str(datetime.now())
+ "\n",file = sys.stderr)
		single_labeled_tempfile1 = make_tempfile(basename,"single_labeled_1",
tempfolder)
		single_labeled_tempfile2 = make_tempfile(basename,"single_labeled_2",
tempfolder)
		label_files(single_coords_tempfile1,single_labeled_tempfile1,"single",debug)
		label_files(single_labeled_tempfile1,single_labeled_tempfile2,"R1",debug)

		#find unique single alignments
		first_tempfile1 = make_tempfile(basename,"first_1", tempfolder)
		unique_tempfile1 = make_tempfile(basename,"unique_1", tempfolder)
		multi_tempfile1 = make_tempfile(basename,"multi_1", tempfolder)

		find_uniq(single_labeled_tempfile2,first_tempfile1,unique_tempfile1,
multi_tempfile1,debug)
```

```
		#label uniq, multi, or single
		multi_bed = make_tempfile(basename,"multi_bed", tempfolder)
		unique_bed = make_tempfile(basename,"unique_bed", tempfolder)

		label_files(unique_tempfile1, unique_bed, "uniq",debug)
		label_files(multi_tempfile1, multi_bed, "multi",debug)

		aligned_libsize = getlibsize(logfile,
bamfile,multi_bed,unique_bed,paired_end,debug)

	if paired_end:
		#intersect bam files with TE bed files
		if verbosity:
			print("Identifying properly paired reads "+ str(datetime.now())  + "\n",file =
sys.stderr)
		paired_bam = bamfile
		proper_bam = make_tempfile(basename,"proper_bam", tempfolder)
		nonproper_bam = make_tempfile(basename,"nonproper_bam", tempfolder)
		find_properpair(paired_bam, proper_bam,nonproper_bam)

		if verbosity:
			print("Intersecting bam files with TE bedfile "+ str(datetime.now())  +
"\n",file = sys.stderr)

		proper_bed = make_tempfile(basename,"proper_bed", tempfolder)
		nonproper_bed = make_tempfile(basename,"nonproper_bed", tempfolder)
		intersect_flank(proper_bam, rmsk_bed, proper_bed,debug)
		intersect_flank(nonproper_bam, rmsk_bed, nonproper_bed,debug)

		proper_labeled_tempfile = make_tempfile(basename,"proper_labeled", tempfolder)
		nonproper_labeled_tempfile = make_tempfile(basename,"nonproper_labeled",
tempfolder)
		label_files(proper_bed,proper_labeled_tempfile,"proper",debug)
		label_files(nonproper_bed,nonproper_labeled_tempfile,"nonproper",debug)

		proper_nonproper_labeled_tempfile =
make_tempfile(basename,"proper_nonproper_labeled", tempfolder)

	combine_files(proper_labeled_tempfile,nonproper_labeled_tempfile,proper_nonproper_label
ed_tempfile,debug)

		if verbosity:
			print("Splitting into read1 and read 2 "+ str(datetime.now())  + "\n",file =
sys.stderr)
		paired_bed_tempfile1 = make_tempfile(basename,"paired_1.bed",tempfolder)
		paired_bed_tempfile2 = make_tempfile(basename,"paired_2.bed",tempfolder)

	split_paired(proper_nonproper_labeled_tempfile,paired_bed_tempfile1,paired_bed_tempfile2
,debug)
		paired_bed_tempfile1_sorted = paired_bed_tempfile1 + "_sorted"
		paired_bed_tempfile2_sorted = paired_bed_tempfile2 + "_sorted"
```

```
                if not debug:
                        os.unlink(proper_bam)
                        os.unlink(nonproper_bam)
                #reduce reads   #Find reads aligned to same position but different TE_IDs
(overlapping flanks) and merge
                if verbosity:
                        print("Combining adjacent TEs with same read alignment "+
str(datetime.now())  + "\n",file = sys.stderr)
                paired_reduced_tempfile1 = make_tempfile(basename,"paired_reduced_1",
tempfolder)
                paired_reduced_tempfile2 = make_tempfile(basename,"paired_reduced_2",
tempfolder)
                sort_coord(paired_bed_tempfile1,paired_bed_tempfile1_sorted,1,2,debug)
                sort_coord(paired_bed_tempfile2,paired_bed_tempfile2_sorted,1,2,debug)

                reduce_reads(paired_bed_tempfile1_sorted, paired_reduced_tempfile1,debug)
                reduce_reads(paired_bed_tempfile2_sorted, paired_reduced_tempfile2,debug)

                #get genomic coordinates and RNA strand for all alignments
                if verbosity:
                        print("Getting genomic coordinates of read"+ str(datetime.now())  + "\n",file
= sys.stderr)
                paired_coords_tempfile1= make_tempfile(basename,"paired_coords_1", tempfolder)
                paired_coords_tempfile2= make_tempfile(basename,"paired_coords_2", tempfolder)

        get_coords(paired_reduced_tempfile1,1,strandedness,paired_coords_tempfile1,debug)

        get_coords(paired_reduced_tempfile2,2,strandedness,paired_coords_tempfile2,debug)

                paired_labeled_tempfile1 = make_tempfile(basename,"paired_labeled_1",
tempfolder)
                paired_labeled_tempfile2 = make_tempfile(basename,"paired_labeled_2",
tempfolder)
                label_files(paired_coords_tempfile1,paired_labeled_tempfile1,"R1",debug)
                label_files(paired_coords_tempfile2,paired_labeled_tempfile2,"R2",debug)

                #remove /1 and /2 from read ID column
                paired_fixed_tempfile1 = make_tempfile(basename,"paired_fixed_1", tempfolder)
                paired_fixed_tempfile2 = make_tempfile(basename,"paired_fixed_2", tempfolder)
                fix_paired(paired_labeled_tempfile1,paired_labeled_tempfile2,
paired_fixed_tempfile1,paired_fixed_tempfile2,debug)

                #find unique single alignments
                if verbosity:
                        print("Identifying and labeling unique and multi reads"+ str(datetime.now())
+ "\n",file = sys.stderr)
                first_tempfile1 = make_tempfile(basename,"first_1", tempfolder)
                unique_tempfile1 = make_tempfile(basename,"unique_1", tempfolder)
                multi_tempfile1 = make_tempfile(basename,"multi_1", tempfolder)
```

```
		first_tempfile2 = make_tempfile(basename,"first_2", tempfolder)
		unique_tempfile2 = make_tempfile(basename,"unique_2", tempfolder)
		multi_tempfile2 = make_tempfile(basename,"multi_2", tempfolder)

		find_uniq(paired_fixed_tempfile1,first_tempfile1,unique_tempfile1,
multi_tempfile1,debug)
		find_uniq(paired_fixed_tempfile2,first_tempfile2, unique_tempfile2,
multi_tempfile2,debug)

		#label uniq, multi, or paired

		unique_tempfile1_labeled = make_tempfile(basename,"unique_labeled_1",
tempfolder)
		multi_tempfile1_labeled = make_tempfile(basename,"multi_labeled_1", tempfolder)

		unique_tempfile2_labeled = make_tempfile(basename,"unique_labeled_2",
tempfolder)
		multi_tempfile2_labeled = make_tempfile(basename,"multi_labeled_2", tempfolder)

		label_files(unique_tempfile1, unique_tempfile1_labeled, "uniq",debug)
		label_files(unique_tempfile2, unique_tempfile2_labeled, "uniq",debug)
		label_files(multi_tempfile1, multi_tempfile1_labeled, "multi",debug)
		label_files(multi_tempfile2, multi_tempfile2_labeled, "multi",debug)

		paired_tempfile1_ulabeled = make_tempfile(basename,"paired_ulabeled_1",
tempfolder)
		paired_tempfile2_ulabeled = make_tempfile(basename,"paired_ulabeled_2",
tempfolder)
		combine_files(unique_tempfile1_labeled,multi_tempfile1_labeled,
paired_tempfile1_ulabeled,debug)
		combine_files(unique_tempfile2_labeled,multi_tempfile2_labeled,
paired_tempfile2_ulabeled,debug)

		#combine pairs
		if verbosity:
			print("Matching paired-end mates and merging coordinates"+
str(datetime.now())  + "\n",file = sys.stderr)
		paired_unmatched1= make_tempfile(basename,"paired_unmatched_1", tempfolder)
		paired_unmatched2 = make_tempfile(basename,"paired_unmatched_2", tempfolder)
		paired_matched_tempfile = make_tempfile(basename,"paired_matched", tempfolder)

	match_reads(paired_tempfile1_ulabeled,paired_tempfile2_ulabeled,strandedness,paired_matc
hed_tempfile,paired_unmatched1, paired_unmatched2,debug) #match pairs between paired files

		#sort matched
		matched_tempfile_sorted = make_tempfile(basename,"paired_matched_sorted",
tempfolder)
		sort_temp(paired_matched_tempfile,4,matched_tempfile_sorted,debug)

		#combine start and stop of paired reads
		matched_bed = make_tempfile(basename,"matched_bed", tempfolder)
```

```
		merge_coords(matched_tempfile_sorted,matched_bed,debug)

		# os.unlink(matched_tempfile_sorted)

		combined_unmatched = make_tempfile(basename,"combined_unmatched",
tempfolder)
		combine_files(paired_unmatched1, paired_unmatched2,
combined_unmatched,debug)

		#Find single reads that are matched outside of TE but are still proper pair
		if verbosity:
			print("Adding properly paired reads that have mates outside of TE into
matched file"+ str(datetime.now())  + "\n",file = sys.stderr)
		proper_single = make_tempfile(basename,"proper_single",tempfolder)
		combined_unmatched2 = make_tempfile(basename,"combined_unmatched2",
tempfolder)

		find_proper(combined_unmatched,combined_unmatched2,proper_single,debug)
#outputs only nonproper pairs in combined_unmatched2, and single reads that are part of proper pairs
in proper_single

		combined_matched = make_tempfile(basename,"combined_matched", tempfolder)
		combine_files(matched_bed,proper_single,combined_matched,debug)
		# os.unlink(proper_single)
		###Remove single alignments of reads that have paired matches using other valid
alignments
		if verbosity:
			print("Removing single-end reads that have matching paired-end mates at
other alignment locations"+ str(datetime.now())  + "\n",file = sys.stderr)
		only_unmatched = make_tempfile(basename,"only_unmatched", tempfolder)

	remove_repeat_reads(combined_matched,combined_unmatched2,only_unmatched,debug)

		#combine matched and unmatched alignments
		combined_bed = make_tempfile(basename,"combined_bed", tempfolder)
		combine_files(combined_matched,only_unmatched,combined_bed,debug)

		if verbosity:
			print("Identifying and labeling unique and multi fragments"+
str(datetime.now())  + "\n",file = sys.stderr)
		first_tempfile = make_tempfile(basename,"first", tempfolder)
		paired_uniq_tempfile =
make_tempfile(basename,"paired_uniq_tempfile",tempfolder)
		unique_bed = make_tempfile(basename,"unique_bed", tempfolder)
		multi_bed_pre = make_tempfile(basename,"multi_bed_pre", tempfolder)
		find_uniq(combined_bed,first_tempfile,unique_bed,multi_bed_pre,debug)

		#find unique pairs in multi_bed
		multi_bed = make_tempfile(basename,"multi_bed", tempfolder)
		if verbosity:
```

```
                print("Identifying multi read pairs with one end unique"+ str(datetime.now())
+ "\n",file = sys.stderr)
                find_paired_uniq(multi_bed_pre,paired_uniq_tempfile,multi_bed,unique_bed,debug)

                aligned_libsize = getlibsize(logfile,
bamfile,multi_bed,unique_bed,paired_end,debug)

        ######## COUNT READ(S) ##########################
        read_multidict={}  #dictionary to store TE_IDs for each read alignment
        read_locdict = {}  #dictionary to store genomic location of each alignment

        if verbosity:
                print("counting unique alignments "+ str(datetime.now())  + "\n",file = sys.stderr)

        unique_bedfile = open(unique_bed,'r')
        avg_fraglength=uniquecount(unique_bedfile,RepCalc_dict,read_locdict)

        if verbosity:
                print("counting multi alignments "+ str(datetime.now()) + "\n",file = sys.stderr)

        multi_bedfile = open(multi_bed, 'r')
        multicount(multi_bedfile,RepCalc_dict,read_multidict,read_locdict)

        unique_bedfile.close()
        multi_bedfile.close()

        if verbosity:
                print("Adding Tag information to aligned TEs "+ str(datetime.now())  + "\n",file =
sys.stderr)

        for TE_ID,RepClass in RepCalc_dict.iteritems():
                RepClass.calcuniqRep()

        if verbosity:
                                print("Calculating multialignment assignments "+
str(datetime.now())  + "\n",file = sys.stderr)

        comparedict(read_multidict,RepCalc_dict)
        iteration=0
        if EM == "auto":
                        notconverged=True
                        prev_read_change=1
                        prev_count_change = 0
                        max_count_change = 0
                        while notconverged:
                                iteration +=1
                                changed_count = 0
                                total_TE =0
                                total_TE_0 = 0
                                total_TE_1 = 0
                                total_TE_10 = 0
```

```
                                avg_changed_count_pct =0
                                max_count_change=0
                                total_TE_10_1pct =0
                                if verbosity:
                                        print("Running expectation-maximization calculation for
iteration:" + str(iteration) + " " + str(datetime.now())  + "\n",file = sys.stderr)

                                for TE_ID,RepClass in RepCalc_dict.iteritems():
                                        TE_changecount = RepClass.calcmultiRep(iteration)
                                        max_count_change =
max(TE_changecount,max_count_change)
                                        changed_count +=TE_changecount
                                        total_TE +=1
                                        if TE_changecount > 0:
                                                total_TE_0 +=1
                                        if TE_changecount >= 1:
                                                total_TE_1 += 1
                                        if TE_changecount >= 1 and RepClass.counts_tot >= 10:
                                                total_TE_10 += 1
                                        if TE_changecount >= 1 and RepClass.counts_tot >= 10 and
(TE_changecount/RepClass.counts_tot) > 0.01:
                                                total_TE_10_1pct += 1
                                        avg_changed_count_pct = changed_count/total_TE

                                if verbosity:
                                        print("Average change in TE count:" +
str(avg_changed_count_pct) + " " + str(datetime.now())  + "\n",file = sys.stderr)
                                        print("Max change in TE count:" + str(max_count_change) +
" " + str(datetime.now())  + "\n",file = sys.stderr)
                                        print("Number changed TE:" + str(total_TE_0) + " " +
str(datetime.now())  + "\n",file = sys.stderr)
                                        print("Number TEs changed by at least 1 count:" +
str(total_TE_1) + " " + str(datetime.now())  + "\n",file = sys.stderr)
                                        print("Number TEs changed by at least 1 count with at least
10 counts:" + str(total_TE_10) + " " + str(datetime.now())  + "\n",file = sys.stderr)
                                        print("Number TEs changed by at least 1 count with at least
10 counts and > 1pct total count:" + str(total_TE_10_1pct) + " " + str(datetime.now())  + "\n",file =
sys.stderr)
                                new_read_change = estdict(read_multidict,RepCalc_dict)
                                if total_TE_10_1pct == 0 and iteration > 1:
                                        notconverged = False
                                else:
                                        prev_read_change = new_read_change
                        if verbosity:
                                print("Finished running expectation-maximization calculation after
iteration:" + str(iteration) + " " + str(datetime.now())  + "\n",file = sys.stderr)

        elif int(EM) > 0:
                notconverged=True
                prev_read_change=1
                prev_count_change = 0
```

```
		max_count_change = 0
		while iteration < int(EM):
			iteration +=1
			changed_count = 0
			total_TE =0
			total_TE_0 = 0
			total_TE_1 = 0
			total_TE_10 = 0
			avg_changed_count_pct =0
			max_count_change=0
			total_TE_10_1pct =0
			if verbosity:
				print("Running expectation-maximization calculation for iteration:"
+ str(iteration) + " " + str(datetime.now())  + "\n",file = sys.stderr)

			for TE_ID,RepClass in RepCalc_dict.iteritems():
				TE_changecount = RepClass.calcmultiRep(iteration)
				max_count_change = max(TE_changecount,max_count_change)
				changed_count +=TE_changecount
				total_TE +=1
				if TE_changecount > 0:
					total_TE_0 +=1
				if TE_changecount >= 1:
					total_TE_1 += 1
				if TE_changecount >= 1 and RepClass.counts_tot >= 10:
					total_TE_10 += 1
				avg_changed_count_pct = changed_count/total_TE
				if TE_changecount >= 1 and RepClass.counts_tot >= 10 and
(TE_changecount/RepClass.counts_tot) > 0.01:
					total_TE_10_1pct += 1
			if verbosity:
				print("Average change in TE count:" + str(avg_changed_count_pct)
+ " " + str(datetime.now())  + "\n",file = sys.stderr)
				print("Max change in TE count:" + str(max_count_change) + " " +
str(datetime.now())  + "\n",file = sys.stderr)
				print("Number changed TE:" + str(total_TE_0) + " " +
str(datetime.now())  + "\n",file = sys.stderr)
				print("Number TEs changed by at least 1 count:" + str(total_TE_1) +
" " + str(datetime.now())  + "\n",file = sys.stderr)
				print("Number TEs changed by at least 1 count with at least 10
counts:" + str(total_TE_10) + " " + str(datetime.now())  + "\n",file = sys.stderr)
				print("Number TEs changed by at least 1 count with at least 10
counts and > 1pct total count:" + str(total_TE_10_1pct) + " " + str(datetime.now())  + "\n",file =
sys.stderr)
			new_read_change = estdict(read_multidict,RepCalc_dict)

	if verbosity:
				print("Writing counts "+ str(datetime.now())  + "\n",file = sys.stderr)

	read_multidict.clear()
```

```
	if copies:
		temp_subF = tempfile.NamedTemporaryFile(delete=False, dir = tempfolder,
prefix="count" +  ".SFtmp")
		subF_filepath = outfolder + "/" + basename + "_subFcounts.txt"
		subF_file_header = open(subF_filepath + ".header",'w')
	#		subF_file_header.writelines("Sample" + "\t" + "aligned_libsize" + "\t" +
"Subfamily:Family:Class" + "\t" + "copies" + "\t" + "exp_copies" + "\t" + "uniq_counts" + "\t" +
"tot_counts" + "\t" + "avg_conf"  + "\t" + "tot_sense" + "\t" + "tot_antisense" + "\n")
		subF_file_header.writelines("Sample" + "\t" + "aligned_libsize" + "\t" +
"Subfamily:Family:Class" + "\t" + "copies"  + "\t" + "fpkm" + "\t" + "uniq_counts" + "\t" +
"tot_counts" + "\t" + "tot_reads" + "\t" + "score"  + "\n")

		subF_file_header.close()
		subF_dict = {}

	for TE_ID,RepClass in RepCalc_dict.iteritems(): #for each TE_ID
		RepClass.writeRep(aligned_libsize,counts_temp,basename,strandedness,iteration)
	##########Sort by highest total counts before writing and add to subF dictionary

		if copies:
			subF = get_subF(TE_ID)
			subF_list = split_subF(subF)
			if subF not in subF_dict:
				subF_dict[subF]=subfamily(subF,subF_reads[subF])
				subF_dict[subF].add_TE_count(RepClass,strandedness)
			else:
				subF_dict[subF].add_TE_count(RepClass,strandedness)

	#Close dictionaries from memory
	counts_temp.close()

	sort_counts(counts_temp.name,counts_file_header.name,countsfilepath,5,debug) #sort on 5th
column (fpkm)
	read_locdict.clear()
	RepCalc_dict.clear()

	if copies:
		if verbosity:
				print("Writing subfamily counts "+ str(datetime.now())  + "\n",file =
sys.stderr)
		with open(copies,'r') as copiesfile: #copiesfile is sorted
			copiesfile.readline() #skip header
			for line in copiesfile:
				line = line.rstrip()
				line_tabs = line.split("\t")
				line_subF = line_tabs[0]
				if line_subF in subF_dict:
					subF_dict[line_subF].add_copy_info(line_tabs)
				#### Write lines ###
```

```
                                        # temp_subF.writelines(basename + "\t" + str(aligned_libsize) + "\t"
+ line_subF + "\t" + line_copies + "\t" + str(uniq) + "\t" + "{0:.2f}".format(multi) + "\t" +
str(subF_conf) + "\t" + "{0:.2f}".format(sense_reads) + "\t"  +  "{0:.2f}".format(antisense_reads) +
"\n")

        subF_dict[line_subF].write_subfamily(temp_subF,basename,aligned_libsize,iteration)
                                        else:
                                                subF_dict[line_subF]=subfamily(line_subF,0)
                                                subF_dict[line_subF].add_copy_info(line_tabs)

        subF_dict[line_subF].write_subfamily(temp_subF,basename,aligned_libsize,iteration)

                copiesfile.close()
                temp_subF.close()

                sort_counts(temp_subF.name,subF_file_header.name,subF_filepath,6,debug) #Sort
by 7th field (multi)

                if not debug:
                        os.unlink(unique_bed)
                        os.unlink(multi_bed)

        ######## STOP TIMING SCRIPT ########################
        if verbosity:
                print("finished writing outputs at "+ str(datetime.now()) + "\n",file = sys.stderr)

                endTime = datetime.now()
                print('end time is: '+ str(endTime) + "\n", file = sys.stderr)
                print('it took: ' + str(endTime-startTime) + "\n", file = sys.stderr)

    ####################
    if __name__ == "__main__":
        main()
```

Appendix F. SQuIRE Call

```
#!/usr/bin/env python

############MODULES##########################
from __future__ import print_function,division
import sys
import os
import errno
import argparse #module that passes command-line arguments into script
from datetime import datetime
import operator #for doing operations on tuple
from operator import itemgetter
import subprocess as sp
from subprocess import Popen, PIPE,STDOUT
import io
import tempfile
from collections import defaultdict #for dictionary
import glob
import re
from six import itervalues
import call_deseq2
import call_deseq2_prefilter
import shutil


def find_file(folder,pattern,base, wildpos, needed):
    foundfile=False
    if wildpos == 1:
        file_list=glob.glob(folder + "/" + "*" + pattern)
    elif wildpos ==2:
        file_list=glob.glob(folder + "/" + pattern + "*")
    if len(file_list)>1: #if more than one file in folder
        if not base:
            raise Exception("More than 1 " + pattern + " file")
        for i in file_list:
            if base in i:
                foundfile = i
    elif len(file_list) == 0:
        foundfile = False
    else:
        foundfile = file_list[0]
    if not foundfile:
        if needed:
            raise Exception("No " + pattern + " file")
        else:
            foundfile = False
    return foundfile

def make_tempfile(basename, step, outfolder):
```

```
        tmpfile = tempfile.NamedTemporaryFile(delete=False, dir = outfolder, prefix= basename + "_"
+ step +  ".tmp")
        tmpname = tmpfile.name
        tmpfile.close()
        return tmpname

    def filter_files(file_in,file_out, string, column):
        command = "'$" + str(column) + "==" + "'" + string + "'"+ "'"
        pastecommandlist = ["awk", "-v", "OFS='\\t'",command,file_in, ">", file_out]
        pastecommand = " ".join(pastecommandlist)
        sp.check_call(["/bin/sh","-c",pastecommand])

    def rename_file(oldname,newname):
        shutil.move(oldname, newname)

    def make_dir(path):
        try:
            original_umask = os.umask(0)
            os.makedirs(path, 0770)
        except OSError as exception:
            if exception.errno != errno.EEXIST:
                raise
        finally:
            os.umask(original_umask)

    def get_basename(filepath):
            filename = os.path.basename(filepath)
            filebase = os.path.splitext(filename)[0]
            return filebase

    def get_groupfiles(group,gene_files,subF_files,TE_files,subfamily,count_folder):
        if "*" not in group:
            if "," in group:
                group_list = group.split(",")
            else:
                group_list=[group]
            for sample in group_list:
                if subfamily:
                    subF_files.append(find_file(count_folder,"_subFcounts.txt",sample,1,True))
                else:
                    TE_files.append(find_file(count_folder,"_TEcounts.txt",sample,1,True))
                gene_files.append(glob.glob(count_folder + "/" + sample + "_refGenecounts.txt")[0])
        elif "*" in group:
            if subfamily:
                subF_files+=(glob.glob(count_folder + "/" + group + "_subFcounts.txt"))
            else:
                TE_files+=(glob.glob(count_folder + "/" + group + "_TEcounts.txt"))
            gene_files+=(glob.glob(count_folder + "/" + group + "_refGenecounts.txt"))
            group_list=[get_basename(gene_file).replace("_refGenecounts","") for gene_file in
(glob.glob(count_folder + "/" + group + "_refGenecounts.txt"))]
        return group_list
```

```
    def create_count_dict(infilepath,count_dict,stringtie_list):
        name=get_basename(infilepath).replace("_refGenecounts","")
        with open(infilepath,'r') as infile:
            header = infile.readline().rstrip()
            for line in infile:
                line = line.rstrip()
                line = line.split("\t")
                chrom = line[0]
                start=line[1]
                stop = line[2]
                gene_ID = line[3]
                fpkm=line[4]
                strand = line[5]
                count = line[6]
                if (gene_ID,strand) in stringtie_list:
                    continue
                if (gene_ID,strand) not in count_dict:
                    count_dict[(gene_ID,strand)] = {name:count}
                else:
                    count_dict[(gene_ID,strand)][name]=count

    #subF_file_header.writelines("Sample" + "\t" + "aligned_libsize" + "\t" +
"Subfamily:Family:Class" + "\t" + "copies"  + "\t" + "EM_iteration" + "\t" + "uniq_counts" + "\t" +
"tot_counts_preEM" + "\t" + "tot_counts_postEM" + "\t" + "tot_reads" + "\t" + "avg_conf"  + "\n")

    def create_TE_dict(infilepath,sample_count_dict,threshold):
        conf_dict={}
        count_dict={}
        with open(infilepath,'r') as infile:
            header = infile.readline().rstrip()
            for line in infile:
                line = line.rstrip()
                line = line.split("\t")
                if "milliDiv" in line[12]:
                    continue
                TE_ID = line[3]
                strand = line[5]
                milliDiv = int(line[12])
                count = str(int(float(line[15])))
                conf = float(line[17])
                sample = line[6]
                if (TE_ID,strand) not in sample_count_dict:
                    sample_count_dict[(TE_ID,strand)] = {sample:count}
                    conf_dict[(TE_ID,strand)] = [conf]
                    count_dict[(TE_ID,strand)] = [count]
                else:
                    sample_count_dict[(TE_ID,strand)][sample]=count
                    conf_dict[(TE_ID,strand)].append(conf)
                    count_dict[(TE_ID,strand)].append(count)
```

```
        for TE_tuple,conf_list in conf_dict.iteritems():
            mean_conf=sum(conf_list)/len(conf_list)
            if mean_conf <= threshold:
                sample_count_dict.pop(TE_tuple, None)
        for TE_tuple,TEcount_list in count_dict.iteritems():
            TEcount_list=TEcount_list = [int(i) for i in TEcount_list]
            mean_count=sum(TEcount_list)/len(TEcount_list)
            if mean_count <= 5:
                sample_count_dict.pop(TE_tuple, None)

    def create_subfamily_dict(infilepath,count_dict):
        TE_classes=["LTR","LINE","SINE","Retroposon","DNA","RC"]
        with open(infilepath,'r') as infile:
            for line in infile:
                line = line.rstrip()
                line = line.split("\t")
                taxo = line[2]
                count=line[6]
                if any(x in taxo for x in TE_classes):
                    if count=="tot_counts":
                        continue
                    else:
                        count = str(int(round(float(line[5]))))
                    sample = line[0]
                    if taxo not in count_dict:
                        count_dict[taxo] = {sample:count}
                    else:
                        count_dict[taxo][sample]=count

    def combinefiles(infile,catfile):
        with open(catfile, 'a') as outFile:
            with open(infile, 'rb') as inFile:
                shutil.copyfileobj(inFile, outFile)

    def
create_rscript(count_table,coldata,outfolder,output_format,projectname,verbosity,pthreads,prefilter,c
ondition1,condition2,label_no):
        r_script = make_tempfile(projectname,"R_script",outfolder)
        outfolder=os.path.abspath(outfolder)
        count_table = os.path.abspath(count_table)
        coldata=os.path.abspath(coldata)
        if prefilter:
            call_deseq2_prefilter.write_Rscript(r_script)
        else:
            call_deseq2.write_Rscript(r_script)
        #outfile = open(outfolder + "/" + projectname + "call_results.txt","w")

        if verbosity:
            print("Creating DESeq2 results"+ str(datetime.now())  + "\n",file = sys.stderr)
```

```
        Rcommandlist = ["Rscript", r_script,
count_table,coldata,outfolder,projectname,pthreads,condition1,condition2,str(label_no)]
        Rcommand = " ".join(Rcommandlist)
        sp.check_call(["/bin/sh","-c",Rcommand])

        # if output_format=="html":
        #     render_command="rmarkdown::render('" + r_script + "')"
        # elif output_format == "pdf":
        #     render_command="rmarkdown::render('" + r_script + "', 'pdf_document')"
        # Rcommandlist = ["R","-e", render_command]
        # Rcommand = " ".join(Rcommandlist)
        # sp.check_call(["/bin/sh","-c",Rcommand])

        os.unlink(r_script)

    def main(**kwargs):

        ######## ARGUMENTS ###########
        #check if already args is provided, i.e. main() is called from the top level script
        args = kwargs.get('args', None)
        if args is None: ## i.e. standalone script called from command line in normal way
            parser = argparse.ArgumentParser(description = """Performs differential expression analysis
on TEs and genes""")
            parser._optionals.title = "Arguments"
            parser.add_argument("-1","--group1", help = "List of basenames for group1 (Treatment)
samples, can also provide string pattern common to all group1 basenames with * ",required = True,
type = str, metavar = "<str1,str2> or <*str*>")
            parser.add_argument("-2","--group2", help = "List of basenames for group2 (Control)
samples, can also provide string pattern common to all group2 basenames with * ",required = True,
type = str, metavar = "<str1,str2> or <*str*>")
            parser.add_argument("-A","--condition1", help = "Name of condition for group1",required =
True, type = str, metavar = "<str>")
            parser.add_argument("-B","--condition2", help = "Name of condition for group2",required =
True, type = str, metavar = "<str>")
            parser.add_argument("-i","--count_folder", help = "Folder location of outputs from SQuIRE
Count (optional, default = 'squire_count')", type = str, metavar = "<folder>",default="squire_count")
            parser.add_argument("-o","--call_folder", help = "Destination folder for output files
(optional; default='squire_call')", type = str, metavar = "<folder>", default="squire_call")
            parser.add_argument("-s","--subfamily", help = "Compare TE counts by subfamily.
Otherwise, compares TEs at locus level (optional; default=False)", action = "store_true", default =
False)
            parser.add_argument("-p","--pthreads", help = "Launch <int> parallel threads(optional;
default='1')", type = int, metavar = "<int>", default=1)
            parser.add_argument("-N","--projectname", help = "Basename for project,
default='SQuIRE'",type = str, metavar = "<str>",default="SQuIRE")
            parser.add_argument("-f","--output_format", help = "Output figures as html or pdf", type =
str, metavar = "<str>",default="html")
            parser.add_argument("-t","--table_only", help = "Output count table only, don't want to
perform differential expression with DESeq2", action = "store_true", default = False)
            #parser.add_argument("-c","--cluster", help = "Want to cluster samples by gene and TE
expression", action = "store_true", default = False)
```

```
        parser.add_argument("-v","--verbosity", help = "Want messages and runtime printed to
stderr (optional; default=False)", action = "store_true", default = False)

        args,extra_args = parser.parse_known_args()
    ########## I/O #########
      ###### ARGUMENTS ######

      group1 = args.group1
      group2 = args.group2
      condition1=args.condition1
      condition2 = args.condition2
      count_folder = args.count_folder
      outfolder=args.call_folder
      verbosity=args.verbosity
      projectname = args.projectname
      subfamily=args.subfamily
      output_format = args.output_format
      pthreads= args.pthreads
      table_only=args.table_only
      debug = True
      label_no=20
      threshold=0
      ######### Call TIMING SCRIPT ############
      if verbosity:
         CallTime = datetime.now()
         print("Script start time is:" + str(CallTime) + '\n', file = sys.stderr)# Prints Call time
         print("Script Arguments" + '\n' + "==================", file = sys.stderr)
         args_dict = vars(args)
         for option,arg in args_dict.iteritems():
            print(str(option) + "=" + str(arg), file = sys.stderr) #prints all arguments to std err
         print("\n", file = sys.stderr)
      if os.path.isfile(outfolder):
         raise Exception (outfolder + " exists as a file" )


      make_dir(outfolder)
      gene_files = []
      subF_files=[]
      TE_files=[]


      group1_list=get_groupfiles(group1,gene_files,subF_files,TE_files,subfamily,count_folder)
      group2_list=get_groupfiles(group2,gene_files,subF_files,TE_files,subfamily,count_folder)

      count_dict = {}
      gene_list=set()

      TE_dict={}
      subF_combo = outfolder + "/" + projectname + "_subF_combo" + ".txt"
      TE_combo = outfolder + "/" + projectname + "_TE_combo" + ".txt"
      if subfamily:
```

```
    for subF in subF_files:
        combinefiles(subF,subF_combo)
    create_subfamily_dict(subF_combo,TE_dict)
else:
    for TE in TE_files:
        combinefiles(TE,TE_combo)
    create_TE_dict(TE_combo,TE_dict,threshold)

for genefile in gene_files:
    create_count_dict(genefile,count_dict,gene_list)

coldata=outfolder + "/" + projectname + "_coldata.txt"
with open(coldata,'w') as datafile:
    datafile.writelines("sample" + "\t" + "condition" + "\n")
    for group1_sample in group1_list:
        datafile.writelines(group1_sample + "\t" + condition1 + "\n")
    for group2_sample in group2_list:
        datafile.writelines(group2_sample + "\t" + condition2 + "\n")
if subfamily:
    counttable = outfolder + "/" + projectname + "_gene_subF_counttable.txt"
else:
    counttable = outfolder + "/" + projectname + "_gene_TE_counttable.txt"

with open(counttable,'w') as DEfile:
    sample_list = group1_list + group2_list
    header_list = ["gene_id"] + sample_list
    header = "\t".join(header_list)
    DEfile.writelines(header + "\n")
    for gene_key,sample_dict in count_dict.iteritems():
        if type(gene_key) is tuple:
            gene=",".join(gene_key)
        else:
            gene=gene_key
        count_list = []
        for sample in sample_list:
            if sample in sample_dict:
                count_list.append(str(sample_dict[sample]))
            else:
                count_list.append("0")
        countline = "\t".join(count_list)
        DEfile.writelines(gene + "\t" + countline + "\n")
    for TE_key,sample_dict in TE_dict.iteritems():
        TE_out=",".join(TE_key)
        count_list = []
        for sample in sample_list:
            if sample in sample_dict:
                count_list.append(str(sample_dict[sample]))
            else:
                count_list.append("0")
        countline = "\t".join(count_list)
        DEfile.writelines(TE_out + "\t" + countline + "\n")
```

```
        prefilter = True
        if not table_only:

create_rscript(counttable,coldata,outfolder,output_format,projectname,verbosity,str(pthreads),prefilter
,condition1,condition2,label_no)

        ####### STOP TIMING SCRIPT ######################
        if verbosity:
           print("finished writing outputs at "+ str(datetime.now()) + "\n",file = sys.stderr)

           endTime = datetime.now()
           print('end time is: '+ str(endTime) + "\n", file = sys.stderr)
           print('it took: ' + str(endTime-CallTime) + "\n", file = sys.stderr)

    ###################
    if __name__ == "__main__":
       main()
```

Appendix G. SQuIRE Draw

```
#!/usr/bin/env python

############MODULES##########################
from __future__ import print_function,division
import sys
import os
import errno
import argparse #module that passes command-line arguments into script
from datetime import datetime
import operator #for doing operations on tuple
from operator import itemgetter
import subprocess as sp
from subprocess import Popen, PIPE,STDOUT
import io
import tempfile
#for creating interval from start
from collections import defaultdict #for dictionary
import glob
import re
from six import itervalues
import shutil


def find_file(folder,pattern,base, wildpos):
    foundfile=False
    needed=False
    if wildpos == 1:
        file_list=glob.glob(folder + "/" + "*" + pattern)
    elif wildpos ==2:
        file_list=glob.glob(folder + "/" + pattern + "*")
    if len(file_list)>1: #if more than one file in folder
        if not base:
            raise Exception("More than 1 " + pattern + " file")
        for i in file_list:
            if base in i:
                foundfile = i
        if not foundfile:
            if needed:
                raise Exception("No " + pattern + " file")
            else:
                foundfile = False
    elif len(file_list) == 0:
        foundfile = False
    else:
        foundfile = file_list[0]
    return foundfile

def make_tempfile(basename, step, outfolder):
```

```
        tmpfile = tempfile.NamedTemporaryFile(delete=False, dir = outfolder, prefix= basename + "_"
+ step +  ".tmp")
        tmpname = tmpfile.name
        tmpfile.close()
        return tmpname

    def filter_files(file_in,file_out, string, column):
        command = "'$" + str(column) + "==" + "'" + string + "'"+ "'"
        pastecommandlist = ["awk", "-v", "OFS='\\t'",command,file_in, ">", file_out]
        pastecommand = " ".join(pastecommandlist)
        sp.check_call(["/bin/sh","-c",pastecommand])

    def rename_file(oldname,newname):
        shutil.move(oldname, newname)

    def make_dir(path):
        try:
            original_umask = os.umask(0)
            os.makedirs(path, 0770)
        except OSError as exception:
            if exception.errno != errno.EEXIST:
                raise
        finally:
            os.umask(original_umask)

    def get_basename(filepath):
            filename = os.path.basename(filepath)
            filebase = os.path.splitext(filename)[0]
            return filebase

    def sort_coord(infile, outfile,chrcol,startcol):
        chrfieldsort = "-k" + str(chrcol) + "," + str(chrcol)
        startfieldsort = "-k" + str(startcol) + "," + str(startcol) + "n"
        sort_command_list = ["sort",chrfieldsort,startfieldsort, infile, ">", outfile]
        sort_command = " ".join(sort_command_list)
        sp.check_call(["/bin/sh", "-c", sort_command])
        os.unlink(infile)

    def bedgraph(infile,strandedness,outfolder,basename,normlib,pthreads,bedgraph_list):
        if strandedness==1:
            stranded_yesno= "Stranded"
            plus_bedgraph_unique=outfolder + "/" + basename + "Signal.Unique.str2.out.bg"
            minus_bedgraph_unique = outfolder + "/" + basename + "Signal.Unique.str1.out.bg"
            plus_bedgraph_multi=outfolder + "/" + basename + "Signal.UniqueMultiple.str2.out.bg"
            minus_bedgraph_multi = outfolder + "/" + basename + "Signal.UniqueMultiple.str1.out.bg"
        elif strandedness==2:
            stranded_yesno= "Stranded"
            plus_bedgraph_multi=outfolder + "/" + basename + "Signal.UniqueMultiple.str1.out.bg"
            minus_bedgraph_multi = outfolder + "/" + basename + "Signal.UniqueMultiple.str2.out.bg"
            plus_bedgraph_unique=outfolder + "/" + basename + "Signal.Unique.str1.out.bg"
            minus_bedgraph_unique = outfolder + "/" + basename + "Signal.Unique.str2.out.bg"
```

```
        else:
            stranded_yesno="Unstranded"
            bedgraph_unique = outfolder + "/" + basename + "Signal.Unique.str1.out.bg"
            bedgraph_multi = outfolder + "/" + basename + "Signal.UniqueMultiple.str1.out.bg"

        inputs = ["""--inputBAMfile""", infile]
        outputs = ["""--outWigType""", "bedGraph", """--outWigStrand""", stranded_yesno, """--
outFileNamePrefix""", outfolder + "/" + basename]

        if not normlib:
            normalization=["""--outWigNorm""", "None"]
        else:
            normalization=["""--outWigNorm""", "RPM"]
        STARcommand_list = ["STAR","""--runMode""","inputAlignmentsFromBAM","""--
runThreadN""",str(pthreads)] + inputs + outputs + normalization
        STARcommand=" ".join(STARcommand_list)
        sp.check_call(["/bin/sh", "-c", STARcommand])

        if strandedness !=0:
            sort_coord(plus_bedgraph_unique,outfolder + "/" + basename +
"_plus_unique.bedgraph",1,2)
            sort_coord(minus_bedgraph_unique,outfolder + "/" + basename +
"_minus_unique.bedgraph",1,2)
            sort_coord(plus_bedgraph_multi,outfolder + "/" + basename + "_plus_multi.bedgraph",1,2)
            sort_coord(minus_bedgraph_multi,outfolder + "/" + basename +
"_minus_multi.bedgraph",1,2)

            bedgraph_list += [outfolder + "/" + basename + "_plus_unique.bedgraph",outfolder + "/" +
basename + "_minus_unique.bedgraph",outfolder + "/" + basename +
"_plus_multi.bedgraph",outfolder + "/" + basename + "_minus_multi.bedgraph"]
        else:
            sort_coord(bedgraph_unique,outfolder + "/" + basename + "_unique.bedgraph",1,2)
            sort_coord(bedgraph_multi,outfolder + "/" + basename + "_multi.bedgraph",1,2)
            bedgraph_list += [outfolder + "/" + basename + "_unique.bedgraph",outfolder + "/" +
basename + "_multi.bedgraph"]

    def make_bigwig(chrominfo,bedgraph_list):
        for bedgraph in bedgraph_list:
            outfile=bedgraph + ".bw"
            igvcommand_list = ["bedGraphToBigWig",bedgraph, chrominfo,outfile]
            igvcommand=" ".join(igvcommand_list)
            sp.check_call(["/bin/sh", "-c", igvcommand])


    def main(**kwargs):

        ######## ARGUMENTS ############
        #check if already args is provided, i.e. main() is called from the top level script
        args = kwargs.get('args', None)
        if args is None: ## i.e. standalone script called from command line in normal way
```

```
        parser = argparse.ArgumentParser(description = """Makes unique and multi bedgraph
files""")
        parser._optionals.title = "Arguments"
        parser.add_argument("-f","--fetch_folder", help = "Folder location of outputs from SQuIRE
Fetch (optional, default = 'squire_fetch'",type = str, metavar = "<folder>",default="squire_fetch")
        parser.add_argument("-m","--map_folder", help = "Folder location of outputs from SQuIRE
Map (optional, default = 'squire_map')", type = str, metavar = "<folder>", default="squire_map")
        parser.add_argument("-o","--draw_folder", help = "Destination folder for output files
(optional; default='squire_draw')", type = str, metavar = "<folder>", default="squire_draw")
        parser.add_argument("-n","--name", help = "Basename for bam file (required if more than
one bam file in map_folder)", type = str, metavar = "<str>",default=False)
        parser.add_argument("-s","--strandedness", help = " '0' if unstranded, 1 if first-strand eg
Illumina Truseq, dUTP, NSR, NNSR, 2 if second-strand, eg Ligation, Standard
(optional,default=1)", type = int, metavar = "<int>", default = False)
        parser.add_argument("-b","--build", help = "UCSC designation for genome build, eg. 'hg38'
(required)", type=str, metavar = "<build>",default=False,required=True)
        parser.add_argument("-l","--normlib", help = "Normalize bedgraphs by library size
(optional; default=False)", action = "store_true", default = False)
        parser.add_argument("-p","--pthreads", help = "Launch <int> parallel threads(optional;
default='1')", type = int, metavar = "<int>", default=1)
        parser.add_argument("-v","--verbosity", help = "Want messages and runtime printed to
stderr (optional; default=False)", action = "store_true", default = False)

        args,extra_args = parser.parse_known_args()
    ########## I/O #########
      ###### ARGUMENTS ######
      fetch_folder=args.fetch_folder
      map_folder = args.map_folder
      outfolder=args.draw_folder
      basename = args.name
      verbosity=args.verbosity
      build=args.build
      pthreads = args.pthreads
      strandedness=args.strandedness
      normlib=args.normlib
      ######### START TIMING SCRIPT ############
      if verbosity:
        startTime = datetime.now()
        print("start time is:" + str(startTime) + '\n', file = sys.stderr)# Prints start time
        print(os.path.basename(__file__) + '\n', file = sys.stderr) #prints script name to std err
        print("Script Arguments" + '\n' + "==================", file = sys.stderr)
        args_dict = vars(args)
        for option,arg in args_dict.iteritems():
          print(str(option) + "=" + str(arg), file = sys.stderr) #prints all arguments to std err
        print("\n", file = sys.stderr)
      make_dir(outfolder)
      infile = find_file(map_folder,".bam",basename, 1)
      if not basename:
        basename = get_basename(infile)
```

```
    if verbosity:
        print("Making unique and total bedgraphs "+ str(datetime.now())  + "\n",file = sys.stderr)
    chrominfo = find_file(fetch_folder,"_chromInfo.txt",build,1)
    bedgraph_list=[]
    bedgraph(infile,strandedness,outfolder,basename,normlib,pthreads,bedgraph_list)
    if verbosity:
        print("Making unique and total bigwigs "+ str(datetime.now())  + "\n",file = sys.stderr)
    make_bigwig(chrominfo,bedgraph_list)
    ####### STOP TIMING SCRIPT ########################
    if verbosity:
        print("finished writing outputs at "+ str(datetime.now()) + "\n",file = sys.stderr)

        endTime = datetime.now()
        print('end time is: '+ str(endTime) + "\n", file = sys.stderr)
        print('it took: ' + str(endTime-startTime) + "\n", file = sys.stderr)

####################
if __name__ == "__main__":
    main()
```

Appendix H. SQuIRE Seek

```
#!/bin/env python

##################### MODULES ###################
from __future__ import print_function,division
import sys
import os
import errno # error code module
import os.path
from datetime import datetime
import argparse #module that passes command-line arguments into script
from pyfaidx import Fasta #Pyfasta module flattens fasta data without spaces or headers so fasta
doesn't need to be read into memory
import glob
import tempfile
import subprocess as sp
import re
from subprocess import Popen, PIPE,STDOUT


#########################################################
def make_dir(path):
    try:
        original_umask = os.umask(0)
        os.makedirs(path, 0770)
    except OSError as exception:
        if exception.errno != errno.EEXIST:
            raise
    finally:
        os.umask(original_umask)

def isempty(filepath):
    if os.path.getsize(filepath) == 0:
        raise Exception(filepath + " is empty")

def basename(filepath):
        filename = os.path.basename(filepath)
        filebase = os.path.splitext(filename)[0]
        return filebase

class bed(object):
    def __init__(self, line):
        self.chromosome = line[0] # chr = first tab/first in list
        self.start = int(line[1])
        self.end = int(line[2])
        self.name=line[3]
        self.score=float(line[4])
        self.strand = line[5]

class gtf(object):
```

```
        def __init__(self,line):
            self.chromosome = line[0] # chr = first tab/first in list
            self.source = (line[1])
            self.feature = (line[2])
            self.start=int(line[3])
            self.end = int(line[4])
            self.score=float(line[5])
            self.strand = str(line[6])
            self.frame = line[7]
            self.attributes = line[8]

    def main(**kwargs):

        ######## ARGUMENTS ############
        #check if already args is provided, i.e. main() is called from the top level script
        args = kwargs.get('args', None)
        if args is None: ## i.e. standalone script called from command line in normal way
            parser = argparse.ArgumentParser(description = "Retrieves sequences from chromosome
fasta files")
            parser._optionals.title = "Arguments"
            parser.add_argument("-i","--infile", help = """Repeat genomic coordinates, can be TE_ID,
bedfile, or gff (required)""", type=argparse.FileType('r'), metavar = "<file.bed>", required=True)
            parser.add_argument("-o","--outfile", help = """Repeat sequences output file (FASTA), can
use "-" for stdout (required)""", type = argparse.FileType('w'), metavar = "<file.fa>", required=True)
            parser.add_argument("-g","--genome", help = "Genome build's fasta chromosomes - .fa file
or .chromFa folder (required)", type = str, metavar="<file.fa or folder.chromFa>", required=True)
            parser.add_argument("-v","--verbosity", help = "Want messages and runtime printed to
stderr (optional; default=False)",  action = "store_true", default = False)

            args,extra_args = parser.parse_known_args()

        ########## PARSE ARGUMENTS  #########
        infile = args.infile
        outfile = args.outfile #if outfile not given, give basename of infile to outfile with .seq extension
        genome = args.genome
        verbosity=args.verbosity


        ######### START TIMING SCRIPT #############
        if verbosity:
            startTime = datetime.now()
            print("start time is:" + str(startTime) + '\n', file = sys.stderr)# Prints start time
            print(os.path.basename(__file__) + '\n', file = sys.stderr) #prints script name to std err
            print("Script Arguments" + '\n' + "==================", file = sys.stderr)
            args_dict = vars(args)
            for option,arg in args_dict.iteritems():
                print(str(option) + "=" + str(arg), file = sys.stderr) #prints all arguments to std err
            print("\n", file = sys.stderr)

        ####### CHECK ARGUMENTS AND SET DEFAULTS ###########
```

```
        ########### REFERENCES #################
        required_columns = 6 ###For checking if infile is BED format
        previous_chromosome=0 #This is needed to avoid reopening chromosome sequence files,
which would make the script run time a lot longer.

        if os.path.isfile(genome): #if genome is file
           chromosome_infile = Fasta(genome)

        ### START FOR LOOP ####

        for line in infile:
           line = line.rstrip() #removes white space at end of line
           if line.startswith("track"):
              continue

           line = line.split("\t")  # returns list of items that were separated by tab in original file

        #########CHECK FILE FORMAT ###########
           column_count = len(line)
           if column_count == 1:
              line=line.split("|")
              bedline=bed(line)
              chromosome = bedline.chromosome # chr = first tab/first in list
              repstart = bedline.start
              repstop = bedline.end
              name=bedline.name
              strand = bedline.strand
              score=bedline.score
              header = str(chromosome) + ":" + str(repstart) + "-" + str(repstop) + "/" + str(strand )+ "/"
+ str(name)

           elif column_count > 1:
              if re.match("\d+", line[1]):
                 bedline = bed(line)
                 chromosome = bedline.chromosome # chr = first tab/first in list
                 repstart = bedline.start
                 repstop = bedline.end
                 name=bedline.name
                 strand = bedline.strand
                 score=bedline.score
                 header = str(chromosome) + ":" + str(repstart) + "-" + str(repstop) + "/" + str(strand )+
"/" + str(name)
              else:
                 gtfline = gtf(chromosome)
                 chromosome = gtfline.chromosome # chr = first tab/first in list
                 repstart = gtf.start
                 repstop = gtfline.end
                 name=gtfline.feature
                 strand = gtfline.strand
                 score=bedline.score
```

```
                header = str(chromosome) + ":" + str(repstart) + "-" + str(repstop) + "/" + str(strand )+
"/" + str(name)


        ######## FETCH SEQUENCES ###############

        if (chromosome != previous_chromosome):  #only reopens new chromosome file if a new
chr is reached in coordinates file
            print("Opening " + chromosome + "file" + '\n',file=sys.stderr)
            previous_chromosome = chromosome
            chromstart=0
            if os.path.isdir(genome): #if genome is folder
                chrom_infile = genome + '/'  + chromosome + '.fa'
                chromosome_infile = Fasta(chrom_infile)


        plus_strand_sequence = chromosome_infile[chromosome][repstart:repstop]
        if strand == '-':
            desired_sequence = -(plus_strand_sequence) #if negative strand, give reverse complement
in fasta file
        else:
            desired_sequence = plus_strand_sequence

    #       fix_BED.writelines(str(chromosome) + "\t" + str(repstart) + "\t" + str(repstop) + "\t" +
str(TE_ID) + "\t" + str(score) + "\t" + str(strand) + "\t" + str(repstart) + "\t" + str(repstop) + "\t" +
str(RGB) + "\n")

        #FASTA id for each repeat sequence is first 6 columns of the BED file
        outfile.writelines('>' + header + '\n' + str(desired_sequence) + '\n')
    print("Finished writing " + str(outfile) + '\n',file=sys.stderr)
    if verbosity:
        print("Finished writing RepChr FASTA file" + "\n", file = sys.stderr)

    ###### I/O ###############
    infile.close()
    outfile.close()
    ###### STOP TIMING SCRIPT ######################
    if verbosity:
        print("finished writing: " + outfile.name + '\n', file = sys.stderr)

        endTime = datetime.now()

        print('end time is: '+ str(endTime) + '\n', file = sys.stderr)
        print('it took: ' + str(endTime-startTime) + '\n', file = sys.stderr)

##################
if __name__ == "__main__":
    main()
```

7. Bibliography

1. Lander ES, Linton LM, Birren B, Nusbaum C, Zody MC, Baldwin J, et al. Initial sequencing and analysis of the human genome. Nature. Macmillian Magazines Ltd.; 2001;409:860–921.

2. Kazazian HH. Mobile elements: drivers of genome evolution. Science. American Association for the Advancement of Science; 2004;303:1626–32.

3. Huang CRL, Burns KH, Boeke JD. Active Transposition in Genomes. Annu Rev Genet. Annual Reviews ; 2012;46:651–75.

4. Burns KH, Boeke JD. Human Transposon Tectonics. Cell. 2012;149:740–52.

5. Wicker T, Sabot F, Hua-Van A, Bennetzen JL, Capy P, Chalhoub B, et al. A unified classification system for eukaryotic transposable elements. Nat Rev Genet. Nature Publishing Group; 2007;8:973–82.

6. Beck CR, Garcia-Perez JL, Badge RM, Moran J V. LINE-1 elements in structural variation and disease. Annu Rev Genomics Hum Genet. NIH Public Access; 2011;12:187–215.

7. Deininger P. Alu elements: know the SINEs. Genome Biol. BioMed Central; 2011;12:236.

8. Hancks DC, Kazazian HH, Jr. SVA retrotransposons: Evolution and genetic instability. Semin Cancer Biol. NIH Public Access; 2010;20:234–45.

9. Stewart C, Kural D, Strömberg MP, Walker JA, Konkel MK, Stütz AM, et al. A comprehensive map of mobile element insertion polymorphisms in humans. Malik HS, editor. PLoS Genet. Public Library of Science; 2011;7:e1002236.

10. Abecasis GR, Auton A, Brooks LD, DePristo M a, Durbin RM, Handsaker RE, et al. An integrated map of genetic variation from 1,092 human genomes. Nature. 2012;491:56–65.

11. Faulkner GJ, Kimura Y, Daub CO, Wani S, Plessy C, Irvine KM, et al. The regulated retrotransposon transcriptome of mammalian cells. Nat Genet. Nature Publishing Group; 2009;41:563–71.

12. Medstrand P, van de Lagemaat LN, Mager DL. Retroelement Distributions in the Human Genome: Variations Associated With Age and Proximity to Genes. Genome Res. 2002;12:1483–95.

13. Jordan IK, Rogozin IB, Glazko G V, Koonin E V. Origin of a substantial fraction of human regulatory

sequences from transposable elements. Trends Genet. 2003;19:68–72.

14. de Souza FSJ, Franchini LF, Rubinstein M. Exaptation of transposable elements into novel cis-regulatory elements: is the evidence always strong? Mol Biol Evol. 2013;30:1239–51.

15. Xie M, Hong C, Zhang B, Lowdon RF, Xing X, Li D, et al. DNA hypomethylation within specific transposable element families associates with tissue-specific enhancer landscape. Nat Genet. Nature Publishing Group, a division of Macmillan Publishers Limited. All Rights Reserved.; 2013;45:836–41.

16. Huda A, Tyagi E, Mariño-Ramírez L, Bowen NJ, Jjingo D, Jordan IK. Prediction of transposable element derived enhancers using chromatin modification profiles. PLoS One. Public Library of Science; 2011;6:e27513.

17. Feschotte C. Transposable elements and the evolution of regulatory networks. Nat Rev Genet. Nature Publishing Group; 2008;9:397–405.

18. Chuong EB, Rumi MAK, Soares MJ, Baker JC. Endogenous retroviruses function as species-specific enhancer elements in the placenta. Nat Genet. NIH Public Access; 2013;45:325–9.

19. Chuong EB, Elde NC, Feschotte C. Regulatory evolution of innate immunity through co-option of endogenous retroviruses. Science. NIH Public Access; 2016;351:1083–7.

20. Trizzino M, Park Y, Holsbach-Beltrame M, Aracena K, Mika K, Caliskan M, et al. Transposable elements are the primary source of novelty in primate gene regulation. Genome Res. Cold Spring Harbor Laboratory Press; 2017;27:1623–33.

21. Chuong EB, Elde NC, Feschotte C. Regulatory activities of transposable elements: from conflicts to benefits. Nat Rev Genet. NIH Public Access; 2017;18:71–86.

22. Wang T, Zeng J, Lowe CB, Sellers RG, Salama SR, Yang M, et al. Species-specific endogenous retroviruses shape the transcriptional network of the human tumor suppressor protein p53. Proc Natl Acad Sci U S A. National Academy of Sciences; 2007;104:18613–8.

23. Gifford WD, Pfaff SL, Macfarlan TS. Transposable elements as genetic regulatory substrates in early development. Trends Cell Biol. NIH Public Access; 2013;23:218–26.

24. Wang J, Xie G, Singh M, Ghanbarian AT, Raskó T, Szvetnik A, et al. Primate-specific endogenous retrovirus-driven transcription defines naive-like stem cells. Nature. Nature Publishing Group; 2014;516:405–9.

25. Ecco G, Cassano M, Kauzlaric A, Duc J, Coluccio A, Offner S, et al. Transposable Elements and Their KRAB-ZFP Controllers Regulate Gene Expression in Adult Tissues. Dev Cell. Europe PMC Funders; 2016;36:611–23.

26. Imbeault M, Helleboid P-Y, Trono D. KRAB zinc-finger proteins contribute to the evolution of gene regulatory networks. Nature. Nature Publishing Group; 2017;543:550–4.

27. Wolf G, Yang P, Füchtbauer AC, Füchtbauer E-M, Silva AM, Park C, et al. The KRAB zinc finger protein ZFP809 is required to initiate epigenetic silencing of endogenous retroviruses. Genes Dev. Cold Spring Harbor Laboratory Press; 2015;29:538–54.

28. Jacobs FMJ, Greenberg D, Nguyen N, Haeussler M, Ewing AD, Katzman S, et al. An evolutionary arms race between KRAB zinc-finger genes ZNF91/93 and SVA/L1 retrotransposons. Nature. Nature Publishing Group; 2014;516:242–5.

29. Slotkin RK, Martienssen R. Transposable elements and the epigenetic regulation of the genome. Nat Rev Genet. 2007;8:272–85.

30. Belancio VP, Roy-Engel AM, Deininger PL. All y'all need to know 'bout retroelements in cancer. Semin Cancer Biol. Elsevier Ltd; 2010;20:200–10.

31. Burns KH. Transposable elements in cancer. Nat Rev Cancer. Nature Publishing Group; 2017;17:415–24.

32. Babaian A, Mager DL. Endogenous retroviral promoter exaptation in human cancer. Mob DNA. BioMed Central; 2016;7:24.

33. Muotri AR, Marchetto MCN, Coufal NG, Oefner R, Yeo G, Nakashima K, et al. L1 retrotransposition in neurons is modulated by MeCP2. Nature. Nature Publishing Group; 2010;468:443–6.

34. Li W, Jin Y, Prazak L, Hammell M, Dubnau J. Transposable Elements in TDP-43-Mediated Neurodegenerative Disorders. Iijima KM, editor. PLoS One. Public Library of Science; 2012;7:e44099.

35. Larsen PA, Hunnicutt KE, Larsen RJ, Yoder AD, Saunders AM. Warning SINEs: Alu elements, evolution of the human brain, and the spectrum of neurological disease. Chromosom Res. Springer Netherlands; 2018;26:93–111.

36. Larsen PA, Lutz MW, Hunnicutt KE, Mihovilovic M, Saunders AM, Yoder AD, et al. The Alu

neurodegeneration hypothesis: A primate-specific mechanism for neuronal transcription noise, mitochondrial dysfunction, and manifestation of neurodegenerative disease. Alzheimer's Dement. Elsevier; 2017;13:828–38.

37. Ambati J, Fowler BJ. Mechanisms of age-related macular degeneration. Neuron. NIH Public Access; 2012;75:26–39.

38. Newkirk SJ, Lee S, Grandi FC, Gaysinskaya V, Rosser JM, Vanden Berg N, et al. Intact piRNA pathway prevents L1 mobilization in male meiosis. Proc Natl Acad Sci U S A. National Academy of Sciences; 2017;114:E5635–44.

39. Brennecke J, Malone CD, Aravin AA, Sachidanandam R, Stark A, Hannon GJ. An epigenetic role for maternally inherited piRNAs in transposon silencing. Science. American Association for the Advancement of Science; 2008;322:1387–92.

40. Houwing S, Kamminga LM, Berezikov E, Cronembold D, Girard A, van den Elst H, et al. A Role for Piwi and piRNAs in Germ Cell Maintenance and Transposon Silencing in Zebrafish. Cell. Cell Press; 2007;129:69–82.

41. Malki S, van der Heijden GW, O'Donnell KA, Martin SL, Bortvin A. A Role for Retrotransposon LINE-1 in Fetal Oocyte Attrition in Mice. Dev Cell. 2014;29:521–33.

42. Giordano J, Ge Y, Gelfand Y, Abrusán G, Benson G, Warburton PE. Evolutionary history of mammalian transposons determined by genome-wide defragmentation. PLoS Comput Biol. Public Library of Science; 2007;3:e137.

43. Criscione SW, Zhang Y, Thompson W, Sedivy JM, Neretti N. Transcriptional landscape of repetitive elements in normal and cancer human cells. BMC Genomics. 2014;15:583.

44. Jin Y, Tam OH, Paniagua E, Hammell M. TEtranscripts: a package for including transposable elements in differential expression analysis of RNA-seq datasets. Bioinformatics. 2015;31:3593–9.

45. Lerat E, Fablet M, Modolo L, Lopez-Maestre H, Vieira C. TEtools facilitates big data expression analysis of transposable elements and reveals an antagonism between their activity and that of piRNA genes. Nucleic Acids Res. Oxford University Press; 2016;45:gkw953.

46. Philippe C, Vargas-Landin DB, Doucet AJ, van Essen D, Vera-Otarola J, Kuciak M, et al. Activation of

individual L1 retrotransposon instances is restricted to cell-type dependent permissive loci. Elife. eLife Sciences Publications Limited; 2016;5:e13926.

47. Deininger P, Morales ME, White TB, Baddoo M, Hedges DJ, Servant G, et al. A comprehensive approach to expression of L1 loci. Nucleic Acids Res. Oxford University Press; 2017;45:e31.

48. Scott EC, Gardner EJ, Masood A, Chuang NT, Vertino PM, Devine SE. A hot L1 retrotransposon evades somatic repression and initiates human colorectal cancer. Genome Res. Cold Spring Harbor Laboratory Press; 2016;26:745–55.

49. Kent WJ, Sugnet CW, Furey TS, Roskin KM, Pringle TH, Zahler AM, et al. The human genome browser at UCSC. Genome Res. 2002;12:996–1006.

50. Pruitt KD, Brown GR, Hiatt SM, Thibaud-Nissen F, Astashyn A, Ermolaeva O, et al. RefSeq: an update on mammalian reference sequences. Nucleic Acids Res. 2014;42:D756-63.

51. Smit, AFA, Hubley, R & Green P. RepeatMasker Open-4.0. 2013-2015 [Internet]. [cited 2018 Apr 21]. Available from: http://www.repeatmasker.org

52. Dobin A, Davis CA, Schlesinger F, Drenkow J, Zaleski C, Jha S, et al. STAR: ultrafast universal RNA-seq aligner. Bioinformatics. Oxford University Press; 2013;29:15–21.

53. Pertea M, Pertea GM, Antonescu CM, Chang T-C, Mendell JT, Salzberg SL. StringTie enables improved reconstruction of a transcriptome from RNA-seq reads. Nat Biotechnol. Nature Publishing Group; 2015;33:290–5.

54. Love MI, Huber W, Anders S. Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. Genome Biol. BioMed Central; 2014;15:550.

55. Huber W, Carey VJ, Gentleman R, Anders S, Carlson M, Carvalho BS, et al. Orchestrating high-throughput genomic analysis with Bioconductor. Nat Methods. Nature Publishing Group, a division of Macmillan Publishers Limited. All Rights Reserved.; 2015;12:115–21.

56. Robinson JT, Thorvaldsdóttir H, Winckler W, Guttman M, Lander ES, Getz G, et al. Integrative genomics viewer. Nat Biotechnol. Nature Publishing Group, a division of Macmillan Publishers Limited. All Rights Reserved.; 2011;29:24–6.

57. Li B, Ruotti V, Stewart RM, Thomson JA, Dewey CN. RNA-Seq gene expression estimation with read

mapping uncertainty. Bioinformatics. Oxford University Press; 2010;26:493–500.

58. Saito T, Rehmsmeier M. The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets. Brock G, editor. PLoS One. Public Library of Science; 2015;10:e0118432.

59. Perepelitsa-Belancio V, Deininger P. RNA truncation by premature polyadenylation attenuates human mobile element activity. Nat Genet. Nature Publishing Group; 2003;35:363–6.

60. Schwahn U, Lenzner S, Dong J, Feil S, Hinzmann B, van Duijnhoven G, et al. Positional cloning of the gene for X-linked retinitis pigmentosa 2. Nat Genet. Nature Publishing Group; 1998;19:327–32.

61. Kimberland ML, Divoky V, Prchal J, Schwahn U, Berger W, Kazazian HH. Full-Length Human L1 Insertions Retain the Capacity for High Frequency Retrotransposition in Cultured Cells. Hum Mol Genet. Oxford University Press; 1999;8:1557–60.

62. Smit AFA, Tóth G, Riggs AD, Jurka J. Ancestral, Mammalian-wide Subfamilies of LINE-1 Repetitive Sequences. J Mol Biol. Academic Press; 1995;246:401–17.

63. Boissinot S, Chevret P, Furano A V. L1 (LINE-1) Retrotransposon Evolution and Amplification in Recent Human History. Mol Biol Evol. Oxford University Press; 2000;17:915–28.

64. Lee J, Cordaux R, Han K, Wang J, Hedges DJ, Liang P, et al. Different evolutionary fates of recently integrated human and chimpanzee LINE-1 retrotransposons. Gene. NIH Public Access; 2007;390:18–27.

65. Upton KR, Gerhardt DJ, Jesuadian JS, Richardson SR, Sánchez-Luque FJ, Bodea GO, et al. Ubiquitous L1 mosaicism in hippocampal neurons. Cell. Elsevier; 2015;161:228–39.

66. Rodić N, Steranka JP, Makohon-Moore A, Moyer A, Shen P, Sharma R, et al. Retrotransposon insertions in the clonal evolution of pancreatic ductal adenocarcinoma. Nat Med. NIH Public Access; 2015;21:1060–4.

67. Iskow RC, McCabe MT, Mills RE, Torene S, Pittard WS, Neuwald AF, et al. Natural mutagenesis of human genomes by endogenous retrotransposons. Cell. Elsevier; 2010;141:1253–61.

68. Ewing AD, Kazazian HH, Jr. High-throughput sequencing reveals extensive variation in human-specific L1 content in individual human genomes. Genome Res. Cold Spring Harbor Laboratory Press; 2010;20:1262–70.

69. Gardner EJ, Lam VK, Harris DN, Chuang NT, Scott EC, Pittard WS, et al. The Mobile Element Locator

Tool (MELT): population-scale mobile element discovery and biology. Genome Res. Cold Spring Harbor Laboratory Press; 2017;27:1916–29.

70. Lee E, Iskow R, Yang L, Gokcumen O, Haseley P, Luquette LJ, et al. Landscape of somatic retrotransposition in human cancers. Science. American Association for the Advancement of Science; 2012;337:967–71.

71. Keane TM, Wong K, Adams DJ. RetroSeq: transposable element discovery from next-generation sequencing data. Bioinformatics. Oxford University Press; 2013;29:389–90.

72. Sudmant PH, Rausch T, Gardner EJ, Handsaker RE, Abyzov A, Huddleston J, et al. An integrated map of structural variation in 2,504 human genomes. Nature. Nature Publishing Group; 2015;526:75–81.

73. Ewing AD, Kazazian HH, Jr. Whole-genome resequencing allows detection of many rare LINE-1 insertion alleles in humans. Genome Res. Cold Spring Harbor Laboratory Press; 2011;21:985–90.

74. Brawand D, Soumillon M, Necsulea A, Julien P, Csárdi G, Harrigan P, et al. The evolution of gene expression levels in mammalian organs. Nature. Nature Publishing Group; 2011;478:343–8.

75. Gnanakkan VP, Jaffe AE, Dai L, Fu J, Wheelan SJ, Levitsky HI, et al. TE-array--a high throughput tool to study transposon transcription. BMC Genomics. BioMed Central; 2013;14:869.

76. Yue F, Cheng Y, Breschi A, Vierstra J, Wu W, Ryba T, et al. A comparative encyclopedia of DNA elements in the mouse genome. Nature. 2014;515:355–64.

77. Li B, Dewey CN. RSEM: accurate transcript quantification from RNA-Seq data with or without a reference genome. BMC Bioinformatics. BioMed Central; 2011;12:323.

78. Beck CR, Collier P, Macfarlane C, Malig M, Kidd JM, Eichler EE, et al. LINE-1 retrotransposition activity in human genomes. Cell. NIH Public Access; 2010;141:1159–70.

79. Mir AA, Philippe C, Cristofari G. euL1db: the European database of L1HS retrotransposon insertions in humans. Nucleic Acids Res. Oxford University Press; 2015;43:D43-7.

80. Wang J, Song L, Grover D, Azrak S, Batzer MA, Liang P. dbRIP: A highly integrated database of retrotransposon insertion polymorphisms in humans. Hum Mutat. 2006;27:323–9.

81. Payer LM, Steranka JP, Yang WR, Kryatova M, Medabalimi S, Ardeljan D, et al. Structural variants

caused by Alu insertions are associated with risks for many human diseases. Proc Natl Acad Sci U S A. National Academy of Sciences; 2017;114:E3984–92.

82. Brouha B, Schustak J, Badge RM, Lutz-Prigge S, Farley AH, Moran J V, et al. Hot L1s account for the bulk of retrotransposition in the human population. Proc Natl Acad Sci U S A. National Academy of Sciences; 2003;100:5280–5.

83. Tubio JMC, Li Y, Ju YS, Martincorena I, Cooke SL, Tojo M, et al. Mobile DNA in cancer. Extensive transduction of nonrepetitive DNA mediated by L1 retrotransposition in cancer genomes. Science. NIH Public Access; 2014;345:1251343.

84. Pitkänen E, Cajuso T, Katainen R, Kaasinen E, Välimäki N, Palin K, et al. Frequent L1 retrotranspositions originating from TTC28 in colorectal cancer. Oncotarget. Impact Journals, LLC; 2014;5:853–9.

85. Kalitsis P, Saffery R. Inherent promoter bidirectionality facilitates maintenance of sequence integrity and transcription of parasitic DNA in mammalian genomes. BMC Genomics. 2009;10:498.

86. Le TN, Miyazaki Y, Takuno S, Saze H. Epigenetic regulation of intragenic transposable elements impacts gene transcription in Arabidopsis thaliana. Nucleic Acids Res. 2015;43:3911–21.

87. Stower H. Alternative splicing: Regulating Alu element “exonization”. Nat Rev Genet. Nature Publishing Group, a division of Macmillan Publishers Limited. All Rights Reserved.; 2013;14:152–3.

88. Sorek R, Ast G, Graur D. Alu-containing exons are alternatively spliced. Genome Res. Cold Spring Harbor Laboratory Press; 2002;12:1060–7.

89. Athanasiadis A, Rich A, Maas S. Widespread A-to-I RNA editing of Alu-containing mRNAs in the human transcriptome. PLoS Biol. Public Library of Science; 2004;2:e391.

90. Quinlan AR, Hall IM. BEDTools: a flexible suite of utilities for comparing genomic features. Bioinformatics. 2010;26:841–2.

91. Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N, et al. The Sequence Alignment/Map format and SAMtools. Bioinformatics. Oxford University Press; 2009;25:2078–9.

92. R Development Core Team R. R: A Language and Environment for Statistical Computing. Team RDC, editor. R Found. Stat. Comput. R Foundation for Statistical Computing; 2011. p. 409.

93. Taylor MS, LaCava J, Mita P, Molloy KR, Huang CRL, Li D, et al. Affinity Proteomics Reveals Human Host Factors Implicated in Discrete Stages of LINE-1 Retrotransposition. Cell. 2013;155:1034–48.

94. Langmead B, Trapnell C, Pop M, Salzberg SL. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. Genome Biol. 2009;10:R25.

95. Langmead B, Salzberg SL. Fast gapped-read alignment with Bowtie 2. Nat Methods. 2012;9:357–9.

96. Rosenbloom KR, Armstrong J, Barber GP, Casper J, Clawson H, Diekhans M, et al. The UCSC Genome Browser database: 2015 update. Nucleic Acids Res. 2014;43:D670-81.

97. Kapusta A, Kronenberg Z, Lynch VJ, Zhuo X, Ramsay L, Bourque G, et al. Transposable elements are major contributors to the origin, diversification, and regulation of vertebrate long noncoding RNAs. Hoekstra HE, editor. PLoS Genet. Public Library of Science; 2013;9:e1003470.

98. Huda A, Bowen NJ, Conley AB, Jordan IK. Epigenetic regulation of transposable element derived human gene promoters. Gene. 2011;475:39–48.

99. van de Lagemaat LN, Landry J-R, Mager DL, Medstrand P. Transposable elements in mammals promote regulatory variation and diversification of genes with specialized functions. Trends Genet. Elsevier Current Trends; 2003;19:530–6.

100. Rebollo R, Romanish MT, Mager DL. Transposable elements: an abundant and natural source of regulatory sequences for host genes. Annu Rev Genet. Annual Reviews; 2012;46:21–42.

101. Sasaki T, Nishihara H, Hirakawa M, Fujimura K, Tanaka M, Kokubo N, et al. Possible involvement of SINEs in mammalian-specific brain formation. Proc Natl Acad Sci U S A. 2008;105:4220–5.

102. Johnson R, Guigo R. The RIDL hypothesis: transposable elements as functional domains of long noncoding RNAs. RNA. 2014;20:959–76.

103. Kaer K, Branovets J, Hallikma A, Nigumann P, Speek M. Intronic L1 Retrotransposons and Nested Genes Cause Transcriptional Interference by Inducing Intron Retention, Exonization and Cryptic Polyadenylation. Kanai A, editor. PLoS One. Public Library of Science; 2011;6:e26099.

104. Sorek R. The birth of new exons: mechanisms and evolutionary consequences. RNA. 2007;13:1603–8.

105. Vilborg A, Passarelli MC, Yario TA, Tycowski KT, Steitz JA. Widespread Inducible Transcription

Downstream of Human Genes. Mol Cell. 2015;59:449–61.

106. Oricchio E, Sciamanna I, Beraldi R, Tolstonog G V, Schumann GG, Spadafora C. Distinct roles for LINE-1 and HERV-K retroelements in cell proliferation, differentiation and tumor progression. Oncogene. 2007;26:4226–33.

107. Cordaux R, Batzer MA. The impact of retrotransposons on human genome evolution. Nat Rev Genet. Nature Publishing Group; 2009;10:691–703.

108. Kapitonov V V, Jurka J. A universal classification of eukaryotic transposable elements implemented in Repbase. Nat Rev Genet. 2008;9:411–412; author reply 414.

109. Sultan M, Amstislavskiy V, Risch T, Schuette M, Dökel S, Ralser M, et al. Influence of RNA extraction methods and library selection schemes on RNA-seq data. BMC Genomics. BioMed Central; 2014;15:675.

110. Chen J, Sun M, Kent WJ, Huang X, Xie H, Wang W, et al. Over 20% of human transcripts might form sense-antisense pairs. Nucleic Acids Res. 2004;32:4812–20.

111. Villegas VE, Zaphiropoulos PG. Neighboring gene regulation by antisense long non-coding RNAs. Int J Mol Sci. 2015;16:3251–66.

112. Hutchins AP, Pei D. Transposable elements at the center of the crossroads between embryogenesis, embryonic stem cells, reprogramming, and long non-coding RNAs. Sci Bull. 60:1722–33.

113. Elbarbary RA, Lucas BA, Maquat LE. Retrotransposons as regulators of gene expression. Science (80-). American Association for the Advancement of Science; 2016;351:aac7247-aac7247.

114. Volders P-J, Helsens K, Wang X, Menten B, Martens L, Gevaert K, et al. LNCipedia: a database for annotated human lncRNA transcript sequences and structures. Nucleic Acids Res. 2013;41:D246-51.

115. Liu C, Bai B, Skogerbø G, Cai L, Deng W, Zhang Y, et al. NONCODE: an integrated knowledge database of non-coding RNAs. Nucleic Acids Res. 2005;33:D112-5.

116. Erwin JA, Marchetto MC, Gage FH. Mobile DNA elements in the generation of diversity and complexity in the brain. Nat Rev Neurosci. Nature Publishing Group, a division of Macmillan Publishers Limited. All Rights Reserved.; 2014;15:497–506.

117. McCall MN, Kent OA, Yu J, Fox-Talbot K, Zaiman AL, Halushka MK. MicroRNA profiling of diverse

endothelial cell types. BMC Med Genomics. BioMed Central; 2011;4:78.

118. Xu J-C, Fan J, Wang X, Eacker SM, Kam T-I, Chen L, et al. Cultured networks of excitatory projection neurons and inhibitory interneurons for studying human cortical neurotoxicity. Sci Transl Med. 2016;8:333ra48.

119. Kriks S, Shim J-W, Piao J, Ganat YM, Wakeman DR, Xie Z, et al. Dopamine neurons derived from human ES cells efficiently engraft in animal models of Parkinson's disease. Nature. Nature Publishing Group, a division of Macmillan Publishers Limited. All Rights Reserved.; 2011;480:547–51.

120. Benjamini Y, Hochberg Y. Controlling the False Discovery Rate : A Practical and Powerful Approach to Multiple Testing Author (s): Yoav Benjamini and Yosef Hochberg Source : Journal of the Royal Statistical Society . Series B (Methodological), Vol . 57 , No . 1 Published by : J R Stat Soc Ser B. 1995;57:289–300.

121. Kaneko H, Dridi S, Tarallo V, Gelfand BD, Fowler BJ, Cho WG, et al. DICER1 deficit induces Alu RNA toxicity in age-related macular degeneration. Nature. 2011;471:325–30.

122. Hollister JD, Gaut BS. Epigenetic silencing of transposable elements: a trade-off between reduced transposition and deleterious effects on neighboring gene expression. Genome Res. Cold Spring Harbor Laboratory Press; 2009;19:1419–28.

123. Akers SN, Moysich K, Zhang W, Collamat Lai G, Miller A, Lele S, et al. LINE1 and Alu repetitive element DNA methylation in tumors and white blood cells from epithelial ovarian cancer patients. Gynecol Oncol. 2014;132:462–7.

124. Xie M, Hong C, Zhang B, Lowdon RF, Xing X, Li D, et al. DNA hypomethylation within specific transposable element families associates with tissue-specific enhancer landscape. Nat Genet. Nature Publishing Group; 2013;45:836–41.

125. Wang C, Zou J, Ma X, Wang E, Peng G. Mechanisms and implications of ADAR-mediated RNA editing in cancer. Cancer Lett. Elsevier; 2017;411:27–34.

126. Seplyarskiy VB, Soldatov RA, Popadin KY, Antonarakis SE, Bazykin GA, Nikolaev SI. APOBEC-induced mutations in human cancers are strongly enriched on the lagging DNA strand during replication. Genome Res. Cold Spring Harbor Laboratory Press; 2016;26:174–82.

127. Swahari V, Nakamura A, Deshmukh M. The paradox of dicer in cancer. Mol Cell Oncol. Taylor &

Francis; 2016;3:e1155006.

128. Goelz SE, Vogelstein B, Hamilton SR, Feinberg AP. Hypomethylation of DNA from benign and malignant human colon neoplasms. Science. American Association for the Advancement of Science; 1985;228:187–90.

129. Jeong H-H, Yalamanchili HK, Guo C, Shulman JM, Liu Z. An ultra-fast and scalable quantification pipeline for transposable elements from next generation sequencing data. Biocomput 2018. WORLD SCIENTIFIC; 2018. p. 168–79.

130. Grossman RL, Heath AP, Ferretti V, Varmus HE, Lowy DR, Kibbe WA, et al. Toward a Shared Vision for Cancer Genomic Data. N Engl J Med. Massachusetts Medical Society; 2016;375:1109–12.

131. Carithers LJ, Ardlie K, Barcus M, Branton PA, Britton A, Buia SA, et al. A Novel Approach to High-Quality Postmortem Tissue Procurement: The GTEx Project. Biopreserv Biobank. Mary Ann Liebert, Inc. 140 Huguenot Street, 3rd Floor New Rochelle, NY 10801 USA ; 2015;13:311–9.

132. Kitkumthorn N, Mutirangura A. Long interspersed nuclear element-1 hypomethylation in cancer: biology and clinical applications. Clin Epigenetics. 2011;2:315–30.

133. Suzuki K, Suzuki I, Leodolter A, Alonso S, Horiuchi S, Yamashita K, et al. Global DNA demethylation in gastrointestinal cancer is age dependent and precedes genomic damage. Cancer Cell. Cell Press; 2006;9:199–207.

134. RStudio Team. RStudio: Integrated Development for R. Boston, MA: RStudio, Inc.; 2016.

135. H. Wickham. ggplot2: Elegant Graphics for Data Analysis. New York: Springer-Verlag; 2009.

136. Cerbin S, Jiang N. Duplication of host genes by transposable elements. Curr Opin Genet Dev. Elsevier Current Trends; 2018;49:63–9.

137. Doucet TT, Kazazian HH, Jr. Long Interspersed Element Sequencing (L1-Seq): A Method to Identify Somatic LINE-1 Insertions in the Human Genome. Methods Mol Biol. NIH Public Access; 2016;1400:79–93.

138. Lu X, Sachs F, Ramsay L, Jacques P-É, Göke J, Bourque G, et al. The retrovirus HERVH is a long noncoding RNA required for human embryonic stem cell identity. Nat Struct Mol Biol. Nature Publishing Group, a division of Macmillan Publishers Limited. All Rights Reserved.; 2014;21:423–5.

139. Jursch T, Izsvák Z, Ivics Z. Regulation of DNA transposition by CpG methylation and chromatin structure in human cells. Mob DNA. 2013;4:15.

8. Curriculum Vitae

Born on July 21st, 1986 in Yuanlin, Taiwan

8.1 EDUCATION AND TRAINING

Science and Medicine

2002-2007 B.S.,B.S., *cum laude* University of Washington, Seattle, WA

Majors: Neurobiology and Biochemistry

Minor: Chemistry

2009-present MD-PhD program, Johns Hopkins University, Baltimore, MD

Workshops

- *2008* Neural-Immune Interactions in Health and Disease, Foundation for Advanced Education in the Sciences Graduate School, Rockville, MD
- *2012* Statistical Analysis for Genomic Data, Cold Spring Harbor Laboratory Courses and Meetings, Long Island, NY
- *2014-2015* Emerging Women's Leadership Program for Women Faculty, JHUSOM.

8.2 RESEARCH ACTIVITIES

Research Experience

- *October 2013 –present:* Laboratory Kathleen Burns, MD-PhD. Departments of Pathology and Oncology, Johns Hopkins School of Medicine, Baltimore, MD
- *Aug 2011-October 2013:* Laboratories of Hyam Levitsky, MD and Kathleen Burns, MD-PhD. Departments of Pathology and Oncology, Johns Hopkins School of Medicine, Baltimore, MD
- *June 2010- August 2010:* Laboratory of Janice Clements, PhD. Department of Molecular and Comparative Pathobiology, Johns Hopkins School of Medicine, Baltimore, MD
- *June 2008-May 2009*: Laboratory of Jack Tsao, M.D., D.Phil., Department of Neurology, Uniformed Services University of the Health Sciences, Bethesda, MD
- *May 2007-May 2008:* Laboratory of Huaibin Cai, Ph.D., Laboratory of Neurogenetics, Transgenics Unit, National Institute of Aging, National Institutes of Health, Bethesda, MD
- *June 2005-Mar 2007:* Laboratory of Thomas J. Montine, M.D., Ph.D.,Department of Pathology, Neuropathology division, University of Washington, Seattle, WA
- *June 2006-Aug 2006:* Laboratory of Eugene O. Major, Ph.D, Laboratory of Molecular Medicine and Neuroscience, National Institute of Neurological Disorders and Stroke, Bethesda, MD

Peer Reviewed Original Science Publications

1. **Yang W**, Woltjer RL, Sokal I, Pan C, Wang Y, Brodey M, Peskind ER, Leverenz JB, Zhang J, Perl DP, Galasko DR, Montine TJ. Quantitative proteomics identifies surfactant-resistant alpha-synuclein in cerebral cortex of Parkinsonism-dementia complex of Guam but not Alzheimer's disease or progressive supranuclear palsy. Am J Pathol. 2007 Sep; 171(3):993-1002.

2. Lai C, Lin X, Chandran J, Shim H, **Yang WJ**, Cai H. The G59S mutation in p150$^{(glued)}$ causes dysfunction of dynactin in mice. J Neurosci. 2007 Dec 19; 27(51):13982-90.

3. Wang L, Xie C**,** Greggio E, Parisiadou L, Shim H, Sun L, Chandran J, Lin X, Lai C, **Yang W**, Moore DJ, Dawson TM, Dawson VL, Chiosis G, Cookson MR, and Cai H (2008) The Chaperone Activity of Heat Shock Protein 90 is Critical for Maintaining the Stability of Leucine Rich Repeat Kinase 2. J Neurosci. 2008 Mar 26; 28(13):3384-3391.

4. Cai H, Shim H, Lai C, Xie C, Lin X, **Yang WJ** and Chandran J ALS2/Alsin Knockout Mice and Motor Neuron Diseases. Neurodegenerative Diseases 2008;5(6):359-66

5. Lin X, Parisiadou L, Gu XL, Wang L, Shim H, Sun L, Xie C, Long CX, **Yang WJ**, Ding J, Chen ZZ, Gallant PE, Tao-Cheng JH, Rudow G, Troncoso JC, Liu Z, Li Z, Cai H. Leucine-rich repeat kinase 2 regulates the progression of neuropathology induced by Parkinson's-disease-related mutant alpha-synuclein. Neuron. 2009 Dec 24; 64(6):807-27.

6. Monaco MC, Maric D, Bandeian A, Leibovitch E, **Yang W**, Major EO. Progenitor-derived oligodendrocyte culture system from human fetal brain. J Vis Exp. 2012 Dec 20;(70).

7. Pragathi Achanta; Jared Steranka; Zuojian Tang; Nemanja Rodić; Reema Sharma; **Wan Rou Yang**; Sisi Ma; Mark Grivainis; Cheng Ran Lisa Huang; Anna M Schneider; Gary L Gallia; Gregory J Riggins; Alfredo Quinones-Hinojosa; David Fenyö; Jef D Boeke; Kathleen H Burns. Somatic retrotransposition is infrequent in glioblastomas. Mobile DNA. 2016. **7**:22

8. A map of mobile DNA insertions in the NCI-60 human cancer cell panel. Zampella JG, Rodić N, **Yang WR**, Huang CR, Welch J, Gnanakkan VP, Cornish TC, Boeke JD, Burns KH. Mob DNA. 2016 Oct 31;7:20.

9. Payer LM, Steranka JP, **Yang WR**, Kryatova M, Medabalimi S, Ardeljan D, Liu C, Boeke JD, Avramopoulos D, Burns KH. Structural variants caused by *Alu* insertions are associated with risks for many human diseases.Proc Natl Acad Sci U S A. 2017 May 2. pii: 201704117. doi: 10.1073/pnas.1704117114.

10. **Yang Wan R**., Daniel Ardeljan, Clarissa N. Pacyna, Lindsay M. Payer, Kathleen H. Burns. SQuIRE Reveals Locus-specific Regulation of Interspersed Repeat Expression. *(in revision, Nucleic Acids Research)*

11. **Yang Wan R.**, Min-Sik Kim, Jin-Chong Xu, Manoj Kumar, Paul Schaughency, Daniel Ardeljan, Jane A. Welch, Lindsay M. Horvath, Srikanth S. Manda, Chunhong Liu, Jef D. Boeke, Sarah J. Wheelan, Valina L. Dawson, Ted M. Dawson, Marc K. Halushka, Akhilesh Pandey, Hyam I. Levitsky, Kathleen H. Burns. Landscape of Transposable Element Expression in Human Cells. *(submitted, Nature Genetics)*

Presentations

- Varying levels of JCV infection during differentiation of primary human progenitor-derived oligodendrocytes. Poster, National Institutes of Health Summer Research Program; Bethesda, Maryland; August 6, 2006.
- Characterization of a *DCTN1* Conditional Knockout Mouse Model. Poster. National Institutes of Health Spring Research Festival. Bethesda, MD. May 9, 2008.
- Visual representation of the history of immune privilege. Poster, Medical Student Research Day; Baltimore, MD, January 5, 2011.
- What Transposable Elements are Differentially Translated in Cancer? Poster, FASEB SRC Mobile DNA in Mammalian Genomes, June 2013
- "What Transposable Elements are Differentially Translated in Lung Cancer?" Poster, Society of Immunotherapy Conference, November 2013
- "Cancer Biology and the 'Junk' Genome." Speaker, Partnering Toward Discovery series, January 2015.
- "RepTag: Quantifying specific transposable element RNA expression in the genome". Poster, FASEB SRC Mobile DNA in Mammalian Genomes, June 2015
- "Landscape of Transposable Element Expression in Human Cells", Speaker, FASEB SRC Mobile DNA in Mammalian Genomes June 2017
- "Landscape of TE Expression in Cancer." Poster, The Mobile Genome: Genetic and Physiological Impacts of Transposable Elements Conference, October 2017

Inventions, Patents, Copyrights

- *2015-2018* Wan Rou Yang SQuIRE: Software for Quantifying Interspersed Repeat Expression (software)

Extramural Sponsorship (current, pending, previous)

- *8/01/16 – 7/30/18* Transposable Elements and Tumor Immunology
  JUNO Therapeutics, Inc.
  $241,087 (annual DC)
  Role: **PhD student**

  Goals of this sponsored research agreement are to catalog transposable elements that are expressed in cancers and explore their potential as immunotherapeutic targets.

- *9/15/13 – 9/15/15* Junk DNA-Encoded Antigens in Ovarian Cancer

  OC120390
  Department of Defense Congressionally Directed Medical Research Programs (CDMRP)
  $300,000 (total DC) and PhD support (Teal Scholar)
  Role: **PhD student**

8.3 EDUCATIONAL ACTIVITIES

Educational Publications

- Skarupski KA, Levine RB, **Yang WR**, González-Fernández M, Bodurtha J, Barone MA, Fivush B. Leadership Competencies: Do They Differ for Women and Under-Represented Minority Faculty Members? The Journal of Faculty Development, Volume 31, Number 1, 15 January 2017, pp. 49-56(8)

Editorials

- Bipasha Mukherjee-Clavin, Carolina Montaño, Neil M. Neumann and **Wan R. Yang.** "U.S. must restore biomedical research funding". Op-Ed, Baltimore Sun. Sept 17, 2013.

Presentations

- W.R. Yang, E.R. Shamir, E.B. Heikamp*, B.P. Keenan*, C. Montaño*, B. Mukherjee-Clavin*, M. Buntin, S.A. Welling, J.D. Siliciano, R.F. Siliciano. Gender Differences in the Career Outcomes of Johns Hopkins MD-PhD Program Graduates. American Physician Scientists Association Conference, May 2013.

Teaching

- *2015* Practical Genomics Workshop Teaching Assistant, Johns Hopkins University School of Medicine, Baltimore, MD

2007-2009 The Princeton Review, MCAT teacher

Mentoring

- *2015-2016* Angela Hu, undergraduate, Johns Hopkins University
- *2017 -2018* Clarissa N. Pacyna, undergraduate, Johns Hopkins University. Dean's Undergraduate Research Award

8.4 ORGANIZATIONAL ACTIVITIES

- *2011-2016* Co-chair of Professional Development Committee in the MD-PhD Student Advisory Board, December
- *2011-2016* Co-chair of Association of Women Student MD-PhDs student group

8.5 RECOGNITION

- *2002-2007* University of Washington's Dean's List
- *2003-2004* Alfred and Ruth Goddard Scholarship, for achievement in the biological sciences
- *2005* Phi Beta Kappa member
- *2005* Phi Lamda Upsilon member
- *2005-2006* Howard Hughes Research Internship Program for undergraduates
- *2006* Rex J. and Ruth C. Robinson Scholarship Fund in Chemistry
- *2013* APSA Travel Award for Joint ASCI/AAP/APSA Meeting
- *2013-2015* Teal Predoctoral Scholar award from the Department of Defense