A novel weight-driven ATN-based SQL sentence generator to accommodate AI-based Reinforcement Learning

1stChristopher Troy
CEPS
University of the West of Scotland
Paisley, Scotland
christopher.troy@uws.ac.uk

2ndQi Wang
CEPS
University of the West of Scotland
Paisley, Scotland
Qi.Wang@uws.ac.uk

3rdJose M. Alcaraz-Calero
CEPS
University of the West of Scotland
Paisley, Scotland
Jose.Alcaraz-Calero@uws.ac.uk

***Abstract*—This paper presents a novel approach for generating SQL queries through a weight-driven framework using a modified ATN of ANTLR4's runtime components. Our objective is to enhance ATN capabilities for SQL generation by incorporating the functionality to accommodate adaptive learning solutions. We successfully designed and implemented a system that assigns weights to ATN transitions, including token weight assignment when presented with multiple valid tokens to choose from whilst traversing set-transitions. These weights have interfaces for dynamic adjustments based on heuristics and user-defined strategies.Our methodology involves modifying ANTLR4's core components to include weight management and traversal algorithms. We leverage heuristics to guide weight adjustments, addressing loop structures and recursive depth control in a system controlled by weights. Additionally, we establish mechanisms for weight persistence and optimization. Experimental evaluation using a simplistic SQL grammar demonstrates the effectiveness of our approach. We observe that weights can steer the parsing process towards desired outcomes, and that convergence occurs as the exploration-exploitation balance is optimized through parameter tuning. This research lays the groundwork for integrating reinforcement learning with our weight-driven ATN system. This holds promise for tackling complex challenges in structured data analysis that might not be readily apparent through human inspection alone. While our current work primarily focuses on heuristics, future efforts will explore the next stage of our research to further enhance the decision-making capabilities of our framework using reinforcement learning.**

***Index Terms*—Automata, ATN, SQL, Generative AI, Weights, Heuristics, Antlr4**

I. Introduction

SQL Language is present virtually in almost any information system along the world. The usage of generative AI to build effective SQL sentences for a given purpose will be a powerful feature to allow AI to optimize the decision making process of any SQL-based software component. A way to achieve such generation of SQL sentences is using non-deterministic push down automata. However, these are normally associated with parsing requiring a form of input. Generating output when no input is given can raise questions as to how it is done. Our past research [1] was capable of accomplishing this task, as have others [2], [3].

The world has become inter-connected at the macro and micro levels through increasing IoT development and edge based compute solutions. Conversely, with an exponentially growing demand in such devices, data storage requirements evidently rise as well requiring more efficient databases capable of handling large amounts of structured data. Having a solution to automate SQL becomes much more critical, but only when true autonomy can be utilised. Such autonomy would provide solutions through adaptive learning, which we feel highlights the significance of exploring this method for generative AI.

Our research provides a means for expanding our knowledge and enabling AI based SQL generation. SQL is one of the most widely used languages for interacting with relational database management systems. A survey conducted by jet-brains [4] put SQL in 4th place at 52 percent usage in the last 12 months from respondents. This highlights an importance for exploring ways in which we can improve it . Additionally, our research when compared and contrasted against our main use case, it has potential to radically change the way in which we interact with a RDBMS (Relational Database Management System).

In doing so it presents problems, such as semantic ambiguity misalignment's, including issues surrounding recursive nesting which requires complex mitigation implementations to ensure a coherent output is generated. Ambiguity in the context of computational linguistics has been a universal obstacle which many parsers try to mitigate when computing languages. A sentence or statement can be expressed in various ways and be syntactically correct, yet semantically incorrect.

Different left to right parsing algorithms such as the top down LL(K) (leftmost derivation) algorithm, and bottom up LR(K) (rightmost derivation in reverse) where k represents a number of lookaheads. These algorithms are used to mitigate this problem using contextual decisions during parse tree construction. Using these techniques can aid in ascertaining the sequences better aligned with previously acquired context.

Parser generators such as ANTLR4 [5] build a special data structure known as an ATN (Augmented Transition Network).

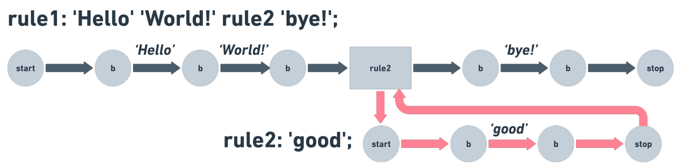


Fig. 1. Grammar represented in its ATN graph form

This is a state machine composed of sub-graphs containing nodes and edges. If using a context-dependant natural language type grammar, it could represent a common SVO (Subject-verb object) pattern which connect, forming the next part of the sequence in a sentence. However, we are going use SQL language. SQL is defined using a BNF grammar and as such it is a context-free grammar.

To keep it simplified for the sake of brevity, one can conceptualise a rule as defined in the grammar being comprised of many interconnecting nodes and rules with a beginning and an end. See Fig. 1 which demonstrates a simplistic grammar using a deterministic path. It highlights its ATN representation with an added depth of one (colored in red) which extends it visually past the first rule.

Using these rules and their pre-defined relationships, we can forcefully traverse them by visiting each node and sub-rules allowing us to build up the equivalent of an SQL statement through expected token acquisition. Up to this research phase, this process has been achieved using a stochastic process until reaching a terminating stop state. Although this approach may be capable of generating syntactically correct outputs, it does not always generate suitable semantically valid outputs. This is especially evident in the widely adopted Large Language Models, examples such as GPT3 [6] and GPT4 [7]. Having a means to fine tune this traversal process ensures there is a way to refine the guidance when acting on non-deterministic transitions

By removing the need to process input to generate new output, you are presented with some key benefits. We will highlight three positive factors which can be considered as gains to be capitalized on.

- **Adaptive Training data**: A zero human input approach in the context of our research provides a near unlimited training dataset of permutations with further generalized exploration capabilities.
- **State Space Reduction**: For generative purposes, EBNF grammars will positively contribute to a reduction of high-dimensional state spaces. This can be achieved through targeted playground environments where only sub-sets of the grammar are used to control the combinatorial explosion of the search space.
- **Fully Autonomous**: The lack of human input results in a more self serving system, i.e lack of supervised learning. Since there is no need for human input, we can focus on solutions with more autonomous functionality such as weight systems as AI-driven input and Reinforcement Learning integration.

Our research explores methods for implementing a novel weight-based system through the direct adaptation of the ANTLR4 parser generators ATN runtime components. This weight system allows us to research different ways in which weights can play a role in smarter guided decision-making outcomes through added heuristics within stochastic environments for the generation of SQL statements. Additionally, this permits us a further opportunity to explore ways to integrate AI functionality using Reinforcement Learning by means of a policy network. This can be achieved by directly mapping states to actions in a stochastic environment which is optimal for such an expansive language such as SQL. Usability can also be extended through its ability to generate not only SQL, but any language which is defined in an EBNF (Extended Backus Naur Form) format. In regards to our EBNF SQL grammar, this pertains to the syntactic structure of which constitutes a DQL (Data Query Language), DDL (Data Definition Language) and DML (Data Manipulation Language) statements. Having these pre-defined grammars provides us with syntactic knowledge beforehand, resulting in an expected reduction in the state space where syntax does not require a solution for correcting.

This novel approach can have many use cases applied. One such use case we are aiming to target resides within the scope of detecting network attacks hidden within large amounts of network flow information. This can be done by basing the rewards on the accuracy the statement has with the appropriate attack metrics using SQL functionality to select large amounts of data for analyses through novel metric generations. Furthermore, optimisations can be achieved through its ability to generate more efficient SQL statements reducing the computational overhead on an RDBMS (Relational Database Management System). This can enable us to provide solutions for innovative optimisation techniques going beyond traditional SQL statements when selecting and or modifying data more effectively. Our research aims to contribute to the body of knowledge through these innovations beyond the state of the art with these contributions:

- Design and implementation of a novel weight-driven ATN-based SQL sentence generation framework to enable AI optimisation through adaptive learning.
- Extend traditional ATN functionalities for accommodating AI use cases through ATN extensions.
- A novel framework for creating and utilising AI based Reinforcement Learning using our extended ANTLR4 component.

This manuscript is structured in this order. Section 2 looks into the current and previous research in augmenting automata using weight-based decisions for optimisations. Section 3 will discuss the methodology applied using a sequence of processes required to establish the interfaces needed for guiding generation traversal using weights. These will be broken down and expanded upon to explain concepts and our chosen implementations in greater detail with their corresponding justifications. Section 4 will highlight our results from validating the tools ability to generate SQL, with a focus on heuristics and logic functionality. Section 5 will go into our conclusions with section 6 acknowledging external funding which has allowed us to do this research.

II. Related Works

Implementing a weight system into automata is not something new, it is considered an optimisation solution for certain application use cases. Chatterjee et al [8] conducted research on probabilistic semantics associated with quantitative automata. Each transition would be assigned a rational number with each weight being aggregated into a single value for each traversal. What is interesting is the authors want to move away from asking if something will happen, to how likely it is to happen. This probabilistic lens helps push automata theory where complex systems can be understood through new methods of analysis. Another study with a similar focus by Jakub et al [9] has gone further into non-deterministic weighted automata and probabilistic semantics. The study looked at how the value of a word can be determined over the sum of weights and limit average over infinite runs. Results showed probabilities to be rather unpredictable at times and complex with some values being irrational and transcendental.

Manfred et al [10] introduced a new normal form for push-down automata, which is the same type of automata used in ANTLR4. The authors refer to it as a "Simple reset push-down automata" where limited access to the stack is applied using only 3 commands. Their results showed this type of weighted push-down automata is capable of recognizing context-free languages and is capable of generating algebraic power series. There can be many benefits to implementing weights into an automaton. One such study, albeit much older was by Hafner et al [11] which used weight-based finite automata for compression on images and video. Within this context, it was used to ascertain if a range block should be approximated or subdivided to optimise the encoding procedure by carefully making the correct decisions.

Despite many of these interesting approaches which showcase a variety of solutions whereby weighted automata can be beneficial, there still lacks much research around non-deterministic push-down automata with weight capabilities and AI.

III. Methodology

Much of the underpinning methodology of this research rests upon the notion that by implementing a novel weight system into ANTLR4 ATN runtime components, it will enhance its generative capabilities through heuristics and stochastic decision strategies. This objective is further explored by extending ways in which ATN usage within ANTLR4 can benefit from artificial intelligence through adapting its code for interfacing functionality. See, Fig.2, which helps conceptualise the overall methodology used to generate SQL and retain knowledge supported by our novel ANTLR4 ATN weight-system.

The top highlights an EBNF grammar rule to generate simple SQL queries. The bottom is a ATN graph to be traversed that maps such rule. The ATN in Fig.2 highlights the various node types and transitions. Some are weighted (*denoted in red*) as a result of their non-deterministic action space.

In the context of a full grammar as opposed to just one basic sample rule, all weights and the logic required to compute them underlines the scope of our novel weight systems capability. Furthermore, the actions taken, whether it is from token selection or transitions, all must be normalised. This normalisation ensures that when decisions are being made they are done so in a way in which probability can be utilised if required.

A. ATN Weight Architecture

ANTLR4's runtime includes the ATN package which contains all the components used to construct the graph from its grammar. ATN graphs are comprised of nodes, transitions, and rules. Each node type is unique in its behaviour and position within the ATN. The behaviour each node expresses will have a varying degree of influence on the weights, thus when opting for a heuristic based influence for optimisation, they must logically conform to the states behaviour when mapped to a numerical value or algorithm. This will directly influence weight optimisation unique to the state and all transitions belonging to it.

To enable this functionality we modified ANTLR4's codebase whilst ensuring all components maintained their original functionality. Fig 3 shows some of the newly included classes which are colored in grey, whereas those in green are our adaptations to the original class source code. The name of the methods is almost self-explanatory and will allow any interested reader to understand their purpose.

Additionally, we opted to have a duo purpose design to help demonstrate the weight system reacting to different environmental influences. This enabled us an opportunity to explore two methods of traversal, with the first focusing on optimisation through a heuristic based influence, whereas the second caters more towards API utilisation.

B. Weight implementation and management

Weight integration is achieved through implementing a new weight array within the "ATNState" class ensuring all nodes have this functionality, regardless of its type. Each ATNState can be considered a control room where much of the weight systems is dynamically adjusted. This makes sense, see Fig.4 which highlights its significance as the only one capable of accessing all possible transition actions at runtime. Furthermore, for RL (Reinforcement Learning) integration an immediate reward property can be used if non-episodic (epoch) weight optimisation is preferable.

One may ask why all node and transition types are targeted as opposed to only classes inherited from the "DecisionState" class. Firstly, the unique node types are used as indicators during traversal which can make use of further heuristics. The weight system is universal throughout the ATN sub-graphs. Lastly, our justification for weighting all transition types is

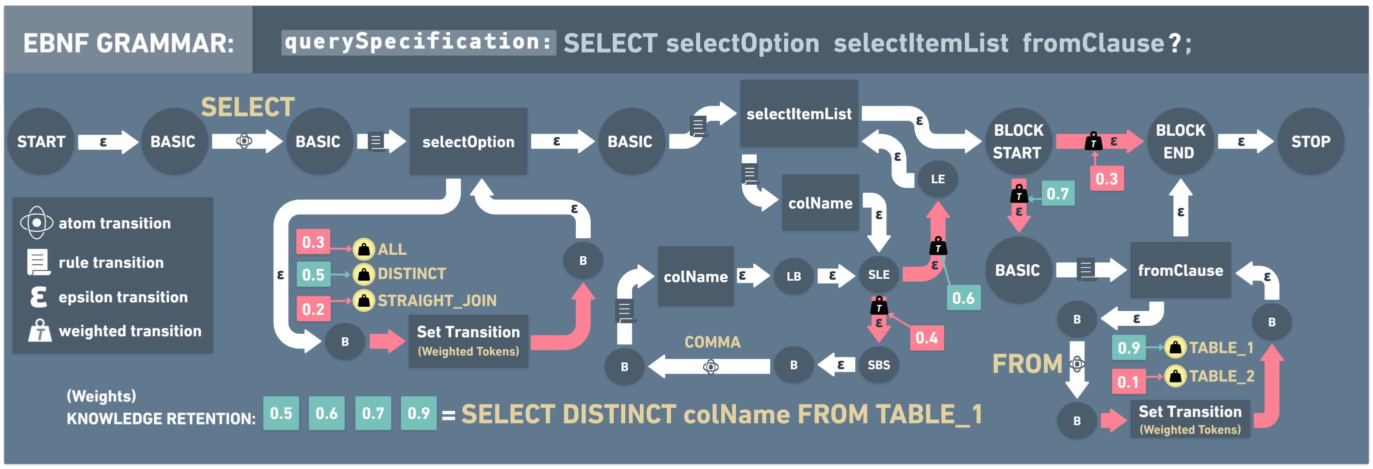


Fig. 2. Weight based generation system methodology

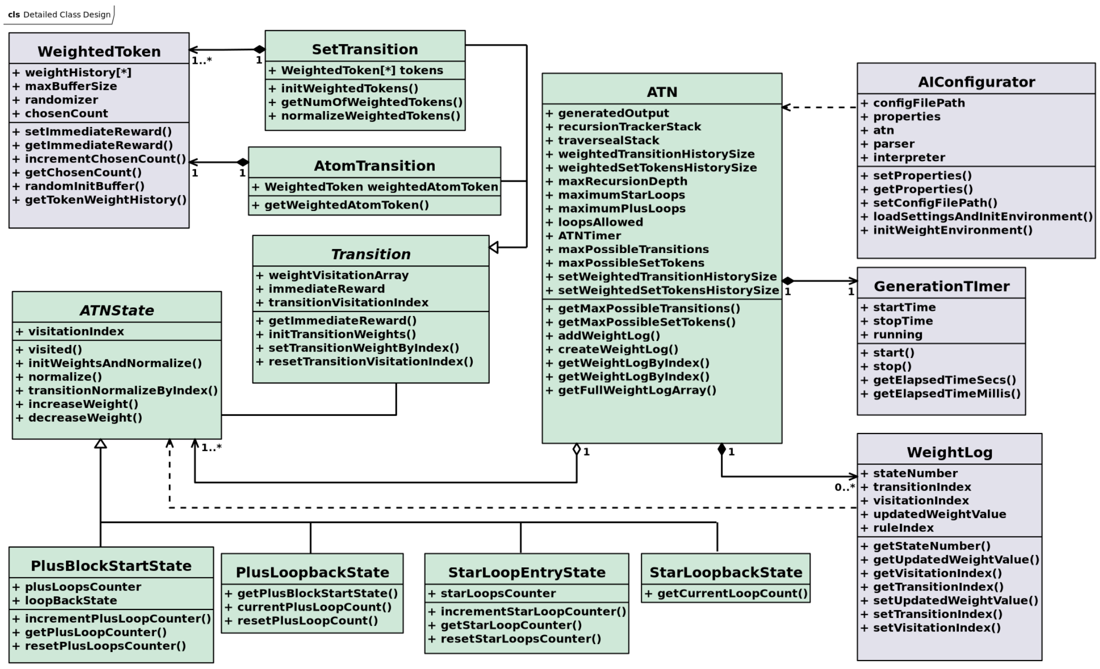


Fig. 3. ATN additional classes and adaptations

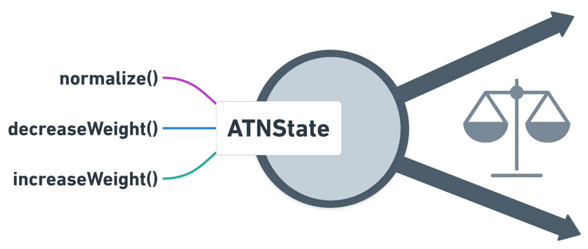


Fig. 4. ATNState modification with new weight methods

a usability decision for extending its capabilities when using AI. The weight system can be used independently without AI integration using external methods for optimising the weights. However, we are aligning both as closely as possible. With reinforcement learning integration we felt the full environment should be captured allowing us to modify the Transition class of ANTLR4's runtime which ensures all transitions contain such a property. It is only during weight initialization that any deterministic state transitions will have all weight values set to a value (e.g. 1.0) through its associated state object. This ensures we do not have any sparsity in the weights logged from start to finish.

.

Our justification for having a fixed size array of weights for each transition can be considered a functional requirement for allowing AI techniques that impose fixed input sizes as well as for limiting the memory footprint request. This size is a parameter that can be modified in the time where the ATN is being generated thus does not impose any hard limitation to the proposed architecture. As mentioned, a state can be visited multiple times, thus having an array of weight histories per transition is essential. It also serves as a solution for a fixed input policy network.

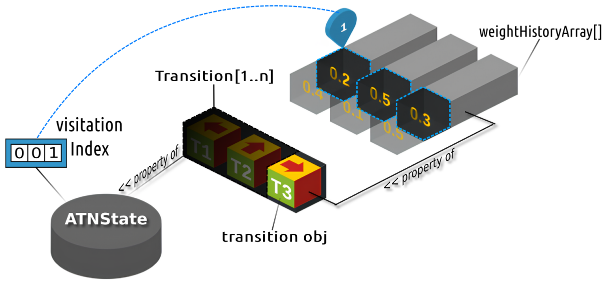
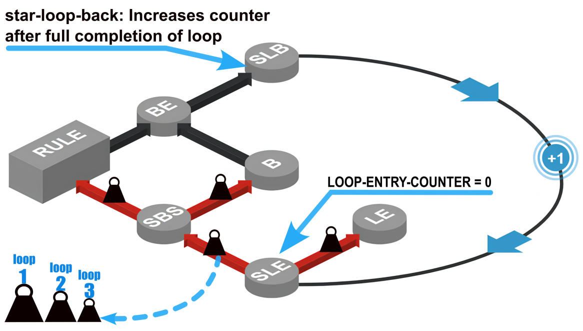


Fig. 5. ATNState modification with new weight methods

This fixed size can be determined at an earlier stage with our API (Application Programming Interface). Maintaining an accurate visitation count required an additional property within the ATN state object called a **visitation index**, see Fig.5. This counter serves as a heuristic which provides a few benefits. This is where every visitation increments the counter by one. How this relates to the transitions of the ATN state object and its weight history array is through the counters value at the point of visitation. This dictates the index of the arrays weights of each transition ensuring they all align for that particular visitation. It can also be used to track a change in weight values over time within the same SQL sentence generation.

C. Heuristic aid for loop structures

The use of certain notation with grammars such as kleene star/plus dictate the repetition of a sequence within the SQL grammar. This sequence presents itself as a loop. We opted for utilising a counter property, see here Fig.6, which is unique to the object which tracks the iteration count. The difference between both star-loop-entry and plus-block-start states is the point in which a decision is made. The higher



star-loop-back: Increases counter after full completion of loop
SLB
BE
RULE
B
+1
LOOP-ENTRY-COUNTER = 0
SBS
LE
SLE
loop 1
loop 2
loop 3

Fig. 6. Star Loop Heuristic Process

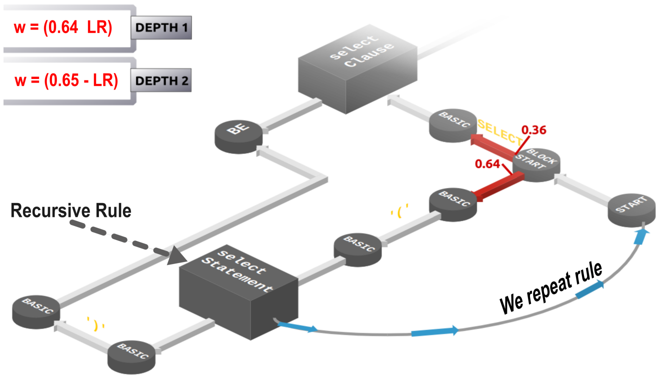


Fig. 7. Recursion influence on weights

the loop count goes, reductions in that weight is calculated to avoid its overuse.

D. Heuristic Depth influence on recursive traversal

Recursion can offer powerful capabilities for generating more complex SQL. This is especially true in the way SQL can nest its own statements. However, unless there is a means to control the recursion, it can easily become problematic and fail to complete. See Fig.7 which helps illustrate how recursive depth can influence the weights. As depth increases, complexity grows, therefore anything more than a depth of 2 results in the weight being reduced. This is applied repeatedly with the aim to make that transition into a recursive rule less appealing. Consequentially, the other weight will increase, thus ensuring we discourage bad choices while steering towards better ones.

E. Weight Persistence

We serialize all weights including sub-classes acting as composites to the ATNState. Firstly, we instantiated a new class type which very closely mirrors the ATNState structure, but differs in how weights are stored making it easier for us to manage. Secondly, we instantiate a new serialized transition class associated to our serialized state. Lastly, we check which type of transition it is, as this dictates whether or not we need to factor in a weighted token, or if a set, multiple weighted tokens. As a result, all the weigths are persistent in a file and we are able to store and re-load them into the ATNs.

$$W = [W1, W2, W3, Wn]$$
$$actionChosen' = min(max(^{w}actionChosen + \delta, 0), 1)$$
$$adjustment = {}^{w'}actionChosen -^{w} actionChosen$$
$$adjustmentPerWeight = \frac{adjustment}{n-1}$$
$$w_i = min(max(w_i - adjustmentPerWeight, 0), 1)\ for\, all\, i\ \neq actionChosen$$

Fig. 8. Expressions used for calculating weight adjustment

F. Weight optimisation through direct updates

The process for updating weights is multi-step. Any delta value added to the weights will impact the other weights available in the same transition or node by affecting the normalization process. How? The weights will redistribute the difference and then re-normalize. See Fig.8 and let **"W"** represent our weights. Let **"actionChosen"** represent the index which points to the weighted value within **"W"**. We must add our delta to the value while ensuring we do not go above 1 and below 0, hence the use of the max function from within the min function. Let **"adjustment"** represent the difference between the updated value and the original value. This is important for factoring in redistribution. Let **"adjustmentPerWeight"** represent the calculated adjustments per weight to compensate for the change. Finally, let **"Wi"** represent the adjustment made to each weight while ensuring the new value does not go below 0 or above 1.

G. Weight Traversal Process

Algorithm 1: Heuristic Traveler Navigation

```
Data: startState, learningRate, epsilon, minEpsilon,
      decayRate
Result: Path of the transverse through states using
        heuristic approach
/* Instantiate HeuristicTraveler */ $traveler \leftarrow$
  HeuristicTraveler(startState, learningRate, epsilon,
  minEpsilon, decayRate)
while true do
    /* Get current state */
    $currentState \leftarrow traveler.getOnState()$
    /* Move onto transition from current state */
    $traveler.moveOntoTransitionFrom(currentState)$
    /* Get target state after transition */
    $targetState \leftarrow traveler.moveTotargetState()$
    /* Set new state as the current state */
    $traveler.setOnState(targetState)$
```

Our Heuristic Traversal handles much of the systems decision making. See Alg.1 which provides the pseudo code for the main loop. Please be aware that a lot of abstraction is used when each of these methods are called. The "HeuristicTraveler" class is instantiated which expects 5 arguments. These arguments are:

1) (Start State) - A numeric value which instructs our system the state we will begin traversing from.
2) (Learning Rate LR) - A float value which dictates the amount of increase and decrease when optimising a weight.
3) (Epsilon) - This is another float value which is used to balance exploration vs exploitation.
4) (minEpsilon) - This is a float value which sets the minimum value epsilon can reach.
5) (decayRate) - The decay rate is a float value which gets factored into our weight optimisation calculation.

IV. Results

The specifications for our test-bed can be seen in Table III. Although the specifications are somewhat high in terms of performance, they do not indicate the requirement needed for computing the tests. GPU usage did not factor into our test from a lack of AI integration. We instead opted to first approach the heuristic influence on transition selection and weight adjustment. The operating system used was Windows 10. The most significant column in Table III is the "lang" column, which indicates the primary language used to implement our modifications and additions to ANTLR4's codebase. We used openjdk-17 within the integrated development environment "intellij". We implemented these into the newest release of ANTLR4s source code.

For the test results please see Table I. The first four columns from right to left were the parameters chosen before traversing the weighted ATN. Our time column represents time taken to generate the SQL within a second. The epsilon stop column provides us the value epsilon was at when we stopped. This became useful as it allowed us to ascertain if the minimum epsilon constraint was met, including an insight into the balance of exploration vs exploitation. The last column provides the output SQL generated from traversing the weighted ATN. The main intent of our experiment was to observe how the generated SQL changed over time as the epsilon value decreased. A high epsilon value is typically used to encourage more exploration. This simply equates to a greater amount of random choices being made at each decision state until convergence is reached. This convergence is a result of the decay rate reducing the epsilon value over time. This results in random decisions becoming less favourable, whilst opting for an exploitable based approach instead during transition selection. Our exploitation is implemented by simply choosing the weight with the highest value.

For testing purposes we opted to use a simplistic SQL grammar. This provided us with more control and capabilities for altering the grammar if needed. Each row indicates a full run from start to finish.

The results show that using weights can indeed be an effective method for generating SQL. What is more revealing, is that once minimum epsilon was reached, the weights of the transitions being chosen become more solidified, meaning the heuristics and strategy we deployed does have an observable impact. This signifies that using this approach when epsilon minimum is reached, and if "col1" was chosen, then that would be the preferred transition the token is associated with for the rest of the visits. This is the result of it containing the weight with the maximum value.

To ensure that normalization was working as intended, we used some of our adapted code to get the ATN to return the states and transitions chosen which produced the generated output. See Table II for reference. Of the first row and first column, it shows all encountered weights with a value of 1.0. This validates our logic when dealing with deterministic paths where you only have one choice, thus the maximum weight is applied during initialisation. Most will be epsilon transitions. However, those in column two which are inside an array highlight the transition weights which were made available at that point during traversal. From this we summed

TABLE I
EXPLORATION VS EXPLOITATION

LR	Epsilon	Min-Epsilon	Decay Rate	Time (s)	Epsilon Stop	SQL Query
0.001	1.0	0.01	0.99	0.048	0.8863	`SELECT col3, col2 FROM TABLE2 WHERE col3 < col2;`
0.001	0.3	0.01	0.99	0.050	0.2852	`SELECT col1 FROM TABLE3;`
0.001	0.7	0.01	0.99	0.048	0.6394	`SELECT col3 FROM TABLE3 WHERE col1 = "STRING";`
0.001	0.3	0.01	0.45	0.044	0.01	`SELECT * FROM TABLE1 WHERE col1 = "STRING";`
0.001	0.5	0.01	0.24	0.041	0.01	`SELECT col1, col1, col1, col1, col1 FROM TABLE1 WHERE col1 = col1;`
0.001	0.9	0.01	0.80	0.046	0.150	`SELECT col2, col1 FROM TABLE1;`
0.001	0.7	0.01	0.40	0.041	0.01	`SELECT col3 FROM TABLE1 WHERE col1 = "STRING";`
0.001	0.9	0.01	0.10	0.046	0.01	`SELECT col1, col1, col1, col1, col1, col1 FROM TABLE1;`
0.001	1.0	0.01	0.10	0.048	0.01	`SELECT col3 FROM TABLE1 WHERE col1 = col1;`

TABLE II
NORMALIZATION CHECK

State (S)	Transition Weights (T)	Sum_T	Is_Normalized
0, 22, 2, 25, 26, 4, 36, 6, 39, 8, 47, 48, 43, 46, 45, 7, 38, 5, 27, 28, 10, 49, 50, 11, 29, 30, 12, 51, 14, 53, 54, 16, 57, 58, 17, 55, 18, 60, 19, 56, 15, 52, 13, 32, 33, 34, 3, 23, 24	[1.0] EPSILON	1	Yes
37	[0.43, 0.57]	1	Yes
9	[0.24, 0.06, 0.33, 0.37]	1	Yes
44	[0.16, 0.84]	1	Yes
9	[0.36, 0.24, 0.21, 0.18]	1	Yes
44	[0.59, 0.41]	1	Yes
31	[0.80, 0.20]	1	Yes
9	[0.38, 0.20, 0.19, 0.23]	1	Yes
61	[0.15, 0.85]	1	Yes
9	[0.09, 0.03, 0.68, 0.20]	1	Yes
1	[]	0	No
GENERATED: "SELECT col2, col1 FROM TABLE2 WHERE col2 $<$ col1;"			

TABLE III
TEST-BED USED FOR EXPERIMENT

CPU	GPU	RAM	OS	IDE	Lang	ANTLR-4
intel 9900k	Nvidia RTX 3090	32GB	Windows 10	IntelliJ	Java	4.13.1

TABLE IV
VALIDATING SET TOKEN NORMALIZATION

Token	[0]	[1]	[2]	[3]	[4]
'STRING'	0.9902	0.6024	0.8054	0.1771	0.5571
'NUMBER'	0.0098	0.3976	0.1946	0.8229	0.4430

the weights to ensure they did not go beyond 1. The third column denotes the success in green. The last row and reason we see No in the last column in red is because State 1 is the stop state and an end has been reached.

Another validation check we wanted to ensure worked as intended was with our weighted tokens. These apply the same type of logic transitions use for normalization. We collected this data through the serialized JSON output and picked a object in the JSON array. See Table IV which can be represented as a set transition. The numbered columns represent the index position of that tokens array. We align the indexes for repeated usage, thus its imperative the values are aligned and normalized correctly. Our sample demonstrated the validation of our logic which shows correct normalization with their aligned indexes.

V. CONCLUSIONS

The findings from our tests were positive highlighting the ATN with weight capabilities and added heuristics can indeed be used as a method for decision making when generating SQL. We also validated our modifications to ANTLR4's source code, showing correct weight normalization and alignment with their corresponding weight indexes.

We do recognise the limitations of our tool at the present surrounding the semantic issue. We feel that semantics will always be a persistent problem if opting for a heuristic only based approach. This would inevitably require greater study to ascertain the most effective heuristics to use, including where, when and how they influence the weights during traversal of the ATN.

This raises an important aspect of our research surrounding a system which can enable reinforcement learning. With our uniquely designed and modified version of ANTLR4, we can now begin the process of integrating a full reinforcement learning solution for decision making. The knowledge gained throughout this research strengthens our believe that a form of Reinforcement Learning integration with our novel weight system can go beyond providing efficient SQL, but instead may solve complex issues hidden within large amounts of structured data not easily perceived through human analyses. We also acknowledge the subject area and the difficulty in truly conceptualizing the inner workings of the full system. Because of this, we have created detailed 3d visualisations to help better contextualise our research, see here [12].

VI. Acknowledgements

This research was funded by the European Commission under Grant Agreement HORIZON-JU-SNS-2022-STREAM-B-01-04/101095933 (RIGOUROUS: secure design and deployment of trusthworthy continuum computing 6G Services).

This work is funded in part by the European Commission under Grant Agreements H2020-SU-DS-2018-2019-2020/101020259 (ARCADIAN-IoT: Autonomous Trust, Security and Privacy Management Framework for IoT)

This work is funded in part by the Carnegie Trust for the Universities of Scotland: PHD010695.

References

[1] N. Nascimento, C. Tavares, P. Alencar, and D. Cowan, “Gpt in data science: A practical exploration of model selection,” in *2023 IEEE International Conference on Big Data (BigData)*, 2023, pp. 4325–4334.

[2] S. Sargsyan, J. Hakobyan, M. Mehrabyan, R. Mkoyan, V. Sahakyan, V. Melkonyan, M. Arutunian, A. Fahradyan, and A. Avetisyan, “Advanced grammar-based fuzzing,” in *2022 Ivannikov Memorial Workshop (IVMEM)*, 2022, pp. 61–64.

[3] S. Sargsyan, S. Kurmangaleev, M. Mehrabyan, M. Mishechkin, T. Ghukasyan, and S. Asryan, “Grammar-based fuzzing,” in *2018 Ivannikov Memorial Workshop (IVMEM)*, 2018, pp. 32–35.

[4] Jetbrains, “The state of developer ecosystem 2023,” 2023, accessed on May 16, 2024. [Online]. Available: https://www.jetbrains.com/lp/devecosystem-2023/

[5] T. Parr, S. Harwell, and K. Fisher, “Adaptive ll(*) parsing: The power of dynamic analysis,” in *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages and Applications*, ser. OOPSLA ’14. New York, NY, USA: Association for Computing Machinery, 2014, p. 579–598. [Online]. Available: https://doi.org/10.1145/2660193.2660202

[6] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, “Language models are few-shot learners,” 2020.

[7] OpenAI, “Gpt-4 technical report,” 2024.

[8] K. Chatterjee, T. A. Henzinger, and J. Otop, “Quantitative automata under probabilistic semantics,” in *2016 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 2016, pp. 1–10.

[9] J. Michaliszyn and J. Otop, “Non-deterministic weighted automata evaluated over markov chains,” *Journal of Computer and System Sciences*, vol. 108, pp. 118–136, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0022000019300741

[10] M. Droste, S. Dziadek, and W. Kuich, “Weighted simple reset pushdown automata,” *Theoretical Computer Science*, vol. 777, pp. 252–259, 2019, in memory of Maurice Nivat, a founding father of Theoretical Computer Science - Part I. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0304397519300337

[11] U. Hafner, “Image and video coding with weighted finite automata,” in *Proceedings of International Conference on Image Processing*, vol. 1, 1997, pp. 326–329 vol.1.

[12] ScottishCoder, “Cits 2024 weighted atn visualisations,” https://github.com/ScottishCoder/CITS-2024-Weighted-ATN-Visualisations.git, 2024, accessed: 2024-06-24.