UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Fine Tuning Transformer Models for Domain Specific Feature Extraction

Bachelor Thesis

Carla Campàs Gené

Director: Joaquim Motger de la Encarnacion
Co-director: Xavier Franch Gutiérrez
GEP Tutor: Pinto Paola Lorenza

18th January 2023

Fine Tuning Transformer Models for Domain Specific Feature Extraction

Carla Campàs Gené

Abstract

The nature of Natural Language Processing has drastically changed in the past years. The implementation of Large Language Models pre-trained on thousands of unlabelled data has opened the door to a new layer of comprehension of text processing. This has shifted research in the area to exploit these large models to obtain better results for smaller tasks.

In this way, fine-tuning Natural Language Processing is becoming increasingly important. By fine-tuning the different large language models with context and task-specific data, these models quickly learn to track patterns and generalize to new concepts. They understand natural language to a great extent and can generate relationships in words, phrases, and paragraphs.

Fine Tuning has become an increasingly important task to simplify the use of machine learning solutions with low resources. The increase in pre-trained transformer models for Natural Language Processing has complicated the selection and experimentation of these models, increasing research and experimentation time.

This study goes through the current state of the art of transformer models and attempts to study the scope and applicability of these models. From this initial work, the paper produces a comprehensive pipeline of model fine-tuning that allows the user to easily obtain a ready-to-use model for a natural language task. To test this model, the pipeline is tested and evaluated for the automatic extraction of features (i.e. functionalities) from mobile applications using available natural language documents, such as descriptions.

Afinar Models Basats en Transformadors per a l'Extracció de Característiques Específiques a un Domini

Carla Campàs Gené

Abstract

La naturalesa del processament de llengües naturals ha canviat dràsticament en els últims anys. La implementació de *Large Language Models* pre-entrenat en milers de dades sense etiquetar ha obert la porta a una nova capa de comprensió del processament de text. Això ha desplaçat la investigació a la zona per explotar aquests grans models per obtenir millors resultats per a les tasques més petites.

D'aquesta manera, el processament de llengües naturals està adquirint una importància cada vegada major. Afinant els diferents models de llenguatge gran amb dades específiques de context i de tasques, aquests models ràpidament aprenen a seguir patrons i generalitzar-los a nous conceptes. Entenen el llenguatge natural en gran mesura i poden generar relacions en paraules, frases i paràgrafs.

La sintonització fina neuronal s'ha convertit en una tasca cada vegada més important per simplificar l'ús de solucions d'aprenentatge automàtic amb pocs recursos. L'augment dels models de transformadors pre-entrenats per al processament del llenguatge natural ha complicat la selecció i l'experimentació d'aquests models, augmentant el temps de recerca i experimentació.

Aquest estudi passa per l'estat actual de l'art dels models transformadors i intenta estudiar l'abast i l'aplicabilitat d'aquests models. A partir d'aquest treball inicial, el document produeix un gasoducte complet d'ajust fi del model que permet a l'usuari obtenir fàcilment un model llest per a utilitzar per a una tasca de llenguatge natural. Per provar aquest model, la canonada es prova i s'avalua per a l'extracció automàtica de característiques (és a dir, funcionalitats) des d'aplicacions mòbils utilitzant documents de llenguatge natural disponibles, com ara descripcions.

Contents

List of Figures

List of Tables

1 Context and Scope

1.1 Introduction and Contextualization

Natural Language Processing (NLP) is found in the intersection of Linguistics and Computer Science. Natural Language Processing (NLP) is the study of artificial intelligence in Computer Science, giving computers comprehension to understand the text and spoken words in a similar form to humans [12]. In other words, NLP can be defined as the field that intends to decipher the connection between computers and language by analyzing large amounts of natural language data and extracting conclusions from those.

The field of NLP has swiftly evolved to atone for new and improved computing techniques. Human-Computer Interaction started with symbolic NLP around the 1950s [13]. This form relied on man-made rules that were detected in the set of data provided. During the surge of Machine Learning and statistical techniques for computation around the early 1990s, the previously symbolic systems began to change to Statistical NLP models. Although Statistical NLP models are still a great part of NLP computation, the introduction of Deep Learning techniques focused the sector towards the use of deep neural network methods, also known as Neural NLP. In the past years, there have been great strides in Neural NLP with the creation of transformers and large language models [13].

Recent advances in Neural NLP are focusing on zero-shot or few-shot learning [14]. There has been a surge of large language models (LLM) that have shown great promise in generalizing to different tasks. These require very few training data points and through the process of fine-tuning (also known as transfer learning) can very easily learn a new task without the need or cost of labeling thousands of data points [14].

These models have been very beneficial in lowering the time to obtain solutions and the computational cost to produce deep learning solutions. This technology has become more accessible and easier to use. Bringing about tools such as Huggingface [1], optimizations in libraries such as Tensorflow [15] and Pytorch [16] that allow ease in the use of pre-trained models.

NLP nowadays is used to support the automation of multiple NLP-related tasks through efficient and effective processing of large natural language document sets, including text summarization, keyword extraction, and topic modeling. Moreover, concerning topic modeling/keyword extraction, this is expanded in our system to mobile application features (or utilities) generation. This will be based on crowdsourced data, and will therefore become an abstractive task rather than an extractive task, differing this process from those previously described.

In this thesis, we are going to use different transformer models pre-trained on thousands of parameters alongside different fine-tuning techniques in order to obtain a few-shot training pipeline for feature extraction in specific domains. This will entail having a look at the current state-of-the-art transformer models and their adaptability to new situations. Using fine-tuning techniques, we will investigate these models and their ability to generalize in a given domain.

1.1.1 Context

This Bachelor's Thesis of the Computer Engineering Degree with a specialization in Computing, done in the Informatics Faculty of the Polytechnic University of Catalonia. The thesis is directed by Joaquim Motger de la Encarnacion and co-directed by Xavier Franch Gutiérrez, both members of the Service and Information Systems (ESSI) department.

1.1.2 Concepts

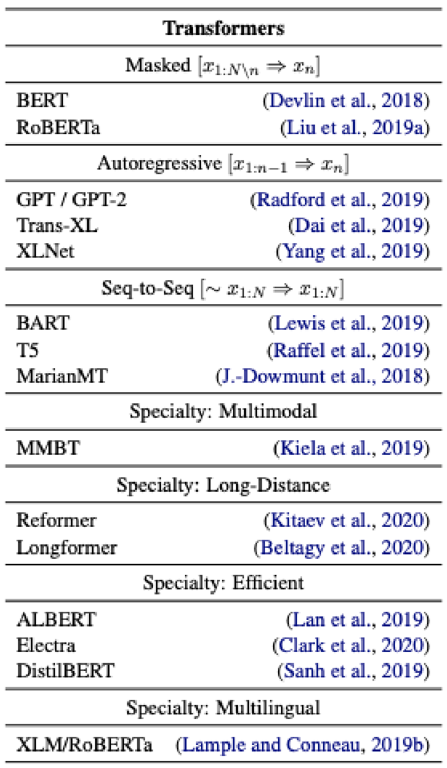
This dissertation focuses on the use of deep learning techniques for fine-tuning transformer models. The concepts outlined below provide the preliminary knowledge required to thoroughly understand this dissertation. The Concepts section serves as a brief introduction to the field of deep learning applied to Natural Language Processing. A deeper look into these techniques is explored in the following sections (Section 6. Deep Learning Fundamentals). In this section, a generic overview of these concepts can be found in order to understand the conceptualization and planning of the project.

This section was initially created in the Project Management course, an obligatory predecessor to the Final Degree Thesis. The contextualization portion of this memory has been split between this section and *Section 6 - Deep Learning Fundamentals*. Therefore, this section will cover the essential terms to understand all the sections leading up to Section 6.

- **Sequence to Sequence (Seq2Seq) Learning:** Architectures made up of an Encoder and a Decoder. The encoder takes care of extracting vectorized interpretations of the input data and the Decoder tried to obtain the correct output from the Encoder interpretations.
- **Attention:** Mechanism that is used to decide which part of the sequence is relevant for the current token and registers this within the internal states.
- **Self-Attention:** Self-attention relates the different tokens in the input sequence to the rest of the sequence positions.
- **Tranformer Models:** Deep Learning Seq2Seq model that contains the addition of self-attention.

State-of-the-art Transformer Models

Transformer models were introduced in the paper *Attention is all you need* [4] by a group of Google researchers. It was later popularized by the surge of the BERT [17] model and its variations RoBERTa [18]. Further studies have been performed using transformer models as their base such as GPT [19], and their variations GPT-2 [20] and GPT-3 [21]. The table below is a small summary of the different state-of-the-art transformer models proposed in the paper *Transformers: State-of-the-Art Natural Language Processing* [22] proposed by Hugging Face.



Transformers	
Masked $[x_{1:N\setminus n} \Rightarrow x_n]$	
BERT	(Devlin et al., 2018)
RoBERTa	(Liu et al., 2019a)
Autoregressive $[x_{1:n-1} \Rightarrow x_n]$	
GPT / GPT-2	(Radford et al., 2019)
Trans-XL	(Dai et al., 2019)
XLNet	(Yang et al., 2019)
Seq-to-Seq $[\sim x_{1:N} \Rightarrow x_{1:N}]$	
BART	(Lewis et al., 2019)
T5	(Raffel et al., 2019)
MarianMT	(J.-Dowmunt et al., 2018)
Specialty: Multimodal	
MMBT	(Kiela et al., 2019)
Specialty: Long-Distance	
Reformer	(Kitaev et al., 2020)
Longformer	(Beltagy et al., 2020)
Specialty: Efficient	
ALBERT	(Lan et al., 2019)
Electra	(Clark et al., 2020)
DistilBERT	(Sanh et al., 2019)
Specialty: Multilingual	
XLM/RoBERTa	(Lample and Conneau, 2019b)

Figure 1: Hugging Face - Transformer Models Recompilation [1]

This table serves as an initial point for our model investigation. However, it was created in 2018 therefore there are certain models and architectures that are not up-to-date. These models are in constant change and therefore require constant updating. The state-of-the-art section in this paper sets us up for success by obtaining a comprehensive study of the current top models.

Fine Tuning Deep Learning Models

Fine-tuning refers to tweaking the underlying model in order to adapt it to a specific situation [14]. The most obvious example of this is adding an input and output layer to the model and training it with new data that is situation specific. This requires labeled data from the situation that we want to test against the model.

Another example of fine-tuning is transfer learning [14]. Transfer learning requires transferring the knowledge of an algorithm into a second algorithm. This follows the same pattern as teaching between a teacher and a student. There are a few ways of doing this. An initial version of this proposes training the Deep Learning model on large amounts of data and transferring the weights of this trained neural network into the new neural network.

Finally, transfer learning includes the possibility of contextualizing the model using embedding to train the following model. This allows the input of the second model to be contextualized in the scenario we are treating and attain an unsupervised fine-tuning of our model. Models for contextualizing include (TSDAE) [23] and Generative Pseudo Labelling (GPL) [**?**].

The examples and definitions mentioned above have been in circulation for a while. The current climate in NLP has a great amount of research based on fine-tuning. Therefore we require a vast amount of research to identify the most current and most used fine-tuning techniques to apply to our system.

1.1.3 Problem definition

The current NLP climate focuses on using large language models and transformer models to extract maximum efficiency in generalizing to different problem types. Based on a state-of-the-art review in the field of transformer models, in this thesis, we will conduct a comparative analysis through the fine-tuning and adaptation of transformer models in the field of domain-specific feature extraction. This analysis will be based on different quality dimensions, including accuracy, efficiency, and sustainability, and will result in the creation of an automated pipeline for NLP-related fine-tuning.

In this sense, feature extraction refers to obtaining key features or app utilities from our corpus (in this case mobile application descriptions). Our initial dataset is created from AlternativeTo, this website provides alternatives to private applications. AlternativeTo uses crowdsourced features to categorize their apps. We will be using these crowd source features as a base for our dataset labelling.

1.1.4 Stakeholders

The actors involved in this thesis can also be considered stakeholders for the research production. Currently, Joaquim Motger de la Encarnacion is pursuing a doctorate for which he requires extracting features from a series of application documentation. This process required a lot of manual work that will mostly be resolved with the creation of a fine-tuning pipeline for feature extraction. The author, Carla Campàs Gené, will be the primary entity for creating this pipeline and will look for potential applications throughout the process of development.

Furthermore, a lot of companies, specifically startups, have lately been relying on few-shot models in order to produce easy and cost-effective solutions. This investigation will produce an easy system to understand which model and fine-tuning technique are more coherent for feature extraction. Finally, the code base that will result from this paper can be utilized for optimizing different NLP problem sets with few-shot models. Therefore, expanding the reach of problem types that can be solved using this dissertation. Although these will not be directly treated in the progression of the thesis due to the temporal limitations.

Finally, the research in NLP has become increasingly more centered on the possibilities and extensibility that the now-arising transformer models have. This thesis will provide a paradigm on which to base the fine-tuning research of these models and simplify the process of tuning and analysis.

1.2 Justification

1.2.1 Previous studies

Previously we exposed the advances in transformer models and the strides that have been done toward the generalization of the models. There has been significant research on the benefits of fine-tuning when generalizing to Deep Learning models.

Several studies have made strides in discovering different ways to perform transfer learning. Its first instances came in 1992, although not related to Deep Learning this paper proposed forms of combining the output knowledge of one algorithm with another algorithm [24]. Further studies have been made on the applicability of transfer learning with Machine Learning and Deep Learning models [25]. The latest research focuses on transfer learning when applied to unsupervised deep learning [26].

Finally, the appearance of transfer learning research and new techniques within the field has increased overwhelmingly with the appearance of larger models. There have been several examples of these techniques done to keep track of the progress in the field.

1.2.2 Justification

The previously exposed studies suggest great improvement in the generalization of NLP tasks. Although these have provided great strides in the NLP area, they are trained on billions of parameters. Training and maintaining these models individually is extremely costly and in many companies, not a viable option.

The purpose of this study is to validate the capabilities of fine-tuning transformer models for feature extraction in the domain of mobile applications by using previously exposed technologies. To do this, we will build a fine-tuning pipeline, from which we will have the ability to easily test and extract the proper validation metrics for each model. The results of this thesis will validate the effects of fine-tuning and creating a simpler system for testing different models and providing faster solutions to NLP tasks with deep learning.

1.3 Scope

1.3.1 Objectives and sub-objectives

There are two major objectives that this thesis should produce. The initial objective of this thesis is to review the state-of-the-art transformer models and fine-tuning techniques to identify the most accurate model for domain-specific feature extraction. The second objective of this thesis will be to create a pipeline for fine-tuning different state-of-the-art transformers with different fine-tuning techniques. This will create an easier form of obtaining fine-tuned models for generating features from mobile application documentation. In order to achieve this, the general objectives have been broken down into sub-objectives.

Theoretical Sub-Objectives

- Research state-of-the-art transformer models and architecture.
- Research fine-tuning techniques for transformer models.
- Explore regularization techniques for fine-tuning deep learning models.
- For each method:
  - Compute spatial and temporal complexities.
  - Analyze the expected loss and accuracy of the model before fine-tuning.

Practical Sub-Objectives

- Develop the code base for the studied transformer models and fine-tuned techniques.
- Generalize fine-tuning process to be applicable to different transformer models.
- Analyze the loss and accuracy of the fine-tuned model.
- Automatize ranking of models and fine-tuning techniques and draw conclusions on obtained results.

1.3.2 Requirements

The following requirements are set in place in order to maintain the quality of the thesis.

- Generate and label data-set for fine-tuning in the mobile application domain.
- Select the most relevant fine-tuning techniques and best-fit models for feature extraction.
- Attain the best possible models by optimizing hyper-parameters for the fine-tuned models.
- Define and add the evaluation process to the pipeline to evaluate models.
- Optimize code for all fine-tuning techniques.
- Maintain code readability and remove code complexity, as per good programming practices.

1.3.3 Potential obstacles and risks

This study has some risks and potential roadblocks to take into account. Having these obstacles in mind while producing the thesis will help avoid and take these into account.

- **Project deadline.** Having only four months to complete the study and the posterior paper that will follow is going to limit the breadth and depth that the study can reach. This will force taking some decisions to be made and leave unexplored paths in the way. In order to avoid having major setbacks due to the decisions, we require rigorous initial research to understand the scope of the study.

- **Computational Power.** Training these models are very computationally expensive. Although fine-tuning is in place to reduce the individual computational power needed, we might need to train the models ourselves to extract the most out of our fine-tuning. For this, we will require large computational power and time for training.
- **Over-fitting and under-fitting.** Models are very prone to over-fitting or under-fitting based on the data that we have and how these models are trained. This can be a setback in the way the span of techniques can fine-tune.
- **Inexperience in the field.** While I have had some experience using NLP frameworks and libraries (such as Rasa and Spacy), I haven't had much experience with creating and tuning models. This means that an initial step is to learn the base of Deep Learning and the libraries involved.

1.4 Methodology and Methodological Rigor

As described, one of the obstacles we have is the short deadline we have to implement and deliver the project. To facilitate this we will require a proper work methodology and validation.

1.4.1 Methodology

Given that this thesis project is done in conjunction with the ESSI department, I will be adopting an agile methodology used in this department. This will facilitate the interaction between the director, the co-director, and the author of the thesis. Furthermore, this methodology has been proven useful in past interactions and will allow for swift refocusing if needed during any point of the project.

To implement this methodology, we will be using the Taiga tool. This tool allows for the creation of tasks and sub-tasks and the continual checking of the state of each task by adding progress and comments. This will allow detection of the point of completion of the task at all points. The tasks will be created for two-week sprints.

1.4.2 Monitoring Tools and Validation

We will use GitHub as a system for version control, all functional parts of the project will be tested and actions will be implemented to check that these aren't broken at any version. Different branches will represent different functionalities of the project (models or fine-tuning techniques). These should be merged into the main branch (previously known as the master branch). This project will be publicly available under an open-source software license (OSS), following the previous work done by the ESSI group.

To keep track of the tasks and progression we will use the Taiga tool, as previously explained. In order to optimize the resources, we use and have access to GPU and TPU computing capacity we will use Google Colaboratory. Having these we will be able to verify the correct functioning of each model and assure small variability by running each experiment more than once.

Weekly meetings will be scheduled with the director of the thesis in order to keep track of the progression and iterate on the following tasks. Bi-weekly meetings will be scheduled with the co-director for the same purpose. Furthermore, for work revision and any errors encountered during the execution of the thesis sporadic meetings might be set up.

2 Project Planning

This section describes the initial organization planned out previous to the start of the final degree project. This organization and planning are carried on through to the budget planning. All deviations from the initial planning are accounted for in the *Section 4 - Modifications with Respect to Initial Planning.*

2.1 Description of Tasks

Attending to the proposed length and dedication given by the faculty, the thesis is 18 ECTS at 27.5 hours per ECTS. This means that during the project's duration, there will be a total dedication of 495 hours. Out of which 82.5 hours should be dedicated to the Project Management (GEP) course. This leaves 412.5 hours for the project development, documentation, and presentation.

The project spans from the 19th of September 2022 to the 18th of January 2023. The first weeks are dedicated to the GEP course. Meaning that the totality of the project should be produced between the 17th of October 2022 and the 23rd of January 2023. During these 14 weeks, there should be a 30-hour dedication.

2.1.1 Task Definition

The task definition will be in constant progression to assure the correct conditioning of the project. The tasks and temporal planning defined below are intended to be an initial version that will be built upon as the project proceeds.

Project Planning (T1)

Initially, we are going to identify the context and scope of the project. This is encompassed in the Thesis Management (GEP) course.

In order to do this, we will follow these sub-tasks:

- **Contextualization and Scope.** (T1.1) Identify the context and concepts relevant to the thesis by setting clear objectives and justifying the capabilities of the project.
- **Project and Temporal Planning.** (T1.2) Break down projects into smaller tasks and identify the time frame for the project and individual tasks. This will include required resources and risk management.
- **Economic Planning and Sustainability.** (T1.3) Analyze the cost that pursuing this project will assume within an economic and sustainable frame.
- **Correction and Final Documentation.** (T1.4) GEP assigns a tutor to track the progress of the documentation. The final step is to revise the comments and turn in a final version.
- **Meetings.** (T1.5) Weekly meetings will be set up with the director of the thesis and bi-weekly meetings with the co-director.

Revise, recompile and analyze state-of-the-art technology (T2)

The initial part of the project will be looking into the current state-of-the-art transformer models and identifying which of these will be best to use in our system. We will also need to identify the possible fine-tuning techniques that we could use in this thesis and identify which of these will be most beneficial.

In order to do this, we will follow these sub-tasks:

- **Describe transformer architecture.** (T2.1)
- **Compute theoretical time and space complexities and bounded loss.** (T2.2)
- **Compare generic use case results between transformers.** (T2.3)

Practical Implementation of Transformer Models (T3)

In order to be able to properly fine-tune the models, we will need access to their architecture. This will require us to program the architecture ourselves which will give us access to manipulating this model for fine-tuning.

In order to do this, we will follow these sub-tasks:

- **Learn syntax for deep learning python library (Tensorflow).** (T3.1)
- **Program transformer models.** (T3.2)
- **Test correct functioning of transformer models.** (T3.3)

Practical Implementation of Fine Tuning Techniques (T4)

Once the transformer models are coded, we will require the fine-tuning techniques to be programmed and adapted for each transformer proposed.

In order to do this, we will follow these sub-tasks:

- **Check the possibility of generalizing fine-tuning techniques amongst all transformers.** (T4.1)
- **Program fine-tuning technique. If the fine-tuning technique is not generalizable manually tune all the transformer models.** (T4.2)
- **Implement parameter sweeping to tune hyper-parameters.** (T4.3)
- **Test correct functioning of transformer models.** (T4.4)

Experimentation, Analysis, and Conclusions (T5)

Having programmed and theoretically analyzed all possible techniques, the next step is to put these models into practice in a specific use case and analyze their realistic use.

- **Generate dataset and labeling.** (T5.1)
- **Select appropriate benchmarks to avoid bias.** (T5.2)
- **Experiment over different models and fine-tuning techniques.** (T5.3)
- **Draw conclusions from results.** (T5.4)

Documentation and Presentation (T6)

This task requires constancy during all the implementation steps, this serves to document the current progress properly and modify these with different decisions and updates. Finally, once the process has been completed a presentation resuming the concepts will serve as a wrap-up for the project.

In order to do this, we will follow these sub-tasks:

- **Memory Write-up** (T6.1)
- **Final Thesis Write-up** (T6.2)

2.1.2 Summary of the Tasks

The table below summarizes the tasks exposed above with expected dependencies and approximate temporal planning of them.

Task ID	**Task**	**Time (h)**	**Dependencies**
T1	**Project Planning**	**85**	–
T1.1	Contextualization and Scope	25	–
T1.2	Project and Temporal Planning	10	T1.1
T1.3	Economic Planning and Sustainability	15	T1.2
T1.4	Correction and Final Documentation	15	T1.3
T1.5	Meetings	20	–
T2	**State-of-the-art Review**	**150**	–
T2.1	Describe transformer architecture	75	–
T2.2	Compute time, space and bounded loss	37.5	–
T2.3	Compare generic transformer use cases	37.5	–
T3	**Practical Implementation of Transformer Models**	**125**	T2
T3.1	Learn Tensorflow	25	–
T3.2	Program transformer models	75	T3.1
T3.3	Test correct functioning of transformer models	25	T3.2
T4	**Practical Implementation of Fine Tuning Techniques**	**100**	T2
T4.1	Check generalization of fine-tuning techniques.	10	–
T4.2	Program fine-tuning technique	50	T4.1
T4.3	Implement parameter sweeping to tune hyper-parameters	15	T4.2
T4.4	Test correct functioning of fine-tuning techniques	25	T4.3
T5	**Experimentation, Analysis and Conclusions**	**85**	T3, T4
T5.1	Generate dataset and labelling	20	–
T5.2	Select appropriate benchmarks to avoid bias	20	T3, T4
T5.3	Experiment over different models and fine-tuning techniques	25	T3, T4
T5.4	Draw conclusions from results	20	T5.3
T6	**Documentation and Presentation**	**50**	–
T6.1	Project Memory Write Up	25	–
T6.2	Thesis Write Up	25	–

Table 1: Tasks summary - duration and dependencies (own creation)

2.2 Resources

As part of the thesis planning, we have to take into account the resources that we will need to execute this thesis. These resources can be broken down into four groups: human, material, software, and hardware.

2.2.1 Human Resources

The main human resources involved in the project are the researcher (Carla Campàs Gené), the director (Joaquim Motger De La Encarnacion), and the co-director (Xavier Franch Gutíerrez). The researcher will be the point guard for the project and the director and co-director will provide the required support and guidance.

Other fundamental human resources involve the GEP tutor who will provide support in the initial steps of the thesis and the final thesis panel composed of examiners from within the Computing specialization who are in charge of revising the final product and grading it.

2.2.2 Material Resources

The lack of base knowledge in the given project will require a lot of study on papers and technologies. We will require access to books dealing with deep learning, NLP, fine-tuning techniques, and Tensorflow technologies. These are evaluated below in *Section 2.4 - Project Deadline.*

2.2.3 Software Resources

As described above we will require access to the following software services:

- **Colaboratory (SR1):** Interactive python notebook with access to more extensive hardware resources.
- **Google Calendar (SR2):** Calendar software to keep track of meetings.
- **Taiga (SR3):** Software tool to keep track of tasks per sprint and the progress associated to each task.
- **VSCode IDE (SR4):** To compile the pipeline and generate a comprehensive component from the models and fine-tuning techniques we will need access to an IDE.
- **Atenea - FIB (SR5):** For the initial project planning we will need documentation posted in Atenea and we will also have to turn in documents in this software service.
- **Raco - FIB (SR6):** Tracking of completion and turn-in of final work will be done directly through the Raco platform.
- **Overleaf (SR7):** To write up the project planning, the memory of the project and the final thesis we will be using the Overleaf platform. This enables writing and visualizing latex code.
- **Github (SR8):** The software service will serve as version control and active service for the code produced.
- **Google Meet (SR9):** In the spirit of maintaining communication with the director and co-director some of these meetings will take form in an online setting.

2.2.4 Hardware Resources

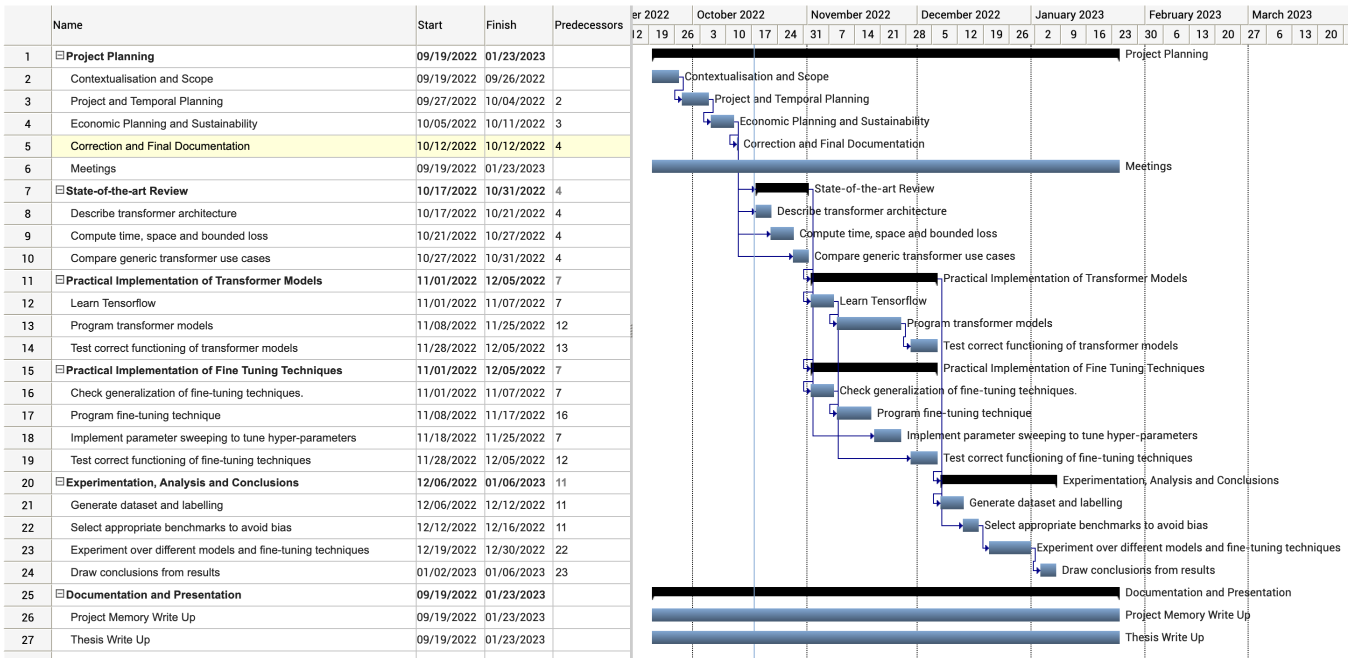
In order to code the required pipeline for this project, we will be using a MacBook Pro M1 with 512 GB memory and 16 GB RAM (HR1). Since we are training very heavy models, the CPU in this computer will not be enough to efficiently run these models. We will be using CPU, GPU, and TPU resources from the Colaboratory software (HR2). We will also need access to the internet for this, and therefore require a router for this connection (HR3).

Task ID	Hardware Resources	Software Resources
T1.1	HR1, HR3	SR5, SR7
T1.2	HR1, HR3	SR3, SR5, SR7
T1.3	HR1, HR3	SR5, SR7
T1.4	HR1, HR3	SR5, SR7
T1.5	HR1, HR3	SR2, SR3, SR9
T2.1	HR1, HR3	SR7
T2.2	HR1, HR3	SR7
T2.3	HR1, HR3	SR7
T3.1	HR1, HR2, HR3	SR1, SR4, SR7
T3.2	HR1, HR2, HR3	SR1, SR4, SR7, SR8
T3.3	HR1, HR2, HR3	SR1, SR4, SR7, SR8
T4.1	HR1, HR3	SR7
T4.2	HR1, HR2, HR3	SR1, SR4, SR7, SR8
T4.3	HR1, HR2, HR3	SR1, SR4, SR7, SR8
T4.4	HR1, HR2, HR3	SR1, SR4, SR7, SR8
T5.1	HR1, HR2	SR4, SR7, SR8
T5.2	HR1, HR2	SR4, SR7, SR8
T5.3	HR1, HR2, HR3	SR4, SR7, SR8
T5.4	HR1, HR2, HR3	SR7
T6.1	HR1, HR3	SR6, SR7
T6.2	HR1, HR3	SR6, SR7

Table 2: Hardware & Software Requirements by Ta (own creation)

2.3 Gantt Chart

The Gantt Chart will be a visual distribution of the described tasks. We ended the task deadline previous to the actual deadline (23rd of January) this will serve to be able to amend for possible delays. It does not include the meetings schedule, it is still not set, and therefore is preferable to exclude it from the visual representation. An enlarged version of the Gantt chart is found in Annex A if a better view is required.



	Name	Start	Finish	Predecessors
1	**Project Planning**	**09/19/2022**	**01/23/2023**	
2	Contextualisation and Scope	09/19/2022	09/26/2022	
3	Project and Temporal Planning	09/27/2022	10/04/2022	2
4	Economic Planning and Sustainability	10/05/2022	10/11/2022	3
5	Correction and Final Documentation	10/12/2022	10/12/2022	4
6	Meetings	09/19/2022	01/23/2023	
7	**State-of-the-art Review**	**10/17/2022**	**10/31/2022**	4
8	Describe transformer architecture	10/17/2022	10/21/2022	4
9	Compute time, space and bounded loss	10/21/2022	10/27/2022	4
10	Compare generic transformer use cases	10/27/2022	10/31/2022	4
11	**Practical Implementation of Transformer Models**	**11/01/2022**	**12/05/2022**	7
12	Learn Tensorflow	11/01/2022	11/07/2022	7
13	Program transformer models	11/08/2022	11/25/2022	12
14	Test correct functioning of transformer models	11/28/2022	12/05/2022	13
15	**Practical Implementation of Fine Tuning Techniques**	**11/01/2022**	**12/05/2022**	7
16	Check generalization of fine-tuning techniques.	11/01/2022	11/07/2022	7
17	Program fine-tuning technique	11/08/2022	11/17/2022	16
18	Implement parameter sweeping to tune hyper-parameters	11/18/2022	11/25/2022	7
19	Test correct functioning of fine-tuning techniques	11/28/2022	12/05/2022	12
20	**Experimentation, Analysis and Conclusions**	**12/06/2022**	**01/06/2023**	11
21	Generate dataset and labelling	12/06/2022	12/12/2022	11
22	Select appropriate benchmarks to avoid bias	12/12/2022	12/16/2022	11
23	Experiment over different models and fine-tuning techniques	12/19/2022	12/30/2022	22
24	Draw conclusions from results	01/02/2023	01/06/2023	23
25	**Documentation and Presentation**	**09/19/2022**	**01/23/2023**	
26	Project Memory Write Up	09/19/2022	01/23/2023	
27	Thesis Write Up	09/19/2022	01/23/2023	

Figure 2: Planned Tasks - Gnatt Chart (own creation)

2.4 Risk Management

In order to properly assess the timeline for this project, we will have to take into account the risks exposed in the previous section (1.3.3).

2.4.1 Project Deadline

A very likely scenario is underestimating the number of hours it takes to complete each task. In this case, we will have to reassess the scope of the project and the reach of the number of models and/or fine-tuning techniques that can be included.

- **Impact:** Medium
- **Proposed Solution:** In order to solve this, there should be a constant assessment of the point we are at. Due to the fact that we are using sprints to control the tasks for each week, after each sprint, I will take some time to evaluate the progress of the project. If at any point this is falling out of track we will add a 10-hour task to re-assess and prioritize.

2.4.2 Computational Power

The models we are attempting to build are very large and will therefore require high computational power in order to train and maintain these models. Training on low computational power will take a very long time and effectively skew the progress we could potentially attain.

- **Impact:** High
- **Proposed Solution:** There are many external sources that provide resources for us to train models without having to localize their training. These include cloud services such as AWS, but also online coding environments such as Colaboratory. We have opted for the latter task, but if need be we will be able to reassess and find a better permanent solution. In the case we see poor functioning in our chosen solution, we will implement a 20-hour task to implement a different solution.

2.4.3 Over-fitting and under-fitting

Training models make them prone to over or underfitting. This might be due to the parameters and hyper-parameters we use to train our model, the bias in the data, or (in the case of fine-tuning) using too much data in the process. The different amount of causes of over-fitting and under-fitting will require an investigation.

- **Impact:** High
- **Proposed Solution:** In order to figure out what is causing the over-fitting or under-fitting of the models we will need to investigate the impact of different causes of over/underfitting. This will require a hefty amount of time setting it at 50 hours. Being that this is such a large risk we will have some prevention metrics to make sure that if we require this investigation everything is set up to facilitate it. This will include checking the data distribution and implementing parameter sweeping, amongst others.

2.4.4 Inexperience in the Field

My inexperience in the field can have some minor setbacks in the project. This might be a lack of understanding of some errors or not knowing how to implement certain tasks fully. There is already a task in place in order to understand the framework there might still be some cases in which we require an extension of the previous knowledge.

- **Impact:** Low
- **Proposed Solution:** Depending on the situation, we will require more or less time to solve the potential bump in the road. In some cases, a simple Google search will fix it, if more investigation is required we will add a 10-hour task to minimize the situation.

3 Budget

The next step in the preparation of the project is figuring out the costs of the workforce, software requirements, hardware, and amortization. This will be divided into staff costs and generic costs. There will also be an exploration of the possible deviations in the budget and how to control and manage this budget.

This section is also part of the initial planning and reflects a theoretical planning, previous to starting. All modifications and pertinent changes have been added to the Modifications with Respect to Initial Planning section.

3.1 Staff Costs

Exploring the staff costs will require figuring out which roles would perform the different programmed tasks. We will then specify a mean salary for each type of role identified and multiply their hourly wage by the predicted hours required.

There will be three people taking up the different roles: the co-director will take up the role of **Team Lead (TL)**, the director will take up the role of **Engineering Manager (EM)** and I will be taking a vast amount of roles exposed below.

The initial tasks of planning and processing the different aspects of building the project will require a **Technical Project Manager (TPM)** who is in charge of setting the scope, budget, context, and stakeholders of the project. To figure out state-of-the-art technologies and research best efforts for the project, we will require a **Junior Researcher (JR)**. In order to build the deep learning pipeline effectively, we will need a **Machine Learning Engineer (MLE)**. Finally to test and experiment we will require a **Tester (T)** and to analyze the results a **Data Analyst (DA)**.

The following table indicates the hourly costs of the titles mentioned above. All of the salary information is extracted from the same website [27] in order to not bias the data obtained. The means are extracted from Spain in order to have accurate data for the people involved in this project, all roles that I will be taking up are assumed to be entry-level. The salary given by this website does not include payment of social security (gross salary), therefore we will have to add this to the final staff costs. We will therefore consider the total salary to be the base plus 35% of the base for social security. We have estimated a full-time salary meaning the total salary will be divided by 40(hours/week) * 4(weeks/month) * 12 months, totaling 1920 hours of work in a year.

Role	**Annual Salary (€)**	**Social Security Cost (€)**	**Cost (€/h)**
Team Lead	67,934	23,776	48
Engineering Manager	98,717	34,550	70
Technical Project Manager	47,763	16,717	34
Junior Researcher	51,416	17,995	36
Machine Learning Engineer	42,590	14,906	30
Tester	28,606	10,012	20
Data Analyst	32,881	11,508	24

Table 3: Estimated Salary per Role (own creation)

Having established these costs we will divide the defined tasks into the predicted roles for them and assign them the correct amount of hours per role.

Task	**Hours**	**TL**	**EM**	**TPM**	**JR**	**MLE**	**T**	**DA**
Project Planning	85	20	20	85	0	0	0	0
Contextualization and Scope	25	0	0	25	0	0	0	0
Project and Temporal Planning	10	0	0	10	0	0	0	0
Economic Planning and Sustainability	15	0	0	15	0	0	0	0
Correction and Final Documentation	15	0	0	15	0	0	0	0
Meetings	20	20	20	20	0	0	0	0
State-of-the-art Review	150	0	0	0	150	0	0	0
Describe transformer architecture	75	0	0	0	75	0	0	0
Compute time, space and bounded loss	37.5	0	0	0	37.5	0	0	0
Compare generic transformer use cases	37.5	0	0	0	37.5	0	0	0
Practical Implementation of Transformer Models	125	0	0	0	0	100	25	0
Learn Tensorflow	25	0	0	0	0	25	0	0
Program transformer models	75	0	0	0	0	75	0	0
Test correct functioning of transformer models	25	0	0	0	0	0	25	0
Practical Implementation of Fine Tuning Techniques	100	0	0	0	0	75	25	0
Check the generalization of fine-tuning techniques.	10	0	0	0	0	10	0	0
Program fine-tuning technique	50	0	0	0	0	50	0	0
Implement parameter sweeping to tune hyper-parameters	15	0	0	0	0	15	0	0
Test correct functioning of transformer models	25	0	0	0	0	0	25	0
Experimentation, Analysis and Conclusions	85	0	0	0	0	55	0	30
Generate dataset and labelling	20	0	0	0	0	20	0	0
Select appropriate benchmarks to avoid bias	20	0	0	0	0	20	0	0
Experiment over different models and fine-tuning techniques	25	0	0	0	0	15	0	10
Draw conclusions from results	20	0	0	0	0	0	0	20
Documentation and Presentation	50	0	0	0	50	0	0	0

Table 4: Temporal Dependencies Split By Role (own creation)

Now we can recompile this data into the total cost per task and per role. These are shown in Tables 5 and 6 respectively.

Role	**Cost (€/h)**	**Hours**	**Total (€)**
Team Lead	48	20	960
Engineering Manager	70	20	1400
Technical Project Manager	34	85	2890
Junior Researcher	36	200	7200
Machine Learning Engineer	30	230	6900
Tester	20	50	1000
Data Analyst	24	30	720

Table 5: Estimated Total Costs by Project Roles (own creation)

Having broken down the work to be done into the different tasks, we can conclude that the total

Role	**Total (€)**
Project Planning	5250
State-of-the-art Review	5400
Practical Implementation of Transformer Models	3500
Practical Implementation of Fine Tuning Techniques	2750
Experimentation, Analysis, and Conclusions	2370
Documentation and Presentation	1800

Table 6: Estimated Costs by Tasks (own creation)

staff costs for this project will be around 21,070€ without counting potential extra hours and other external factors that could affect this project.

3.2 Generic Costs

This section will deal with non-staff-related costs such as software, hardware amortization costs, and other costs.

3.2.1 Hardware Amortization

As outlined in the resources section (2.2.4) we will be using a MacBook Pro M1 for this thesis project. We will also be using GPUs and TPUs as hardware systems. GPUs are much more accessible than TPUs, therefore we will assume that 80 percent of model training will be done with GPUs and the remaining 20 percent of the time will be TPUs. The following table shows the resources and hours that we will use and the total price of the product. The prices for the used GPU and TPU are not available since they are privately owned and not sold to the public. We have found an equivalent Nvidia GPU/TPU, Nvidia is the lead contributor to GPU/TPUs for public consumption and will be the best estimate for the real prices.

The following formula corresponds to the amortization formula we will use to calculate the cost of

Resource	Hours	Price (€)	Life Expectancy
MacBook Pro M1	595	2,745	7
GPU	248	1500	5
TPU	62	3200	5

Table 7: Total hourly use of resources (own creation)

these during the thesis timeline. The expected total days of work will be 230 days working around 3 hours per day. For the years of use, we will use the life expectancy of each component.

$$Amortization = resourcePrice * \frac{1}{yearsOfUse} * \frac{1}{totalDaysOfWork} * \frac{1}{hoursPerDay} * hoursUsed$$

Using this equation, we are able to calculate the total cost of use of the hardware during the development of the thesis.

Resource	Amortization Cost (€)
MacBook Pro M1	385
GPU	240
TPU	290

Table 8: Amortization of hardware (own creation)

From table 8 we can extract the total hardware cost for the development of this project 915€.

3.2.2 Software

We will be using some free software to carry out the thesis. Aside from this, we need access to TPU and GPU systems. The easiest way to access this hardware will be using the Colaboratory Pro version. This is a total of 9.25€ per month during the span of September to January. This will be a total of 46.25€ during the span of the thesis execution.

3.2.3 Indirect Costs

Some costs are indirectly associated with the project. In this section, we will explore the costs.

- **Internet:** The current internet costs using fiber optics lies at 50€ per month. The total cost will be the division of this for the time that we will work. (50€/month) * (1 month/30 days) * (1 day/24 hours) * (230 days * 3 hours/day) = 50€.
- **Electricity Cost:** The current price of electricity is around 0.2038 euros per kilowatts per hour and the average desktop takes up around 200 Watts per hour. The total price (0.2038€/kWh) * 200wH * 230 days * 3h = 30€.

Indirect costs will total up to 80€.

3.3 Deviation of the Budget

Deviations from the proposed budget are inevitable. Having an expectation of these will help us manage the budget correctly and be prepared for the inevitable changes.

3.3.1 Contingency

The staff costs and indirect costs may vary during the development of the project. Therefore, we should be prepared to increase the budget accordingly if these requirements increase. In this case, it is favorable to us to go under budget but we will have to prepare for going over budget, we can do this by creating a contingency fund.

The amount of contingency will vary based on the resource that we are treating and the type of cost that we will have. We assume that staff costs will vary increasingly based on meetings, documentation, and research needed. We will assume a 10% contingency budget is required for staffing costs. Generic costs will assumingly vary less we will therefore include a 5% contingency budget for our generic costs. These are outlined in the table below. There are some costs that will not change such as software costs.

Cost Type	**Contingency Budget (€)**
Staff Costs	1,554.5
Amortization	75.75
Indirect Costs	6.4

Table 9: Contingency Budget per Cost Type (own creation)

The total contingency budget will be 1,637€.

3.4 Incidental Costs

We defined potential risks (1.3.3) and their effect on the project timeline (2.4). Having defined a risk level low, medium, or high, we can define staff costs and resource costs for each of these tasks to define the correct risk cost.

3.4.1 Staff Costs

Following the same process as section 3.1, we can calculate the costs of the identified risk situations.

Task	**Hours**	**TL**	**EM**	**TPM**	**JR**	**MLE**	**T**	**DA**
Project Deadline	10	0	0	5	5	0	0	0
Computational Power	20	0	0	0	0	20	0	0
Over-fitting and under-fitting	50	0	0	0	25	25	0	0
Inexperience in the Field	10	0	0	0	0	10	0	0

Table 10: Risk Management Staff Costs (own creation)

Role	**Total (€)**
Project Deadline	260
Computational Power	440
Over-fitting and under-fitting	1,225
Inexperience in the field	220

Table 11: Estimated Costs by Risk Task (own creation)

The expected total budget for risk interaction related to staffing is 2,145€.

3.4.2 Hardware Amortization

The table below is replicated from the previous section to be able to reproduce the amortization costs for incidental cases.

Resource	**Amortization Cost (€)**
MacBook Pro M1	52
GPU	31
TPU	65

Table 12: Amortization of Hardware for Risk Aversion (own creation)

Therefore for all declared risks, we will have a total incidental hardware amortization cost of 148€.

3.4.3 Software

The only cost endured in software is Colaboratory pro, this will not be affected by the appearance of a roadblock, we will have to buy the same program either way.

3.5 Total Costs

A quick summary of the total budget for the project is shown below.

Cost Type	**Total (€)**
Staff Costs	21,070
Hardware Amortization	915
Software	46.35
Indirect Costs	80
Contingency Budget	2,145
Incidental Costs	2,123
Total	**26,319.35**

Table 13: Amortization of Hardware for Risk Aversion (own creation)

3.6 Management Control

Having declared the potential budget, we now have to be able to track the real costs versus what we had budgeted. To do this, for each task we have to get the difference between the expected cost and the real cost after finishing the task. We will calculate this for the staffing necessities, the resources used, and other indirect costs. This will help us view how accurate our budget proposal is and the margin for error that we have during the previously defined tasks.

4 Modifications with Respect to Initial Planning

As seen in the previous sections, we had an initial development of the tasks we wanted to develop, the order of them, and the time frame allotted for each of these. These were slightly changed by unforeseen circumstances. This section deals with these circumstances, which tasks were added and which were modified.

4.1 Tasks

During the first half of this study, the allotted time was dedicated to the revision of state-of-the-art technologies and implementing the initial version of the system. During the latter part of the project development, I focused on refining the system and running experiments with the selected models. The depth of the project and requirements of the system limited the breadth of the system, which required a deep look into the types of models we wanted to test and how to implement these. Furthermore, the experimentation of the system took a very long time which limited further the scope of the study.

The state-of-the-art review also showcased a lot of blind spots in our contextualization, specifically the base with which the reader would receive the documentation would be very weak. This prompted the addition of the section fundamentals (T1.6). This section describes the base of transformers, starting from the most simple concepts of neural networks and scaling up to the different types of attention found in a transformer model. This also increased the time required to recompile state-of-the-art technologies.

Another change in our initial planning was the proposal of a pipeline. The models we tested were coded in Colaboratory (T5.1). This made it harder to run all of the different fine-tuning techniques for every model at the same time. There was also a lack of RAM to run the training on the pre-trained T5 model due to its size. This meant adding a translation task between Colaboratory and running on a local machine by adding a main method that can parallelize the different fine-tuning techniques and the different models. This isn't a very big task, given that local machines (and sometimes servers) have less access to resources than running in Colaboratory it is a very important task to consider.

While trying to learn Tensorflow for our specific system we realized that Tensorflow was on the fall and in turn, PyTorch is gaining importance in the field. Due to the overbearing amount of information and training found that used PyTorch, we decided to replace Tensorflow with PyTorch. These two systems are similar libraries but PyTorch has gained significant importance because it has "useful abstractions to reduce amounts of boilerplate code and speed up model development" [48].

We have also seen that thanks to the tools Huggingface there is a large ease of use in the implementation of basic fine-tuning techniques, this simplification has Having established these changes we can now identify the new task management and timeline.

In the experimentation section, we saw the analysis of the results was lacking when speaking in generic terms. Therefore, we added a task to perform some simple data analysis and visual assistance for the analysis of the system. Finally, the time taken to run the experiments of the test was a lot greater than what we expected, this severely increased the time requirement due to waiting times. However, thanks to the implementation of the Google Cloud VPN, we could have these tests running in the background while working on a different model or documentation. Therefore there is no direct increase reflected in the table.

The following table is the updated task list.

ID	Task	Time	Deps.
T1	**Project Planning**	**95**	–
T1.1	Contextualization and Scope	25	–
T1.2	Project and Temporal Planning	10	T1.1
T1.3	Economic Planning and Sustainability	15	T1.2
T1.4	Correction and Final Documentation	15	T1.3
T1.5	Meetings	20	–
T1.6	Fundamentals Formalization	10	–
T2	**State-of-the-art Review**	**150**	–
T2.1	Describe transformer architecture	75	–
T2.2	Compute time, space and bounded loss	37.5	–
T2.3	Compare generic transformer use cases	37.5	–
T3	**Practical Implementation of Transformer Models**	**115**	T2
T3.1	Learn PyTorch	25	–
T3.2	Program transformer models	65	T3.1
T3.3	Test correct functioning of transformer models	25	T3.2
T4	**Practical Implementation of Fine Tuning Techniques**	**90**	T2
T4.1	Check generalization of fine-tuning techniques.	10	–
T4.2	Program fine-tuning technique	40	T4.1
T4.3	Implement parameter sweeping to tune hyper-parameters	15	
T4.4	Test correct functioning of fine-tuning techniques	25	T4.3
T5	**Experimentation, Analysis and Conclusions**	**130**	T3, T4
T5.1	[**NEW**] Translate Colaboratory Notebooks to source code	20	T3, T4
T5.2	[**NEW**] Learn Google Cloud Infrastructure & Deploy training on Google Cloud	10	T3, T4
T5.3	Generate dataset and labelling	10	–
T5.4	Select appropriate benchmarks to avoid bias	20	T3, T4
T5.5	Experiment over different models and fine-tuning techniques	25	T3, T4
T5.6	Draw conclusions from results	20	T5.3
T5.7	[**NEW**] Generate visual graphs from training and evaluation metrics	15	T5.3
T6	**Documentation and Presentation**	**50**	–
T6.1	Project Memory Write Up	25	–
T6.2	Thesis Write Up	25	–

Table 14: Evolution of Tasks summary - duration and dependencies (own creation)

4.2 Resources

All the previously exposed resources will still be used, we will however require more resources to complete this study. We have seen that the Colaboratory tool is not enough to run all the experiments. This tool requires the user to be active and has a limited amount of RAM, even with the pro Colaboratory version it is not enough to run all the experiments we require. We decided to run these in a virtual machine. Initially, this Virtual Machine was meant to be requested from the ESSI research team's resources but due to time constraints, we decided to opt for running this virtual machine on Google Cloud. This will also allow the experiments to continue running overnight and therefore extract all the experiment information we require. While this is great for running experiments, we will run initial small experiments in Colaboratory and prepare the code to be easily run in the virtual machine. We will identify the virtual machine as SR10.

The hardware resources will remain the same as those specified in the initial documentation, taking into consideration the specifications of the virtual machine we require. In summary, our hardware specifications will now be the following:

- MacBook Pro M1 with 512 GB memory and 16 GB RAM (HR1)
- CPU, GPU, and TPU resources from the Colaboratory software (HR2)
- Router (HR3)
- GPU from the virtual machine (HR4)

The added virtual machine will have the following characteristics. The following table (Table 15)

Resource	**Quantity**
vCPU Cores	2
RAM memory	13 GB
Amount of GPUs	1 NVIDIA Tesla K80
Framework	Pytorch
Hard Disk Drive (Boot Disk)	100 GB

Table 15: Evolution of Hardware Resources - duration and dependencies (own creation)

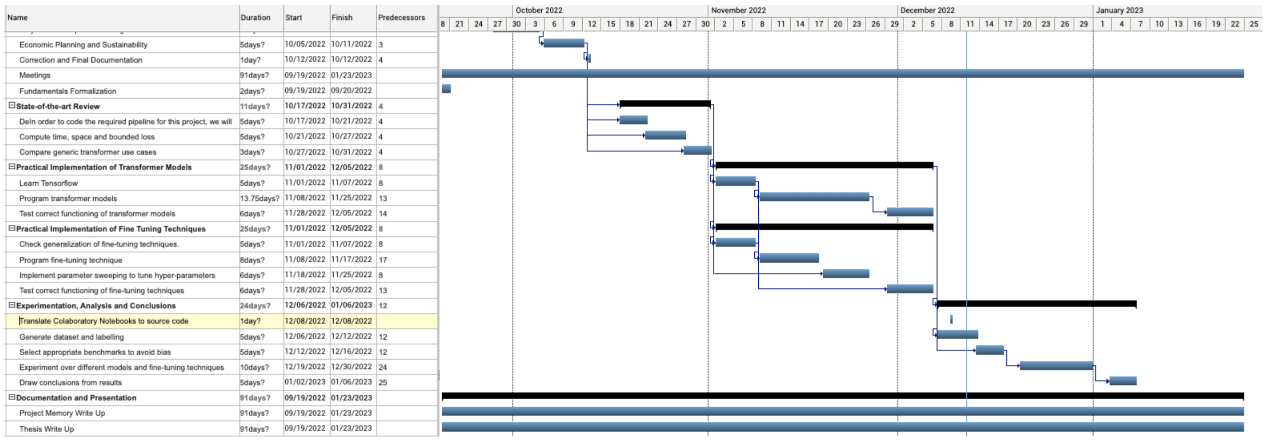
is the distribution of the resources with the specific tasks taking into account the added tasks and resources.

Task ID	**Hardware Resources**	**Software Resources**
T1.1	HR1, HR3	SR5, SR7
T1.2	HR1, HR3	SR3, SR5, SR7
T1.3	HR1, HR3	SR5, SR7
T1.4	HR1, HR3	SR5, SR7
T1.5	HR1, HR3	SR2, SR3, SR9
T1.6	HR1, HR3	SR5, SR7
T2.1	HR1, HR3	SR7
T2.2	HR1, HR3	SR7
T2.3	HR1, HR3	SR7
T3.1	HR1, HR2, HR3	SR1, SR4, SR7
T3.2	HR1, HR2, HR3	SR1, SR4, SR7, SR8
T3.3	HR1, HR2, HR3	SR1, SR4, SR7, SR8
T4.1	HR1, HR2	SR1, SR4
T4.2	HR1, HR2, HR3	SR1, SR4, SR7, SR8
T4.3	HR1, HR2, HR3	SR1, SR4, SR7, SR8
T4.4	HR1, HR2, HR3	SR1, SR4, SR7, SR8
T5.1	HR1, HR2, HR4	SR7, SR10
T5.2	HR1, HR2	SR4, SR7, SR8
T5.3	HR1, HR2	SR4, SR7, SR8
T5.4	HR1, HR2, HR3	SR4, SR7, SR8, SR10
T5.5	HR1, HR2, HR3, HR4	SR7
T5.6	HR1, HR3	SR1
T6.1	HR1, HR3	SR6, SR7
T6.2	HR1, HR3	SR6, SR7

Table 16: Evolution of Hardware & Software Requirements by Task (own creation)

4.3 Gantt Chart

Given all of these considerations, we have drafted an updated version of our initial Gantt chart. The Gantt chart contains a blue line marking the current state of the project.



Name	Duration	Start	Finish	Predecessors
Economic Planning and Sustainability	5days?	10/05/2022	10/11/2022	3
Correction and Final Documentation	1day?	10/12/2022	10/12/2022	4
Meetings	91days?	09/19/2022	01/23/2023	
Fundamentals Formalization	2days?	09/19/2022	09/20/2022	
⊟ **State-of-the-art Review**	11days?	**10/17/2022**	**10/31/2022**	4
DeIn order to code the required pipeline for this project, we will	5days?	10/17/2022	10/21/2022	4
Compute time, space and bounded loss	5days?	10/21/2022	10/27/2022	4
Compare generic transformer use cases	3days?	10/27/2022	10/31/2022	4
⊟ **Practical Implementation of Transformer Models**	25days?	**11/01/2022**	**12/05/2022**	8
Learn Tensorflow	5days?	11/01/2022	11/07/2022	8
Program transformer models	13.75days?	11/08/2022	11/25/2022	13
Test correct functioning of transformer models	6days?	11/28/2022	12/05/2022	14
⊟ **Practical Implementation of Fine Tuning Techniques**	25days?	**11/01/2022**	**12/05/2022**	8
Check generalization of fine-tuning techniques.	5days?	11/01/2022	11/07/2022	8
Program fine-tuning technique	8days?	11/08/2022	11/17/2022	17
Implement parameter sweeping to tune hyper-parameters	6days?	11/18/2022	11/25/2022	8
Test correct functioning of fine-tuning techniques	6days?	11/28/2022	12/05/2022	13
⊟ **Experimentation, Analysis and Conclusions**	24days?	**12/06/2022**	**01/06/2023**	12
Translate Colaboratory Notebooks to source code	1day?	12/08/2022	12/08/2022	
Generate dataset and labelling	5days?	12/06/2022	12/12/2022	12
Select appropriate benchmarks to avoid bias	5days?	12/12/2022	12/16/2022	12
Experiment over different models and fine-tuning techniques	10days?	12/19/2022	12/30/2022	24
Draw conclusions from results	5days?	01/02/2023	01/06/2023	25
⊟ **Documentation and Presentation**	91days?	**09/19/2022**	**01/23/2023**	
Project Memory Write Up	91days?	09/19/2022	01/23/2023	
Thesis Write Up	91days?	09/19/2022	01/23/2023	

Figure 3: Evolution of Planned Tasks - Gantt Chart (own creation)

4.4 Budget

These slight deviations from the initial plans will also have an impact on our budget. These deviations are outlined in this section. Although the hardware requirements have changed, the amortization of these will stay constant, we will be using GPU and TPU for the same amount of time. The staff salaries are still the same, but the hours for these have changed. Table 16 reflects these changes with the addition of new tasks.

Task	Hours	TL	EM	TPM	JR	MLE	T	DA
Project Planning	95	20	20	85	0	0	0	0
Contextualization and Scope	25	0	0	25	0	0	0	0
Project and Temporal Planning	10	0	0	10	0	0	0	0
Economic Planning and Sustainability	15	0	0	15	0	0	0	0
Correction and Final Documentation	15	0	0	15	0	0	0	0
Meetings	20	20	20	20	0	0	0	0
Fundamentals Formalization	10	0	0	10	0	0	0	0
State-of-the-art Review	150	0	0	0	150	0	0	0
Describe transformer architecture	75	0	0	0	75	0	0	0
Compute time, space and bounded loss	37.5	0	0	0	37.5	0	0	0
Compare generic transformer use cases	37.5	0	0	0	37.5	0	0	0
Practical Implementation of Transformer Models	125	0	0	0	0	100	25	0
Learn Tensorflow	25	0	0	0	0	25	0	0
Program transformer models	65	0	0	0	0	65	0	0
Test correct functioning of transformer models	25	0	0	0	0	0	25	0
Practical Implementation of Fine Tuning Techniques	90	0	0	0	0	75	25	0
Check the generalization of fine-tuning techniques.	10	0	0	0	0	10	0	0
Program fine-tuning technique	50	0	0	0	0	50	0	0
Implement parameter sweeping to tune hyper-parameters	15	0	0	0	0	15	0	0
Test correct functioning of transformer models	25	0	0	0	0	0	25	0
Experimentation, Analysis and Conclusions	130	0	0	0	0	85	0	45
Translate Colaboratory Notebooks to source code	20	0	0	0	0	20	0	0
Learn Google Cloud Infrastructure & Deploy training on Google Cloud	10	0	0	0	0	10	0	0
Generate dataset and labelling	20	0	0	0	0	20	0	0
Select appropriate benchmarks to avoid bias	20	0	0	0	0	20	0	0
Experiment over different models and fine-tuning techniques	25	0	0	0	0	15	0	10
Draw conclusions from results	20	0	0	0	0	0	0	20
Generate visual graphs from training and evaluation metrics	15	0	0	0	0	0	0	15
Documentation and Presentation	50	0	0	0	50	0	0	0

Table 17: Fixed Temporal Dependencies Split By Role (own creation)

To view this in perspective, the following tables represent the cost by role and the cost by generic task.

Role	**Cost (€/h)**	**Hours**	**Total (€)**
Team Lead	48	20	960
Engineering Manager	70	20	1400
Technical Project Manager	34	85	2890
Junior Researcher	36	200	7200
Machine Learning Engineer	30	260	7800
Tester	20	50	1000
Data Analyst	24	45	1080

Table 18: Fixed Estimated Total Costs by Project Roles (own creation)

Role	**Total (€)**
Project Planning	5250
State-of-the-art Review	5400
Practical Implementation of Transformer Models	3500
Practical Implementation of Fine Tuning Techniques	2750
Experimentation, Analysis, and Conclusions	3630
Documentation and Presentation	1800

Table 19: Fixed Estimated Costs by Tasks (own creation)

The total staff costs with the proper modifications from the extra tasks bring the total required budget to 22,330€.

4.5 Risks Encountered

There were various issues we encountered during the development of this project. Most of these were previously captured in the previous risk management section.

Project Deadline

The initial scope of this project was very large, attempting to make a pipeline with various models and various fine-tuning techniques. The project deadline and the issues encountered forced both the breadth and depth of this project to shorten. This required a more critical view of the requirements the study required for our specific domain and limiting the experimentation to this.

Computational Power

Computational power was a big issue for experimentation in this system. Our initial plan was to use Colab Pro to train and test the model. The maximum capacity of Colab Pro was still not enough to properly train this model. Therefore, we opted to transfer the experimentation to Google Cloud. This required the addition of two tasks:

- Translate Colaboratory Notebooks to source code (T5.1)
- Learn Google Cloud Infrastructure & Deploy training on Google Cloud (T5.2)

These tasks limited the time for experimentation and the quantity of experimentation we could reach. To do this, we limited the number of models to train to two and the fine-tuning techniques for the removal/addition of layers.

Inexperience in the field

The inexperience in the field played a big part in the amount of work we could get done and the decisions made. There are three instances where this has played a part.

- We had some issues with the T5 training. BERT is fine-tuned on the model output, whereas T5 is meant to be trained on the model generation. Initially, there were a lot of issues with the metrics that I wasn't sure how to deal with. To mend this we searched online and figured out what the issue was. This took a hefty amount of time. Having scoured the internet I have seen that there is a common misconception about this if the user does not have added metrics the loss will be calculated all the same. Therefore, the incorrect training goes unnoticed.
- Inexperience played a large part in the memory issues that went into the selection of the platform. Not only would understanding the model requirements have identified this problem previous to its development but it would have also played a part in the Virtual Machine selection. This initial selection was guided by that of the team but was too slow to obtain in the final stages of our project development.
- The time required to execute the tests would have also been an important giveaway to the scope we could have reached in this paper. This would have helped to initially determine suitable experiment limitations.

5 Sustainability

This section was an initial look into the sustainability of our project and if we understood the impact that this project would have on the different sustainability metrics. This initial work was done during the project management technique as a self-evaluation and introspective look.

5.1 Self-assessment

The current technological sector is affecting our environment at a greater rate than ever before. From creating landfills with old technology to mining for a mineral that is already scarce. This context leaves a sore feeling to anyone entering any technological field. Although I did understand the concept of sustainability and sustainable development I didn't realize the scope and impact that it could have on our day-to-day projects.

While doing the survey I realized I was thinking about sustainability in the more common sense of creating tangible products that are zero waste but not as much in the sense of sustainable software development. Throughout the course of Computer Engineering at this faculty, we have talked about economical aspects but haven't focused as much on the societal and environmental impacts of our products. In my case, this meant that a lot of the concepts (especially the metrics and how to calculate them) were new concepts for me.

Although the environmental impact is always at the back of my mind I had never focused much time on deciphering how to conceptualize the potential impact. The metrics I saw during this initial exploration were surprisingly detailed and I believe there should be more education on these.

5.2 Economic Dimension

Regarding Project Put into Production (PPP): Reflection on the cost you have estimated for the completion of the project

The estimation for the cost has been developed in the previous section. I believe this is fully extended and admits different routes and possible risks. The times that have been estimated seem reasonable for the project type, although in the case of doubt I have always taken the maximum expected time. This way we ensure we are setting a budget with which we will be able to cover the totality of the project.

Regarding Useful Life: How are currently solved economic issues (costs...) related to the problem that you want to address (state of the art)? How will your solution improve economic issues (costs ...) with respect other existing solutions?

There have been a few advances in fine-tuning techniques that have attempted to solved the economic issue related to training models. This means that people spend less time labeling data and training these models, the capacity required is minor and therefore costs less for the company and the work time for investigations has diminished.

This thesis attempts to create a simple pipeline for testing and attaining relevant data for a grapple of transformer-based NLP models for the task of domain specific mobile application feature extraction. The generated results and the conclusions raised from this research will provide the groundwork for simplifying the process of running experiments and analysis over a different set of models. Furthermore, the investigation of the generalization of fine-tuning to different models will allow us to extrapolate the technique to the different models that may appear in the future.

5.3 Environmental Dimension

Regarding PPP: Have you estimated the environmental impact of the project?

Although we haven't fully developed a sustainability report at this point, we have noted that there are several benefits to fine-tuning models rather than training due to the smaller amount of information required for training. Using less electricity and being in general more sustainable. We will be investigating this further during the duration of the thesis development.

Regarding PPP: Did you plan to minimize its impact, for example, by reusing resources?

One of the general goals of this thesis is to create a pipeline for fine-tuning different transformer models. Through this, we will also investigate how to generalize fine-tuning for the different chosen models. This will reduce the work done to tune the different models by reusing the previous work needed.

Regarding Useful Life: How is currently solved the problem that you want to address (state of the art)? How will your solution improve the environment with respect to other existing solutions?

The current state-of-the-art involves deep investigation into how transfer learning is best and how to perform this. There are a few pipelines created to facilitate the process. These have been investigating different concepts such as unifying machine learning pipelines [28], assembling models for fine-tuning [1], or creating a pipeline (data to model) [29] for model tuning.

This thesis attempts to amend all of the different models in order to provide an easy pipeline for investigation of the best model to use. This unifies the field and attempts to make the whole process more efficient by not re-running work and reusing all possible steps.

5.4 Social Dimension

Regarding PPP: What do you think you will achieve -in terms of personal growth- from doing this project?

This project will allow me to discover in depth the field of NLP. Although I have seen this is a very broad spectrum as a developer I will be able to fully immerse myself in the concepts and tools. Furthermore, the work I have done in the field has shown me that usually companies have to move fast and don't have enough time to test different models. This will facilitate future work that I have to do in the field.

Regarding Useful Life: How is currently solved the problem that you want to address (state of the art)? How will your solution improve the quality of life (social dimension) with respect to other existing solutions? Is there a real need for the project?

Currently, companies should have a great number of resources and a great amount of time to launch a product. Start-ups and mid-size companies don't have the luxury to fully investigate the options that they have. Currently, there is a choice made after the analysis of data. Although this is an informed solution, it won't always give the best performance for the company. This project will change the outlook of that by giving them access to a tool that will automatically process the different models and return the best-fine-tuned model for any NLP task given the labeled data.

Regarding Useful Life: Is there a real need for the project?

As seen previously there are several benefits to this project. We have seen the need to conduct this project and obtain the structure for a potential fine-tuning pipeline, especially for the startup climate. Furthermore, the specific application in which we are treating the project is a specific use case of the project. We have identified a specific section of the market that requires a simple component to be able to simplify the process and give better results for these use cases. This justifies the need for this project and identifies use cases for it.

6 Deep Learning Fundamentals

The following section describes the base concepts on which this research is being build. This section describes the base of deep learning and identifies the main characteristics to get a total comprehension of this study. These fundamentals will run through the bases of deep learning, giving a necessary introduction to understand transformer models and fine-tuning.

6.1 Artificial Neural Networks (ANN)

Artificial Neural Networks (ANNs), also known as Simulated Neural Networks (SNNs) and in short Neural Networks have played a part in the evolution of Machine Learning, forming the basic building block of Deep Learning. "A Neural Network is a computational learning system that uses a network of functions to understand and translate a data input of one form into the desired output, usually in another form" [30]. This technical definition is a simplification of what neural networks do, receive input, and output the desired information. A visual example would be given an image identifying if the main focus of the image is a cat or a dog.

ANNs are inspired by the human brain, this is an attempt to mimic the way a biological neuron sends signals to another. It uses perceptrons as building blocks to create entangled networks that identify the context and important characteristics of the desired output. Neural networks are trained on large datasets, this allows for the neural network to distinguish different cases that might be inputted and learn how to identify the correct output. Seen in Figure 4 is the structure

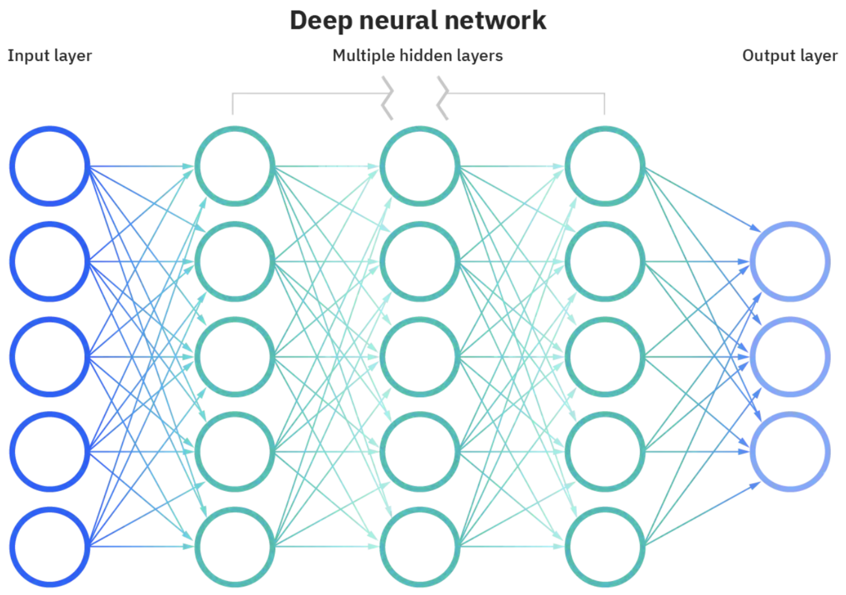


Figure 4: Neural Network [2]

of a basic neural network. The input layer is the layer where we introduce the data to the neural network. This layer is connected to a hidden layer, which manipulates the initial data through weights and biases to obtain meaningful characteristics for the desired (input, output) pairs. There can be many hidden layers with varying amounts of nodes. Finally, a hidden layer is connected to the final layer of the neural network, the output layer. This layer contains as many nodes as the number of classes or characteristics we want to identify from the input data (i.e. the output that we expect). To properly understand these concepts, let's focus on the building blocks.

6.1.1 The Perceptron

The perceptron consists of four different aspects:

- Numerical Input Values
- Weights and a bias
- Weighted sum
- Activation Function

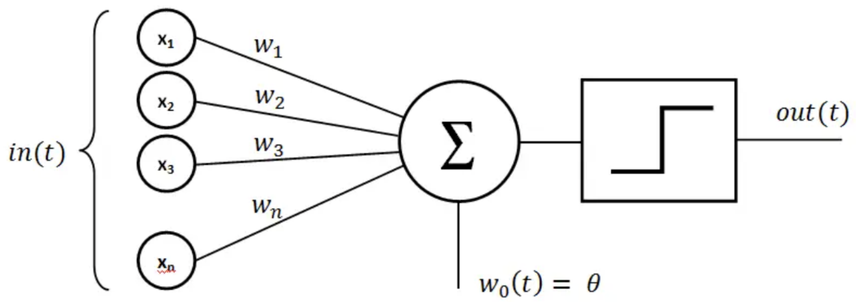


Figure 5: Perceptron [3]

Figure 5 shows a graphical representation of the aspects enumerated previously. Let's assume we have three input data points (x_1, x_2, x_3), for a single perceptron these have individual weights (w_1, w_2, w_3). The output of the weighted sum $w_o(t) = x_1 * w_1 + x_2 * w_2 + x_3 * w_3$. This provides the weighted sum based on the assigned weights for the specific proton and the input variables. the last component of the perceptron is the bias. The bias is a maintained threshold that the specific proton is required to reach before it can produce an output. Therefore the final equation for our weighted sum is the following: $w_o(t) = \sum X * Y + b$ [31].

The final step for data processing by the perceptron is the activation function. This provides a normalization step for our data. For example, if we want the output data to range between 0 and 1. There are various activation functions possible, the activation function used is largely dependent on which stage of the neural network we are at (input layer, hidden layer, output layer) and the type of problem we are dealing with (regression, classification...).

6.1.2 Training

In order to adjust the weights and biases to the training data we have procured we have to identify a method to compute the error of the neural network and change the weights to minimize this error. Training of neural networks is made up of two parts: the forward pass and the backward pass (also known as backpropagation). The forward pass is calculating the outputs for each neuron in the neural network and the loss (error) of the output of the neural network, which we can do using the formulas and concepts exposed previously. Similar to the activation function, there are various loss functions that can serve to calculate the loss of the output. These are selected based on the problem type and the architecture of the neural network.

Backpropagation is a step in which we will recalculate the weight and biases of the neural network based on the loss of the function. This step is essential to the network being able to properly identify the functioning of the data and correct the output. In other words, backpropagation has as an objective to minimize the cost by adjusting the model's weights and biases [32], The gradient is computed in order to assimilate the amount of change required for a specific layer and a specific perceptron. The computation of the gradients is a very simple computation using the chain rule:

$$\frac{dz}{dx} = \frac{dz}{dy} * \frac{dy}{dx}$$

Propagating the error through the different layers means that each layer is dependent on the change produced by the previous layer, all the way up to the first hidden layer. This way the loss affects the weights and biases of each layer until we have found a local minima and have identified the best potential functioning of our neural network [32].

6.2 Sequence to Sequence Learning

Sequence to Sequence (Seq2Seq) models are neural networks that take in sequences of tokens (characters, words...) and output transformed sequences [4]. These models' architecture contains an Encoder and a Decoder. Both the Encoder and Decoder are the most popular Long Short Term Memory (LSTM) [33] although there have been other implementations such as Gated Recurrent Unit (GRU) [34]. The following figure graphically demonstrates the Encoder-Decoder architecture used in Seq2Seq models.

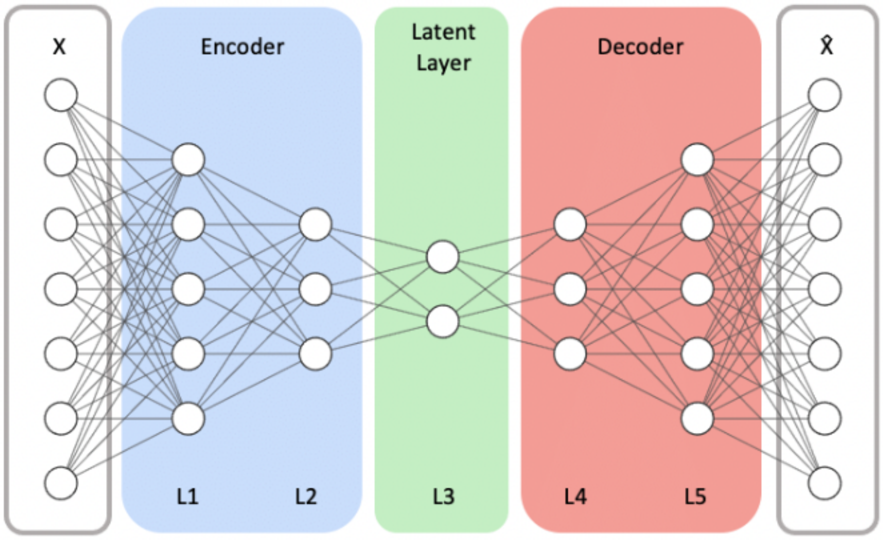


Figure 6: Encoder-Decoder Architecture Visualization [4]

The Encoder takes in the input sequence in the form of tokens, it then summarizes the information within the tokens and stores the important information in internal state vectors (also known as context vectors). The outputs of the Encoder model are discarded and the cell states are preserved.

The Decoders initial state is initialized to the context vectors in the final stages of the Encoder. Using these the decoder will generate the output sequence and take into consideration these outputs for future prediction of the sequence.

Within these models, a very important concept is that of Attention. The attention mechanism decides which part of the sequence is relevant for the current token and registers these within the internal state. The LSTM will consider several other inputs at the same time by adding weights to the different tokens in the sequence. This attention can be bidirectional (tokens before and after the current token are considered) or one-directional (only previously seen tokens are considered) [35].

6.3 Recurrent Neural Networks (RNN)

Recurrent Neural Networks (RNNs) are a type of ANN that allows the nodes to create a cycle, it is known for its excellent progress in structured data usage and manipulation. The data we are proposing, in this case, is sequential data. Sequential data refers to any type of data that contains a series of elements structured into a sequence. Examples of this are video, audio, and text. To deal with these kinds of inputs we require the network to remember previous inputs in order to link them together. RNNs introduce an information cycle within the perceptron that introduces a memory. Figure 7 shows the architecture of an RNN.

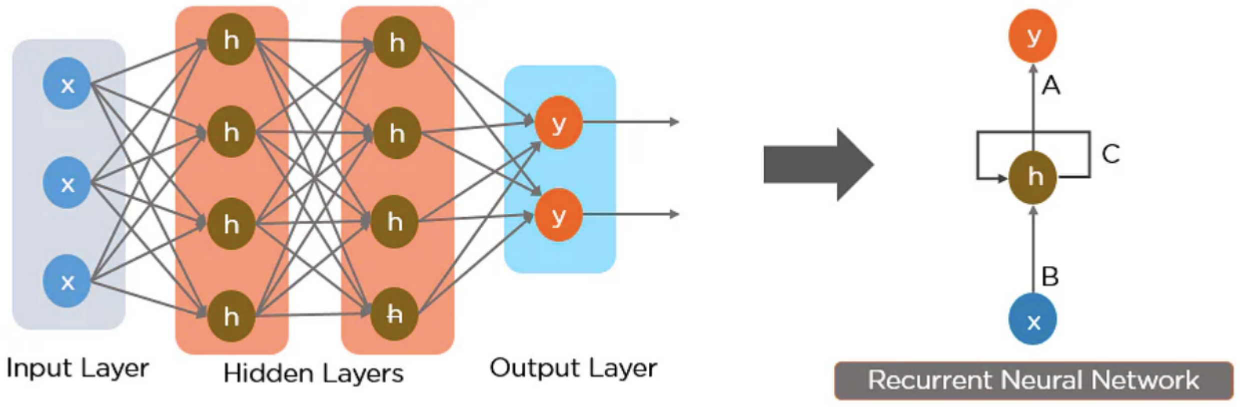


Figure 7: Recurrent Neural Network Architecture [5]

Using this architecture the network is able to remember past words and link them together. In this way, sequential data can be linked together to create artificial understanding. There have been variations of these that have increased the learning capacity associated such as (Long Short Term Memory) LSTM and (Gated Recurrent Unit) GRU.

6.3.1 Temporal and Spatial Complexity

The paper *Attention is All You Need* [4] introduces the temporal and spatial complexity of many of these models. RNN models introduce complexity per layer and the number of sequential steps. The complexity per layer introduced as $O(n * d^2)$ is the multiplication of the matrices of the hidden states from the previous step and the weight matrix. This multiplication takes d^2 operations and is processed for n steps.

The number of sequential steps is bounded by $O(n)$. All the n tokens are processed sequentially and therefore, the processing of the n tokens requires n steps.

6.3.2 Long Short Term Memory (LSTM) & Gated Recurrent Unit (GRU)

RNNs have a very short-term memory which serves a good purpose when small texts have to be processed. As this text grows more and more the link between two words is easily lost on RNNs. Although in theory, it is possible for the RNNs to link two words that are far away, in practice these networks have not shown the same capabilities. The introduction of LSTMs procured the ability to link words in long texts. The LSTM is a type of RNN with a complex structure. The following diagram (Figure 8) shows the repeating module structure.

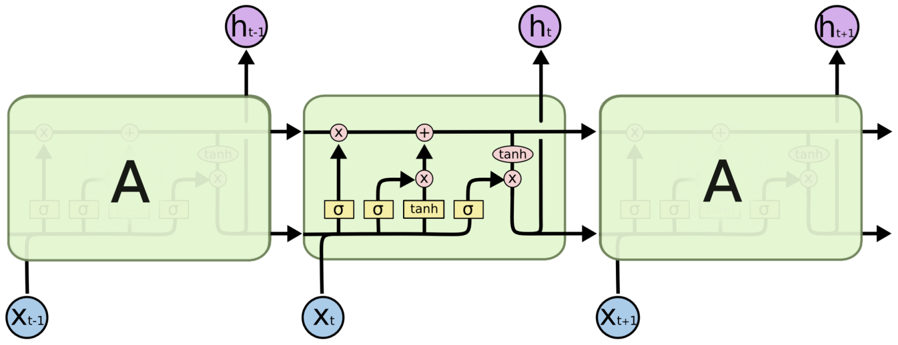


Figure 8: Long Short Term Memory Architecture [6]

The main part of the LSTM is the cell state (the horizontal line running through the top of the diagram. The LSTM has the ability to add or remove information to the cell state through the gates. Gates are made up of a sigmoid neural net layer and a pointwise multiplication operation, these are a form to let information through. The sigmoid layer outputs a number between zero (nothing goes through) and one (everything goes through) to identify how much of each token is let through. This is the attention mechanism. GRUs are LSTMs with a forged gate and without an output gate, it has fewer parameters than an LSTM.

6.4 Transformer Models

"A transformer model is a neural network that learns context and thus meaning by tracking relationships in sequential data like the words in this sentence. Transformer models apply an evolving set of mathematical techniques, called attention or self-attention, to detect subtle ways even distant data elements in a series influence and depend on each other" [36]. As explained above, the transformer model follows an encoder-decoder architecture.

Essentially, the transformer model is a Convolutional Neural Network with attention. Looking back at figure 2 we can see that: each encoder consists of a self-attention layer and a feed-forward network. The attention layer serves to identify the relevant parts of the input sentence. The decoder has an encoder-decoder attention layer between the self-attention and the feed-forward network, this extra layer works to specify important parts of the input.

6.4.1 Convolutional Neural Network (CNNs)

Convolutional Neural Networks solve the following problems: trivial to parallelize per layer, exploit local dependencies, and the distance between the positions is logarithmic. Input in CNNs are independent of each other, this aspect allows them to easily parallelize the network. For comparison, the distance of RNNs is O(N), making the CNN distance of O(log N) much more efficient. The issue with CNNs is that they don't identify dependencies between input tokens, which is where attention plays an important role in modeling these for natural language tasks.

6.4.2 Attention

RNNs introduce the concept of attention. Rather than encoding the whole sentence within a hidden state, RNNs encode each token with a corresponding hidden state passed from the encoder to the decoder. These states are then used to decode the input information for any given task. This computes a score from the given input to all other tokens and uses a softmax layer to level out the outputs, creating attention. The biggest hitch this technique has is that it can not compute different words in parallel, meaning that it cannot add any extra context to a single token.

6.4.3 Encoder Self-Attention

The initial input sequence is passed through to the input embedding and into the positional encoding. These produce encoded representations for each word. The encoded representations capture the meaning and position of each word. These are then passed to the Query, Key, and Value parameters in the Self-Attention layers. This layer adds attention to all the other input tokens with respect to our current token. The different encoders will add their attention representation to extract a world representation of the attention.

6.4.4 Decoder Self-Attention

In the same way that the encoder self-attention works, the decoder self-attention passes the input through the output embedding and positional encoding this in turn produces an encoded representation of the input. Once again, this is passed through to all three parameters Query, Key, and Value into the self-attention layer. Once we have obtained the attention scores we feed the query parameter to the Encoder-Decoder Attention.

6.4.5 Encoder-Decoder Attention

The Encoder-Decoder Attention gets a representation of the target and the input sequence, Decoder Self-Attention, and Encoder stack respectively. This produces a representation for each target sequence capturing the influence of the attention scores on the input sequence. This is continuously revised with every Encoder-Decoder pair.

6.4.6 Multi-head Attention

Transformer models parallelize multiple calculations of the attention in what is called attention heads. This is done by splitting the Query, Key, and Value tuples into N heads and passing it independently to each head. These are then combined together to produce the final attention.

6.4.7 Temporal and Spatial Complexity of Self-Attention

The paper *Attention is All You Need* [4]. The complexity per layer $O(d * n^2)$. In this case, d is the embedding dimension and the task at hand is to compute the attention weights for each word with respect to all the rest of the words. Therefore since it has to compute the attention for every word with respect to every word it will do n^2 computations of size d embeddings. The number of sequential steps is constant. All operations happen in the same time step, not sequential like previously seen in the RNN computations.

6.5 Importance of Fine-Tuning

There are a lot of cases in which using the full architecture of a proposed model (such as those proposed in the state-of-the-art review section) will be too much for the task at hand and will therefore be prone to over-fitting. This will make the model unreliable over unseen data. The idea of fine-tuning is reducing the amount of training required for your specific data. By doing this the model we train will have seen different tasks and will be re-trained with our specific task. This gives our model versatility.

Another important factor of fine-tuning is the versatility of the model architecture. Through fine-tuning, we will be reducing or augment the size of the model while maintaining the pre-trained weights. These techniques will be further explored and explained. This will allow us to find not only the best architecture for our given problem but the smallest one that will give us the best results.

7 State of the Art

This section covers the state-of-the-art models and fine-tuning techniques. In order to go through these. To do these we have examined a few related surveys of the technologies. These will be exposed in the pertaining sections. This section covers an in-depth look into these techniques.

7.1 Transformer Models Review

For this section, the scope is focused on surveys pertaining to transformer models, two models specially stood out and marked increasing importance in the formalization of this process. *A Survey of Transformers* [37] not only provides insight on the model efficiency, their capacity to generalize and ability to adapt but also a comprehensive figure that divides the transformer model by their different types. *AMMUS : A Survey of Transformer-based Pretrained Models in Natural Language Processing* specifies the survey into pre-trained transformer models for natural language tasks. The specification is useful to understand how the model fits into the fine-tuning scope.

7.1.1 Bidirectional Encoder Representations from Transformers (BERT)

BERT is a language model pre-trained on a plain text corpus [17]. The pretraining serves as a base of knowledge, this model can then be fine-tuned to user-specified queries. BERT is pre-trained on two unsupervised: tasks masked language modeling (MLM) and next sentence prediction (NSP). The following figure represents the architecture and training process of the BERT model.

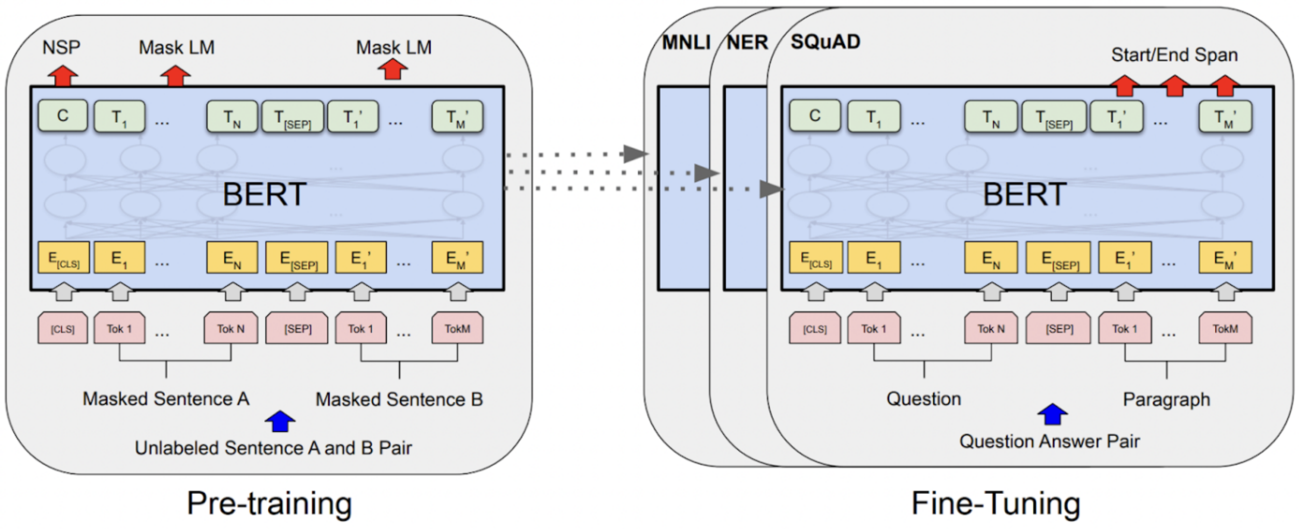


Figure 9: BERT pre-training and fine-tuning [11]

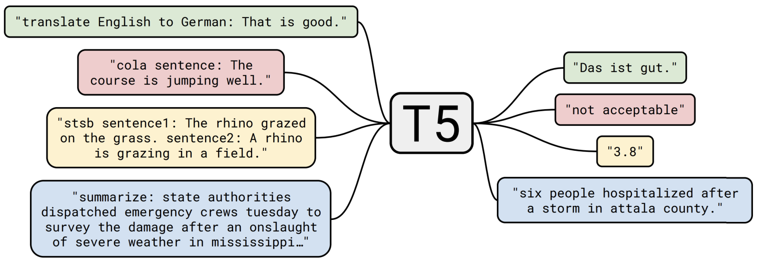
BERT uses an encoder-decoder architecture alongside MLM. The encoder masks part of the corpus and the decoder attempts to figure out which word is contained within the [MASK] token. The predictor only predicts the masked word, it does not restructure the sentence. There is a slight mismatch in the fine-tuning, in training with MLM, the training process has a '[MASK]' token to identify the word to be predicted, this does not exist and the has to be a user-set mask to replace it. To mitigate this, we do not always replace the token with [MASK]. 80% of the time it will be replaced with the [MASK] token, 10% of the time a random token will be used and the other 10% of the time we will use the unchanged token.

BERT is also pre-trained on NSP, and can therefore be tuned on this task. The sentences are generated in $< A, B >$ pairs. In order for this to be effective, we have to generate negative sampling for our data. Half of the sentences should be matched to their appropriate representation, and the other half should be matched to a random sentence of the corpus. BERT is pre-trained on 110 million parameters.

- Performing sentiment analysis, such as predicting the sentiment of a movie.
- Question answering, such as chatbots.
- Text prediction, this is specifically used in Gmail.
- Text generation, such as writing an article from a prompt.
- Summarization, such as generating summaries from articles.

7.1.2 Text to Text Transfer Transformer (T5)

The T5 model proposes a unified text-to-text format where all input and output are in text format. This is trained on the C4 dataset (Colossal Clean Crawled Corpus). This text is the processed version of the Common Crawl dataset, and it has been preprocessed by extracting only English excerpts, removing code lines and duplicates.



"translate English to German: That is good."
"cola sentence: The course is jumping well."
"stsb sentence1: The rhino grazed on the grass. sentence2: A rhino is grazing in a field."
"summarize: state authorities dispatched emergency crews tuesday to survey the damage after an onslaught of severe weather in mississippi…"
T5
"Das ist gut."
"not acceptable"
"3.8"
"six people hospitalized after a storm in attala county."

Figure 10: T5 framework diagram [7]

Figure 10 shows how the T5 model is trained. Different tasks such as translation, question answering, and classification, are fed into the model as input. The model is therefore trained to generate the target text. This model, therefore, allows us to use the same hyperparameters, loss functions, etc. to generate a model for various tasks [7]. T5 follows a similar training process as

BERT's masked language modeling with a slight modification. MLMs are bidirectional models, context is derived from both left and right, and T5 replaces consecutive tokens with a single mask keyword, unlike BERT which uses a mask token for each word. Figure 11 shows a representation of this [38].

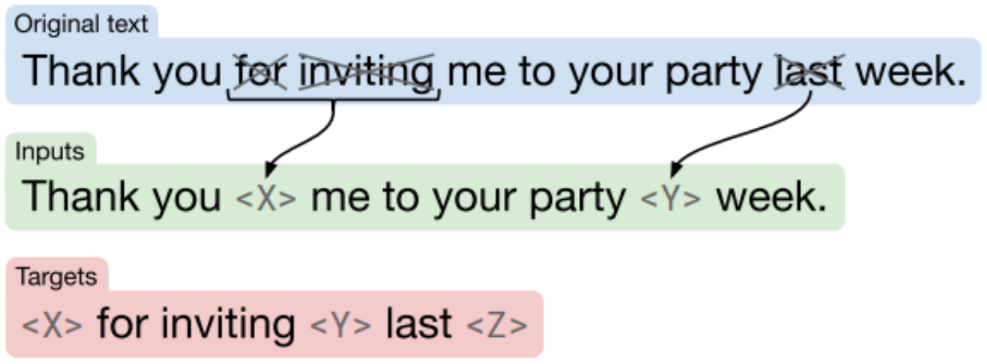

Original text
Thank you ~~for inviting~~ me to your party ~~last~~ week.
Inputs
Thank you <X> me to your party <Y> week.
Targets
<X> for inviting <Y> last <Z>

Figure 11: T5 Training [7]

As shown in Figure 11, T5 masks different lengths of tokens as part of the mask language modeling to obtain different representations of the input. This is ideal for text processing purposes such as summarization, feature extraction, etc. T5 is pre-trained on 11 billion parameters. The following are some example tasks this model can perform.

- Translate between two languages.
- Check if a sentence makes sense, such as essay revision.
- Semantic Textual Similarity, checking if two sentences mean the same thing.
- Summarization, such as generating summaries from articles.
- Keyword generation, generate keywords from a set of documents.

7.1.3 Pathways Language Mode (PaLM)

PaLM is a large language model evaluated on several natural language processing tasks and has shown great performance and generalization ability. It has been trained on 780 billion parameters to achieve great performance over a plethora of tasks.

This model was trained on the Pathways system which made the model able to train on various TPU v4 Pods [39]. PaLM has generation capabilities rather than just understanding and is specifically beneficial for categorical prediction and regression. Figure 12 shows the distribution of data the model was trained on. The various different sections make the model able to generalize in various different contexts. The PaLM model is pre-trained in an unstructured nature, the full training explanation falls out of the scope of this paper.

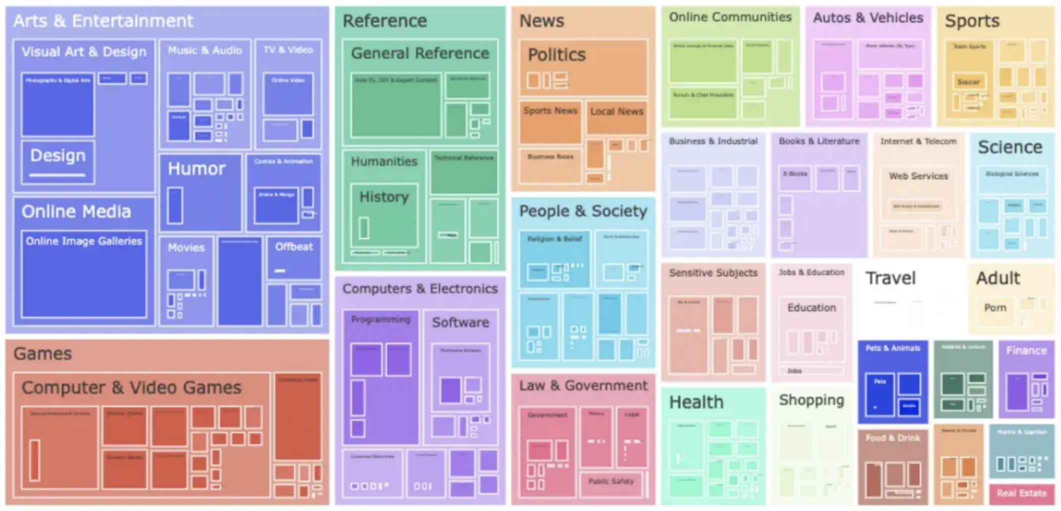


Figure 12: PaLM data distribution [8]

The following are some of the tasks it has been evaluated on.

- Reasoning, such as explaining tricks
- Multi-modal understanding, such as guessing a film from emojis
- Contextualization and scope of the text, such as distinguishing cause and result
- Question answering, such as chatbots

7.1.4 Pre-training with Extracted Gap-sentences for Abstractive Summarization (PEGASUS)

PEGASUS has an encoder-decoder architecture, the encoder takes the context of the input text and encodes it into the context vector. The decoder then decodes the context vector to produce the summary.

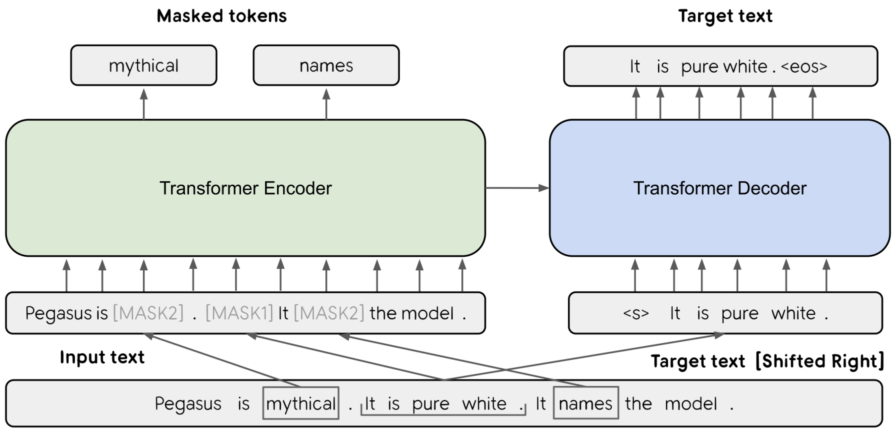

Masked tokens
Target text
mythical
names
It is pure white . <eos>
Transformer Encoder
Transformer Decoder
Pegasus is [MASK2] . [MASK1] It [MASK2] the model .
<s> It is pure white .
Input text
Target text [Shifted Right]
Pegasus is mythical . It is pure white . It names the model .

Figure 13: PEGASUS architecture [9]

PEGASUS is trained in MLM (similar to BERT) and Gap Sentences Generation (GSG). GSG selects several sentences from documents and concatenates pseudo-summaries to them. These pseudo-summaries are used as labels for training the model. PEGASUS is pre-trained on 568 million parameters. These models' skills lie uniquely in abstractive summarization.

7.1.5 Generative Pretrained Transformer 3 (GPT3)

GPT-3 is the latest version of LLMs created by OpenAI, this is a revision of the previous models GPT-1 and GPT-2. These models are pre-trained in a generative, unsupervised manner that shows good performance for zero, one, and few-shot applications. There have been a few recompilations done of what the GPT-3 can do, the following examples provide a brief overview of this model.

- Nonfiction: Dialogue, impersonation, essays, news articles, plot summaries, tweets, teaching.
- Professional: Ads, emails, copywriting, CV generation, team management, content marketing, note-taking.
- Code: Python, SQL, JSX, React app, Figma, javascript, CSS, HTML, LaTeX
- Creativity: Fiction, poetry, songs, humor, online games, board games, memes, cooking recipes, guitar tabs, writing in your unique style.
- Rational skills: Logic, uncertainty, common sense, analogies, concept blending, counting, anagrams, forecasting.
- Philosophy: Meaning of life, number 42, responses to philosophers.

Similar to most, GPT-3 follows an encoder-decoder architecture. This model is specific in that it is trained with generative pre-training, meaning it is trained to predict the next token. This model is the only one of those presented previously that does not have a readily available public pre-trained model. In order to use the OpenAI model you have to access their platform and fine-tune it using their tokens. This limits the possibilities we have for fine-tuning the model. GPT-3 is pre-trained on 173 billion parameters.

7.1.6 Transformer Summary

To determine the models we want to use. For this, we have compiled a summary table of the essential characteristics used to make this decision.

Model	Parameters	#Hidden Layers	#Attention Heads	Open Source Access	Last Update
BERT	110 million	12 [1]	12 [1]	YES	March 17th 2020
T5	11 billion	6 [1]	8 [1]	YES	December 25th 2022
PaLM	780 billion	20 [1]	20 [1]	NO	April 4th 2022
PEGASUS	568 million	12 [1]	16 [1]	YES	November 21st 2022
GPT-3	173 billion	NA	NA	NO	May 5th 2022

Table 20: State-of-the-art model summarization (own creation)

7.2 Fine-Tuning Techniques Review

Fine-tuning reviews are more complex than the previous ones, these are very specific to the type of model being handled. Therefore we concentrated on obtaining fine-tuning surveys or explorations within the scope of text summarization and keyword extraction. The most influential article, in this case, was *Tech-Talk-Sum: fine-tuning extractive summarization and enhancing BERT text contextualization for technological talk videos* [40], this article treats a very similar topic and scopes out the fine-tuning techniques in depth. The basis of fine-tuning lies in inheriting the weights of another model. This model has either been trained with unsupervised learning which already has a strong basis for the data that we will be inputting into the model. This would then classify as few-shot training. The other way of fine-tuning is taking a model trained on a specific task and using tuning this model to your, somewhat similar task. With this latter type, we are using the base of the first but making sure that the model is previously trained on a related task.

7.2.1 Input/Output Layer

The most common form of fine-tuning in neural networks is adding or truncating the input and output layers. This makes the input and output layers fit your specific situation, setting the number of neural networks as output for classification tasks, amending the input shape to the input layers, etc. This is the most common practice due to the versatility this provides to the neural network. We can fit most neural networks to most tasks thanks to this technique.

This has less effectivity with neural networks dealing with text. Text is usually truncated to a certain amount of tokens, this means that even though we would be using. The output is more the same, we will be adapting the input data but not the output format or training task. Therefore although somewhat effective, it doesn't have the same effectivity for natural language as it does for image processing or other deep learning tasks.

This technique does not generalize between models, we have to understand what each model requires as input and as output in order to properly inject or extract the information into the model. This correlates directly to the malleability of the system. In this case, it will be difficult to implement the input and output layers for our model.

7.2.2 Freezing Weights

Freezing the weight in some layers will allow us to train specific layers while maintaining the weights from others. There are a few techniques associated with it one of which is adding new layers and training only these new layers. Adapting this therefore to the pre-trained weights. Another way to do this is to freeze the input layers and calculate the weights on the output layers.

This makes sure that we are training the model to understand the desired input and output rather than understanding a more holistic scope, or overfitting to our data.

This technique is easy to generalize. Thanks to the abstraction we have created by using Huggingface. This will allow us to freeze the layers of the different models without having to specify the model that we are working with. This provides high malleability of the fine-tuning technique without much programming effort.

7.2.3 Adding and Removing Layers

While maintaining the structure and architecture of the base model we want to add specific layers or remove them in accordance with our necessities, if we see the task is overfitting to our model, we can simplify the structure while maintaining the pre-trained nature of the model and being able to build from its previous knowledge.

The addition and removal of layers can be controlled through the Huggingface library. Although each model has its own configuration, the HuggingFace library has a parent class that encapsulates the main components that can be manipulated in transformer models. In this way, the addition and removal of layers can be easily generalized to the different systems but can also

7.2.4 Transfer Learning

Transfer learning is the most intricate of fine-tuning techniques, it requires training another model and making the input for our model the output of this second model. This serves a good purpose for contextualizing our models. The current state-of-the-art technologies for transfer learning are "Using Transformer-based Sequential Denoising Auto-Encoder for Unsupervised Sentence Embedding Learning" (TSDAE) [**?**] and Generative Pseudo Labeling for Unsupervised Domain Adaptation of Dense Retrieval (GPL) [**?**], whose main purpose is to create unsupervised sentence embeddings that in turn take into consideration the context in which we are working in.

We encounter the same problem with transfer learning that we do for adding input/output layers. We have to understand what input each model requires in order to be able to ensure the correct functioning of the system.

7.2.5 Fine-Tuning Techniques

The table below is a summary of important aspects obtained in the previous descriptions. These main points will then be examined further for the decisions on which techniques to use.

Fine-Tune	**Ease of Generalization**	**Meability**
Input/Output Layer	HARD	LOW
Freezing Weights	EASY	HIGH
Adding and Removing Layers	MEDIUM	MEDIUM
Transfer Learning	HARD	LOW

Table 21: State-of-the-art model summarization (own creation)

7.3 Technique Selection

7.3.1 Model Selection

In order to complete the model selection process we took into account the following criteria.

- Open source availability of the model.
- Tasks trained on, capable of doing different NLP tasks.
- Size of the model.

Starting with the open source availability of the model, we require a deeper look into the model (how it is trained, what parameters we can change, etc.) in order to properly pan out the experiments we want to run. Referencing back to Table 20, we can see that the models GPT-3 and PaLM are not available in an open-source setting. There is a major difference between these two, PaLM is available to use on HuggingFace (the library we will be using to support the system) while GPT-3 requires using their propierty system to fine-tune the model. Therefore although there is some lack of visibility in PaLM we get free use of fine-tuning while in GPT-3 we are more subject to their restrictions. In this case, it is restricted to prompt tuning, which slightly differs from that of fine-tuning in the sense that we can only use input and output expectations to fine-tune the model. The lack of control in this model has culminated in us removing it from the list of possible models. An alternative would be to use GPT-2 which has not been updated since more than three years back. Therefore, we have decided to opt out of using the family of Generative Pre-trained Transformers from OpenAI.

The second criterion is the tasks the model is trained on. This is a big indication of how greatly the model will be able to generalize to our task. We, therefore, have to take a look at the previously exposed tasks, most models have many facets from which to pull, and the great majority have the ability to rationalize, and question answering, amongst others. However, a clear outlier in this is the PEGASUS model which has been trained specifically on abstractive summarization. Although this model might be able to capture the necessities of our system, it is a smaller model that has been greatly concentrated into one specific task.

The remaining models are BERT, T5, and PaLM, from these, we will look at the last criteria, the size of the model, and determine the best model to fine-tune for the purpose of domain-specific feature extraction. To do this, we will look at the number of hidden layers, attention heads, and parameters. BERT and T5 lie in the hundred million parameters while PaLM lies in the hundred billion parameters. In contrast, the sizes of the hidden layers and attention heads between BERT and PaLM do not variate drastically, whereas T5 is around half of the size. The drastic difference between the parameters will allow BERT to be faster at training and predicting, for the purpose of this study we require efficiency and therefore will prefer lowering costs. In this case, we can choose BERT and T5 over PaLM:

Finally, since we have identified the training base for both of these task types, changing the model for any of these other models that are trained, and used, in similar ways will require little to no effort on our part. Therefore, the extensibility and scalability provided by the system design will augment the ease of use of this fine-tuning pipeline to scope out user requirements.

7.3.2 Fine-Tuning Techniques Selection

Fine-tuning has been proven to have ease of implementation due to the currently available libraries such as HugginFace and PyTorch. This means there can be a quick implementation of each technique, refer to Table 21 column Ease of Generalization to understand the ease of implementation. However, this study was greatly limited due to the runtime of the experiments. Therefore, we will have to select which of these techniques is easier to implement. Given the libraries, we will be working with have certain facilities for the manipulation of the models. This will therefore facilitate the use of techniques that deal directly with the model implementation (e.g. freezing weights, adding/removing layers). On the other hand, anything that deals with externalizing information extraction passes this to the model (e.g. adding input/output layers, transfer learning). Therefore, due to ease of implementation, we will prefer techniques dealing specifically with the model. We have opted to move forward with the addition and removal of layers. These allow us to run several tests at once and increase the number of experiments if time allows it. The ability to run multiple tests per fine-tuning technique will allow us to increase the depth of the proposed system.

8 Laws And Regulations

8.1 Data Privacy

For the purpose of an initial analysis, we will be using data from the website AlternativeTo, therefore we will adhere to regulations of data usage within this. The data that we are using is publicly sourced (no specific user data will be used for model training) and therefore we do not have to adhere to any data protection regulations.

AlernativeTo's privacy section [41] suggests the terms of use of user data (reviews, descriptions, crowd-sourced data, etc). This ensures the data we obtain adheres to regulations and is available for use. AlternativeTo states the registration of username, full legal name, email, etc. These are not publicly visible and will not be recorded by our system. We will strictly use descriptions, characteristics, and review text. Nonetheless, we do have to consider copyright and implications on ML training in the EU. Data in AlternativeTo is not protected under any Creative Commons license or copyrighted in any form. This means it is available for our free use.

The implication of creating a pipeline for ease of fine-tuning also requires making the potential users of the pipeline aware of the regulations applied to data processing. When using public data we have to be aware of the Creative Commons license and copyright infringement norms (as seen with our use case). If the data we use is private user data, we will require a deeper understanding of regulations within the area the information was captured/used. The EU has the General Data Protection Regulation (GDPR) which protects the usage and free movement of data [42].

8.2 International Organization for Standardization (ISO) Standards

The data we are using should also be up to par, to ensure the correct processing and usage of the data we will adhere to ISO Standards. The following are the identified ISO standards relevant to this project:

- **ISO/IEC CD 5259:** Data quality for analytics and machine learning (ML)
  - **Overview, terminology, and examples** [43]: The terminology in this paper has been fine-grained and exhaustively explained to be able to adapt to this standard.
  - **Data quality measures** [44]: The data quality in our sense is done through the use of one website in order to adhere to the same standards.
  - **Data quality management requirements and guidelines** [44]: By using data from AlternativeTo, our system will adhere to these to follow their required guidelines.
  - **Data quality process framework** [45]: The framework we have created for extracting data is very simple, the data is revised at the end by a human (in this case the author) in order to ensure its quality.
  - **Data quality governance** [46]: The quality will remain stable due to the use of a single website for this project.
- **ISO/IEC TS 4213:2022** Assessment of machine learning classification performance [47]. This is done by adhering to strong regulations and running the assessments under stable conditions.

- **ISO/IEC AWI TS 17847** Verification and validation analysis of AI systems [48]. This is done by splitting the test data into train, test, and validation and ensuring its correct functioning in validation.

The standards identified to account for the treatment of data, results, and assessment of the results of our machine learning system.

9 Design and Implementation of Machine Learning Pipeline

This section will serve as an in-depth understanding of how the developed pipeline works and the functionalities that running it will provide. This will go over the preparation of the model, training, evaluation, and eventual deployment. This preparation will go through what libraries we have used for each step, how each step works, and finally the output of the pipeline. The code for the machine learning pipeline is available on GitHub [49].

9.1 Model Preparation

After selecting the models and the fine-tuning techniques that will build the base for our system we can start preparing the basis of the implementation of these models. In order to prepare the models, we did some research on potential libraries we could use to ease the process. The library best suited to fine-tune models based on their pre-trained version is Huggingface. This library allows us to extract training weights and use models stemming from a repository in GitHub. Since all of the models we will be using are Open Source, and therefore published in GitHub we will be able to use these directly from Huggingface.

The preparation of the models will be structured in classes. These classes will have as input the data we want to train on, whether we want to initialize the model with the weights from the pre-trained model and the configuration of the model. From this, we will load the model we want to work on, train it and evaluate it on a subset of the input data. The expectation of the model is that the data is pre-processed previous to the execution of the model for which we will create a separate class. The following diagram shows the data flow for preparing the data previously to training or evaluate the model.

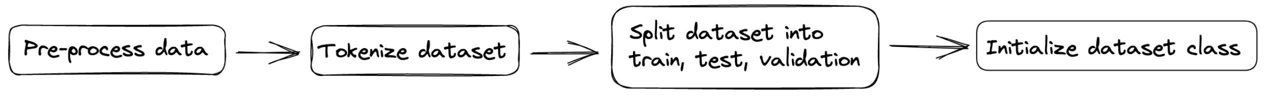


Figure 14: Data pipeline for model initialization (own creation)

This initial process will make sure that the data is ready to be consumed by the model. The dataset class we have created is a child class of the PyTorch DataLoader class. This will facilitate the incorporation of the data for training and testing. This class will use the tokenizer (BERTTokenizer or T5Tokenizer) which will stem from a pre-trained version of the models. This will serve to transform the data from a string representation to a numerical representation that the model can interpret. The model outputs the data as a numerical representation, this tokenizer will also serve as a decoder used to obtain the string representations of the features our model outputs.

Once this initial preparation step has been done, we can initialize the training of the model with our training dataset. To do this we have incorporated the Trainer class from the transformers library. A deeper exploration on the arguments used for training will be explored in the following section. The trained model will then be evaluated with the validation dataset to check the metrics of the model in circumstances it hasn't previously seen. These metrics will let us know how well the model generalizes to unseen data and will therefore give us better insight in how well the models would preform in real-life circumstances. The final training pipeline for an individual model will be the following:

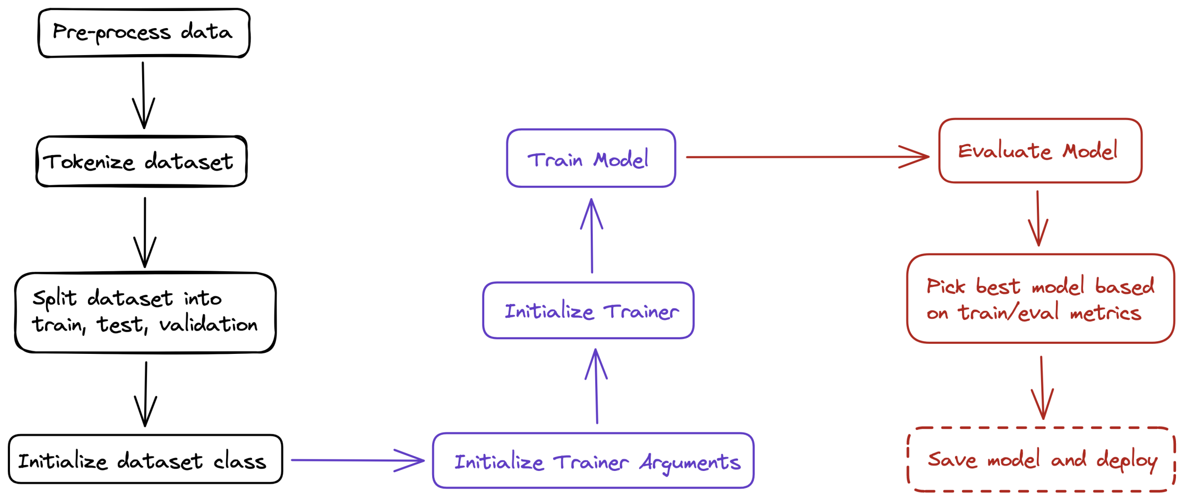


Figure 15: Model training & testing pipeline (own creation)

The final step in this process is to generate the pipeline from which we will be able to extract the best model and configuration. This will be done by parallelizing the models that are currently implemented (BERT and T5). We will run the pipeline in figure 15 for both models and from the final metrics extract the best. Therefore, abstracting the pipeline of training and testing the model, Figure 16 represents the fine-tuning and selection pipeline in our system.

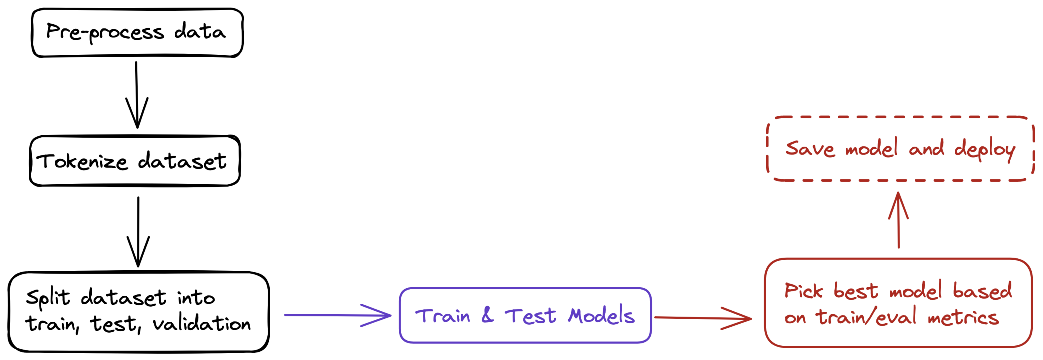

Pre-process data
Tokenize dataset
Split dataset into train, test, validation
Train & Test Models
Pick best model based on train/eval metrics
Save model and deploy

Figure 16: Fine Tuning Pipeline for Model Selection (own creation)

For the purpose of ease of deployment and use the pipeline can easily be deployed to a Google Cloud Virtual Machine instance [50]. This will ease the complications in training and make the pipeline process faster. In our case, we were using a very small amount of data, but this might differ based on the requirements of the system and the amount of data we are using to train. In general, when training we will require the following:

- A multi-threaded CPU.
- Memory to fit big batches of data and model's weights.
- Fast storage (solid-state) to store and read the dataset.
- GPU (Graphics Processing Unit) with enough RAM for a big batch of data and the model's weights.
- Fast network for distributed training.

These requirements are generally not found in at-home setups. Generally, training on at-home GPUs is rare and usually requires a vast amount of technical knowledge. Cloud computing comes in to provide these services, allowing us to train and access the model from anywhere. The specific setup for our virtual machine will be a 10 GB Ubuntu 20.04 with the specifications mentioned in Table 15. This is, generally speaking, a large enough machine to fine-tune the transformer model.

A general specification of fine-tuning transformer models that have been pre-trained on a large dataset of data is that the dataset we fine-tune has to be kept small. This is because these models learn pretty quickly how to do a specific task. Taking into account that these models have been pre-trained in massive amounts of data, they pick up the task at hand with relative ease. Too much training will make these models grossly overfit the data. This strengthens the importance of validating the model on unseen data.

9.1.1 Data preparation - BERT

Due to the nature of the tasks at hand, we will have to prepare the data in two different ways for the different models. BERT will be trained on a classification task, which means we have to generate the data by matching correct and incorrect features and labeling them accordingly. The following is the simple algorithm we used for generating this dataset. Sentence a will be the description of the application, sentence b will be the resulting expected facets and the label will be either 0 or 1, 0 representing that the facets in sentence b are not correct for the given description and 1 representing that the facets in sentence b are correct for the given description.

1. Initialize empty lists to store sentence a, sentence b, and the labels.
2. For half of the examples we match the correct corresponding features to the description. Any example that falls under this category will have the label 1.
3. For the rest we will match it to any other random features from any other description. Any example that falls under this category will have the label 0.

This data could have been generated in a vast amount of ways, the decision for this study was to make it as simple as possible to see how well the model could understand basic next-sentence prediction based on the description. There are different ways the BERT model could be trained. Another possibility would have been training the task on masked modeling, masking the last word, and attempting to predict one of the input features as the next word. This is a harder task to learn. This option will be further explored in the Future Work section.

9.1.2 Data preparation - T5

On the other hand, T5 has a simpler form of data preparation. To prepare the data to be inputted into the T5 model we have to tokenize the input and the expected output. From this, we will extract the input ids and the attention mask for both the input and the expected output. We will label these according to the model input restrictions. In short, the following algorithm will provide the desired input for the T5 model:

1. Tokenize descriptions.
2. Tokenize features.
3. Set input_ids as the input ids from the tokenized descriptions.
4. Set attention_mask as the attention mask from the tokenized descriptions.
5. Set decoder_input_ids as the input ids from the tokenized features.
6. Set decoder_attention_mask as the attention mask from the tokenized features.

9.2 Model Training

As mentioned previously, we will be using the TrainingArguments and Training classes from the transformer library to abstract the process of training and evaluation. Previously, we had explained how the training process works for deep learning models. These classes will allow for the implementation of complex training strategies while simplifying the process of implementation and allowing for a more accurate training process. In this section, we will specify the training decisions, explain these and explain how these are portrayed in the code.

9.2.1 Evaluation Strategy

The evaluation strategy specifies how often we evaluate the model. For the TrainingArguments, we will be using the IntervalStrategy class. This class distinguishes between three different evaluation strategies "no", "steps" and "batch". The "no" strategy signifies that the model will not be evaluated until the training is finished. The "steps" strategy consists of evaluating the model every X steps (batches), where the steps are determined by the user. Finally, the batch strategy consists of evaluating the model every batch feed through the model. To consider which strategy is better we have to consider how the model will be evaluated and tuned. As explained previously, to train the model we are using the Gradient Descent strategy therefore the evaluation at different steps will cause different movements of the gradient space. The following diagram demonstrates how gradient descent responds to each strategy.

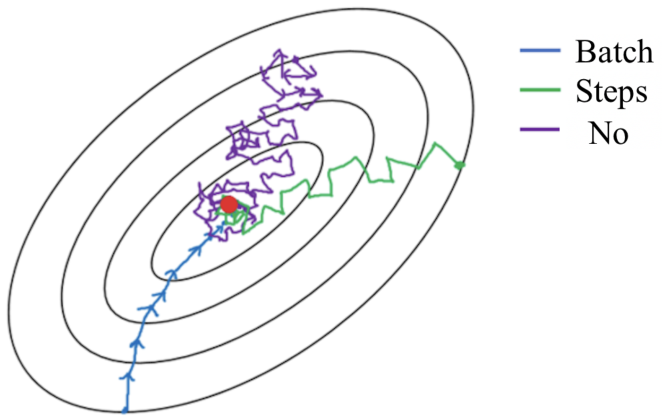


Figure 17: Evaluation Strategies Visualization [10]

Figure 17 shows that the batch strategy performs the best, leading mostly in the correct direction and quickly getting to the local minima. The steps get somewhat deviated but get to the local minima faster than evaluating after training. In our case, we have to pay attention to the evaluation to avoid overfitting the training data. Therefore, we should work with the batch or the steps strategy to ensure we aren't overfitting. In terms of computational complexity, having to calculate the evaluation of every batch is more costly than evaluating every x step. For our purpose and limitations, it will therefore be better to start with steps and give the user the option to run batch evaluation instead.

9.2.2 Callbacks

"A callback is a set of functions to be applied at given stages of the training procedure. You can use callbacks to get a view on internal states and statistics of the model during training" [51]. There are various callbacks our system will implement, we have to take into consideration that adding callbacks will add computational complexity and increase the memory storage required by our system. The following list contains some of the most common callbacks.

- Early Stopping: This function attempts to stop training whenever the loss significantly increases, locating in this way the local minima without having to pre-determine the number of epochs we would like the model to train for.
- Model Checkpoint: Stores the after every epoch.
- Learning Rate Scheduler: The scheduler adjusts the learning rate over time using a schedule, it returns the desired learning rate based on the current epoch.

In our case, we require the handling of hyper-parameters without having to implement systems that will further increase our training and testing time. Therefore, we will implement the early stopping and learning rate scheduler to simplify this process for us.

9.2.3 Epochs, Batch Size & Learning Rate

Other important factors to consider are the epochs, batch size, and learning rate. The epoch refers to an entire passing of the training data through the model, in our case we have determined that the epochs will be controlled by early stopping. We have to set a maximum amount of epochs that we will allow our model to train for. This will make sure that we are not over-computing in any specific set of the model. The fact that we have early stopping allows us to increase the number of epochs while still being able to assume that we will be stopping earlier. Therefore, we will set the maximum amount of epochs to 200.

The batch size determines the number of samples to work through before updating the internal model parameters. This parameter will be overshadowed by the evaluation strategy proposed above. Therefore we will set this to a standard 64 and we will leave the evaluation strategy to change these.

Finally, "the learning rate is a hyperparameter that controls how much to change the model in response to the estimated error each time the model weights are updated" [52]. Choosing the learning rate is a critical task for the training process. Due to this reason, we have decided to implement a learning rate scheduler callback. There are a copious amount of learning rate scheduler strategies that we are able to choose from. Figure 18 gives a comprehensive study of the different learning rate scheduler techniques.

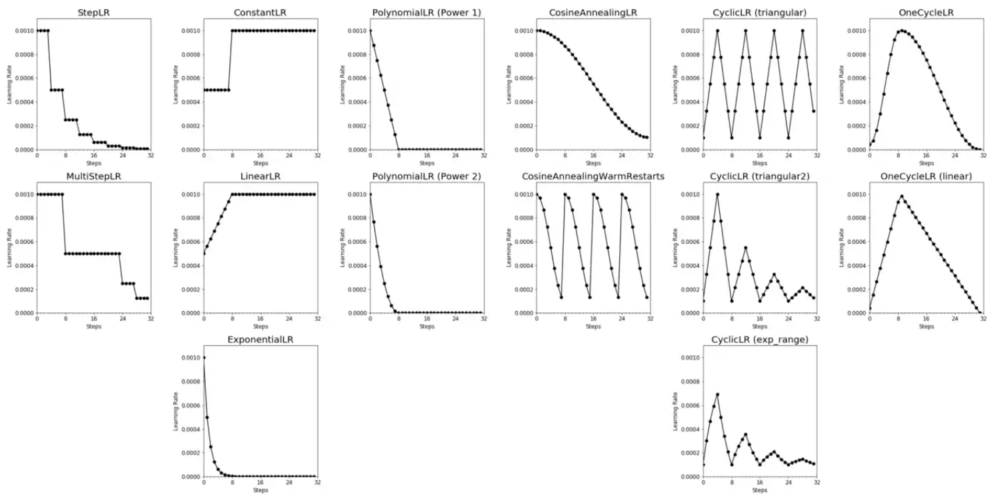


Figure 18: Learning Rate Schedulers [11]

Due to the limited amount of time, we had to develop this project, we have decided to go with a stable learning scheduler that has shown great promise in a great number of projects. For this purpose, we opted to go with the StepLR scheduler. This scheduler takes in a multiplicative factor which will reduce the learning rate at every iteration.

All of these values will also be controllable through user-handled input. The values mentioned above will provide default values and those necessary for determining the features in applications.

9.2.4 Metrics

The metrics for our model will vary depending on the type of model we are using. We will be training on two different types of tasks for the two models we are proposing. This means that we will have to figure out which metrics we will be selected based on the training type. The basic metric calculated by the Training class in HuggingFace is the loss. We will be using this for selecting the best model.

The BERT model will be trained on next-sentence prediction, a classification task. Therefore we can obtain some metrics set for classification tasks. These are described below:

- **Logarithmic Loss:** This loss works by penalizing false classifications and can also be very efficient for multi-class classification. Log loss lies in the range [0, ∞). It has been proven that using log loss as an evaluation metric increases the accuracy of the classifier. The following is the formula for calculating the Logarithmic Loss: $\frac{-1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} * log(p_{ij})$. Where y_{ij} identifies if the sample i belongs to class j and p_{ij} is the probability the model outputted for sample i belongs to class j.

- **Accuracy:** The number of correct predictions divided by the total amount of samples. $Accuracy = \frac{num.of correct predictions}{total num.samples}$.

- **Precision:** The number of correctly identified positive results divided by the number of positive results outputted by the classifier. The formula for precision is the following: $Precision = \frac{TruePositives}{TruePositives+FalsePositives}$, true positives are constituted by any data point tagged as positive that has been identified as positive by the model. False Positives are those data points that have been tagged as negative by the model but are actually positive data points.

- **Recall:** The number of correct positive results divided by the number of true positives and the number of false negatives. The formula for the recall is the following: $Recall = \frac{TruePositives}{TruePositives+FalseNegatives}$.

- **F1:** The F1 score is a metric that lies in the range [0, 1] which identifies how precise the classifier is. In short, it is the harmonic mean between the precision and recall of our model. The following is the formula for the F1 metric: $F1 = 2 * \frac{1}{\frac{1}{precision}+\frac{1}{recall}}$.

The next model we will be training for is the T5. This is a text-to-text model and will be training on the same. Therefore, we have to select different metrics to analyze this model. These text-to-text models are increasingly complicated to evaluate, this is because most evaluation systems have been proven to hallucinate information, therefore providing an unreliable base for the prediction. This has been greatly explored for similar tasks such as summarization, machine translation, etc. This goes hand in hand with the evaluation method of both models. We have obtained an initial look into which metrics we can use for text-to-text generation.

- **Recall-Oriented Understanding for Gisting Evaluation (ROUGE):** The ROUGE metric calculates the syntactic overlap between the desired outputs and the model outputs. This metric is very popular for summarization tasks. It is calculated as ROUGE-N, where N stands for the words of overlap we want to calculate. In our case, we would be interested in Rouge-1 and Rouge-2 since features tend to be between one and two words. [53]. From this, we can also extract R1-precision and R1-recall.

- **Bilingual Evaluation Understudy (BLEU):** This metric is usually used in machine translation tasks, this metric compares the model's output translation to one or more source translations. Although the specific case presented in this paper does not make for a good metric, in the case we decided to expand the sources where we obtained the features it would make for a good matching between the different sources. [54]
- **Cosine Similarity:** Cosine similarity is a very basic calculation that can be made to check semantic similarity between phrases. This uses vector spaces to calculate the distance between the different words in the phrase and extract a value from the embeddings generated by a generic sentence transformer. From this value, we can extract Precision, Recall, and F-score.
- **BERTScore:** "Unlike most of the methods that majorly make use of token or phrasal level syntactic overlaps between hypothesis and reference text pieces, BERTScore, on the other hand, captures the semantic aspect by using the contextualized embeddings generated by the BERT model." [54]. To do this, we generate contextualize embeddings for each word in the model output and expected output. From this, we use cosine similarity to generate a comparison metric.
- **Word Mover's Distance (WMD):** This method calculates the distance between a word from the model output and from the expected output. Similar to many used for these tasks, it is used to calculate the semantic similarity between the two outputs. The computational complexity grows exponentially with this method, which means that it works very well for shorter inputs but is very slow for larger pieces of text [54].

Although initially we thought about using cosine similarity due to it's versatility in understanding, we opted to go with ROUGE. This metric was very easy to implement and has been widely used for experimentation in similar areas. It also provides comparison metrics for precision and recall. We would otherwise be missing these metrics for the text-to-text models. On the other hand, ROUGE is a metric that uses n-grams to check the overlap between the model output and the expected output. In terms of this paper however, we preferred the ease of comparison provided by the precision and recall metrics, and the explainability that they introduced to the system, therefore we prefered ROUGE over cosine similarity.

9.3 Model Evaluation

The model evaluation will be based on the text-to-text transformation of the system. There are various things to take into account when considering the evaluation of the models. In an ideal case, we would compare the metrics for training, testing, and evaluation. This is somewhat complicated to achieve given the difference in training from our system. We will therefore have to take care of the evaluation of these systems individually and combine them once an initial choice has been made.

Initially, we will have to select the model individually between BERT and T5. To do this we will have to rely on heuristic measures to automatize the process. In order to ensure that the user has complete control of the system, the user will be able to disable the heuristic implementation of model choice and manually enter the configuration and model they prefer. The implemented heuristic measure makes sure that there isn't overfitting or underfitting to either the training or evaluation data by taking into account the differences between these and setting a maximum difference between these. In our case, we will allow for a 10% increase or decrease between the training and testing losses.

Once the overfitting or underfitting models have been discarded we will extract the best-performing configuration for each model type. To do this we will select the model with the lowest average loss between the training, testing, and evaluation. Another possibility would be to choose the model with the lowest disparity between the losses, ensuring the generalization of the model. Finally, once we have the best-performing models from each model type we can select the best-functioning model from the text-to-text evaluation metric.

Using a common testing sample we will extract the text-to-text evaluation method and obtain the lowest value for either cosine similarity or ROUGE. This will ensure an equitable comparison of both and the final value extracted. The data we will evaluate will not have previously been seen by any of the models, ensuring that the metrics extracted are those that would appear in real-world use of the model.

9.4 Model Deployment

The development of the model essentially culminates in a usable model, in order to be able to extract predictions from this model, we will want to deploy the model. In order to follow the already implemented system we have opted to deploy the final model on Google Cloud. For this, we require a Cloud Storage bucket, and in this bucket, we can export the best model by saving it into the bucket. It is then externally accessible with the use of a simple command.

The model can also be uploaded to HuggingFace. This is thanks to the use of the Trainer class, which enables the uploading of the best model by setting a simple field in the Trainer class. The development of the HuggingFace models and platforms also allows us to pick the best model and uploaded it after we have selected the model we want to use. This option allows for external use within any system and free storage of the model. It also provides the abstraction of the process for the user, which will simplify the use of the model within any platform.

10 Experimentation

10.1 Experiment Set Up

Once the models are set up and the fine-tuning techniques are ready to be used, we have to set up a generic experimentation plan and then observe which are the best possible experiments with the limited amount of time and resources that we have.

10.1.1 General Experimentation Set Up

In order to generate a general experimentation set up we have to consider which layers we will be able to add/remove. In order to allow for generic experimentation we will create a class function that will take in the generic parameters for the system will run all experiments and return the metrics for every experiment run. The consideration here is that the more experiments we run the larger the memory usage will be, therefore the user should be aware of what the limit of their system is and decide what type of experiment to run based on that.

For the BERT model, we have identified the following parameters to fine-tune by:

- Number of hidden Layers (defaults to 12).
- Number of attention heads (defaults to 12).
- Dropout probability in hidden layers (defaults to 0.1).
- Dropout probability in attention heads (defaults to 0.1).

The default values constitute the initial architecture of the model. Any extra layers added to these configurations represent adding previously unseen layers to the model, otherwise, we will be removing layers. While removing layers we can still obtain the pre-trained weights any added layers will have to be trained from the beginning.

The T5 has the following parameters:

- Number of hidden layers in the encoder (defaults to 6).
- Number of hidden layers in the decoder (defaults to 6). If the number of encoder layers changes, the decoder layers will default to the value of hidden layers in the encoder.
- Number of attention heads for each encoder layer (defaults to 8).
- Dropout rate (defaults to 0.1).

From these, we can identify our experimentation technique. In our case, we have opted for granularizing the possible fine-tuning parameters to allow for the user to fine-tune and select the models over smaller computer architectures. Therefore, for every model, we set up an individual testing process for each parameter. To enable easier testing for those people able to run the experiments on larger machines, we will also set up an experiment method for changes of layers in the model and a generic experiment to test all possible parameters.

10.1.2 Feature Extraction Experimentation

The experimentation for feature extraction, the specific case treated throughout this paper, will be a smaller portion of the experimentation cases developed for the pipeline. This will be contained specifically to the resources and time available for this project. As we can see in the "Modifications with Respect to Initial Planning" due to various different unexpected incident we had to reduce significantly the initially allotted time for the experiments.

As explained in the selection of fine-tuning techniques we will be experimenting with the addition and removal of layers as a fine-tuning techniques. In terms of the previously set up experiments, we will be running all hidden layers with all possible attention heads. For the purpose of proper comparison in the T5 configuration we will always set the number of hidden layers in the encoder and the number of layers in the decoder to the same number. In this way all the experiments run in BERT can be replicated in the T5 model, therefore establishing a base of comparison.

As mentioned previously, we have some pre-defined parameter tuning by using callbacks. This will simplify the amount of times we have to run the code as well as the duration of the tests themselves. This simplification will allow our tests to run quickly and iterate the least amount of times possible throughout the process. In order to ensure the extraction of proper metrics, we will be running each experiment, for each model, three times.

For both models, the experiments will follow the format shown below:

1. Set number of hidden layers to minimum number of layers desired.
2. Set number of attention heads to minimum number of attention heads desired.
3. Run fine-tuning
4. Run evaluation with text-to-text metrics
5. Augment number of attention heads by one
6. If the number of attention heads reaches the maximum, increase number of hidden layers and set number of attention heads back to minimum number of attention heads desired.
7. Repeat instructions 3 to 6 until we reach the configuration with the maxiumum number of hidden layers and attention heads.
8. Repeat items 1 to 7 three times and take the average of all results.

To understand the metrics and the importance of fine-tuning rather than training a model from the start, we will train the model architecture from scratch and we will train the model architecture from the pre-trained model. This will allow us to have a better insight into the effects of fine-tuning in previously unseen tasks.

10.2 Experiment Review and Results

Having established the scope and limitations of our experiments we can run the experiments and extract the results. From these, we can extract the pertinent conclusions and establish which transformer model works best for our specific use case. In the previous section, we explained the process of implementation of the tests, therefore the model experimentation will be split into training models and fine-tuning the models. All obtained values from experimentation are appended in Annex B.

10.2.1 Training BERT Model

In order to set the base for the BERT model we will be training the model from scratch and extracting the metrics to compare with the fine-tuned BERT model. We ran these experiments with the following layer conditions:

- **Number of Attention Heads:** Range [1, 12]
- **Number of Hidden Layers:** Range [1, 12]

Figure 19 reflects the cross entropy loss for training and evaluation.

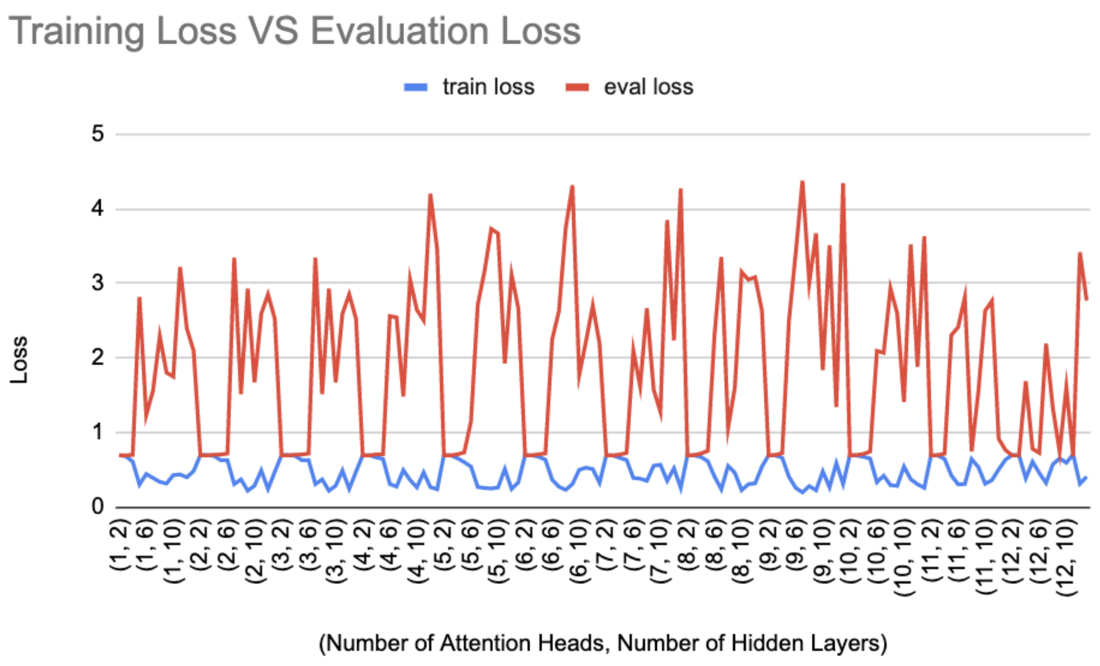


Figure 19: Training BERT Training and Evaluation Loss (own creation)

There are two major factors to discuss for this model. The first one is the disparity between the training and evaluation loss when the number of hidden layers is high (more than 2) and the other is the metric when the hidden layers are low (less than 2).

The disparity between the training and evaluation loss is easily explained by overfitting. This could be one of two things: the model learns the task at hand with such ease that it can easily learn the task at hand and in the following iterations of the learning process will overfit to the training data provided. On the other hand, this could be that the task at hand is very complicated for the model to understand and it is not learning to properly generalize to the input data. This latter option seems more adept for our situation.

We can further analyze this concept by using accuracy, precision, and recall. The metrics for training and evaluation are shown in figures 20 and 21 respectively.

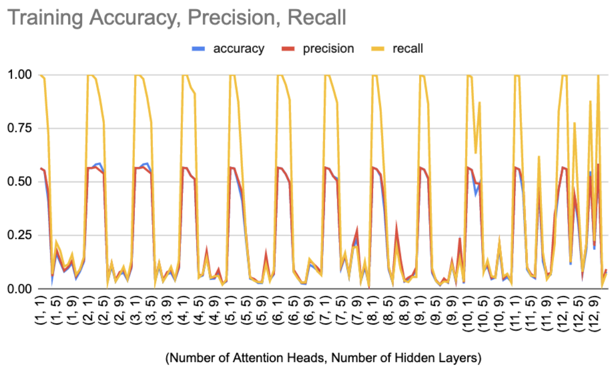


Figure 20: Training BERT Training Metrics (own creation)

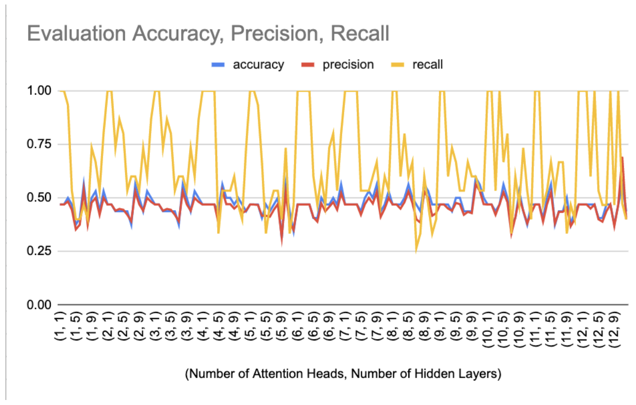


Figure 21: Training BERT Evaluation Metrics (own creation)

These graphs correspond to the accuracy (total correct predictions over the total number of samples), precision (true positives over true positives and false positives), and recall (true positives over the sum of true positives and false negatives). We can clearly see a very high variation of these data points. However, looking at the training metrics we can clearly identify a pattern between increasing the number of hidden layers and the metrics rising/lowering accordingly.

As mentioned previously, the loss for both training and evaluation seems to be very low when the number of hidden layers is very low. When taking a closer look at the metrics we can see that when the number of hidden layers (irrespective of the number of attention heads) lies in the range [0, 3] the accuracy and precision lie around 0.5, and the recall around 1. If we pay attention to the previously stated functions we get the following conclusions:

- **Accuracy:** Around 50% of the total number of predictions are computed correctly. When we think of the data we have generated for this model 50% of the data is labeled as 'true' and 50% of the data is labeled as 'false'. Therefore, to get an accuracy of 50% the model can learn to generate a prediction of either 'true' or 'false' regardless of the input.
- **Precision:** The precision for this range tends to be around the same value as the accuracy. The amount of true positives over all of the positives is around 50%. This means that the amount of positives that our model generates correctly is around half of the total positive predictions of our model. Given that our initial dataset is split halfway into positives and negatives, we can assume that our model is only predicting true labels.
- **Recall:** The recall in this range tends to be high values close to 1. Referencing the previous formula, this recall is stating that the number of true positives is equal to the sum of the true positives and false negatives. From this, we can conclude that our model does not predict any false negatives.

As we can see, in the lower ranges of hidden layers learn to predict the true label to optimize the classification by selecting true for all the labels the models have to generate. This distribution is also seen in the evaluation metrics graph. For the lower ranges of hidden layers, maintain a very similar pattern. However, as the hidden layers increase the accuracy, precision, and recall fall drastically. In these values, we can see an increase in the disparity between the training loss and the evaluation loss, we can also see that all the metrics observed oscillate over low ranges [0, 0.25]. However, the loss associated with training still oscillates over low loss values. Once again we can tend to our metric definitions to understand these outputs.

- **Accuracy:** Low accuracy shows that the total correct predictions are very low compared to the number of samples (e.g. when the data point is true it constantly identifies false and vice-versa). This is a clear indicator that the model is not correctly understanding which features are extracted from which segments of text.
- **Precision:** The precision contemplates the number of true positives over the total amount of true and false positives. The low value of this metric lets us know that there are a large number of false positives. This means that our model is generating positive labels for negative data and negative labels for positive data.
- **Recall:** The recall identifies that there are a large number of false negatives.

From this, we can extract that the number of true positives identified is limited to a very specific set of examples. This can mean that the model has adjusted itself to a very specific set of examples. However, the evaluation accuracy, precision, and recall stay around the 50% identifying therefore that although the model has somewhat low metrics it can do well in unseen situations.

Taking into account that this architecture is very large and the input was a very small subset of examples, it is natural for the neural network to identify very concrete patterns specific to that data.

10.2.2 Fine-Tuning BERT Model

Next, we will fine-tune the BERT model by training the model from the pre-trained weights. Doing this will ensure that the model has previously learned different tasks and obtained unstructured knowledge of various forms. The pre-training of this model should allow it to be able to understand the task at hand to a greater extent than the trained model. Figure 22 shows the training and evaluation loss metric for the pre-trained model.

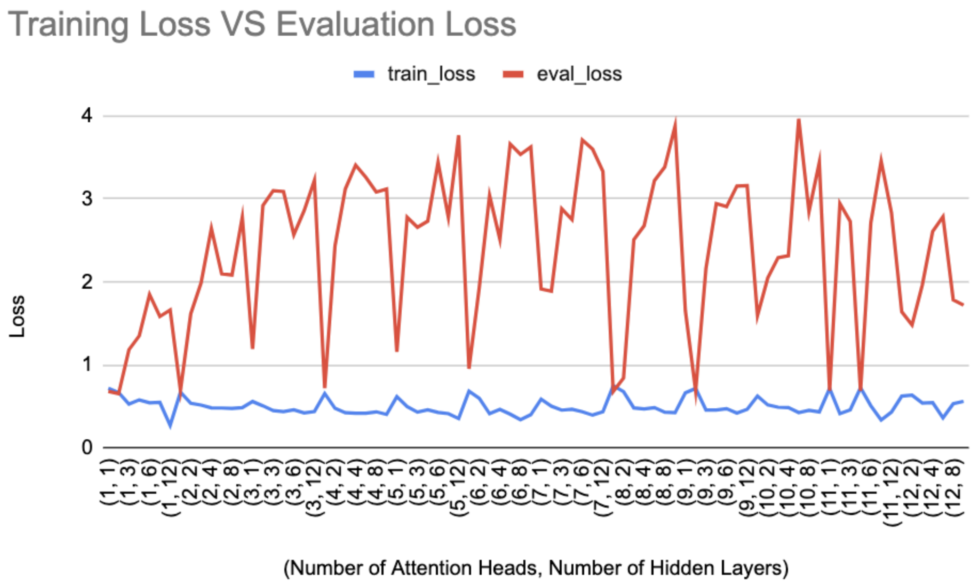


Figure 22: Fine-Tuned BERT Training and Evaluation Loss (own creation)

There is a clear difference between the pre-trained model (Figure 22) and the non-pre-trained model (Figure 20). We can see that the training loss has considerably improved. It has mostly flattened with small peaks and valleys throughout the different architecture. Although not apparent, we can see that the same thing has happened with the evaluation loss. The loss, previously bounded by a maximum loss of 5 has flattened to a maximum loss of 4. The same peaks and valleys remain seen but with greater fluctuations over smaller architectures. These fluctuations encase the ability to understand the data and task at hand that the model has. To make these data points more apparent, we can again take a look at the metrics for evaluation and training of the fine-tuned model. Figure 23 and Figure 24 encase the accuracy, precision, and recall of training and evaluation respectively.

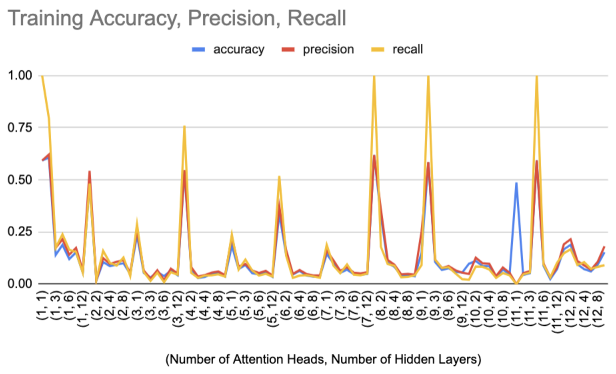


Figure 23: Fine-Tuning BERT Training Metrics (own creation)

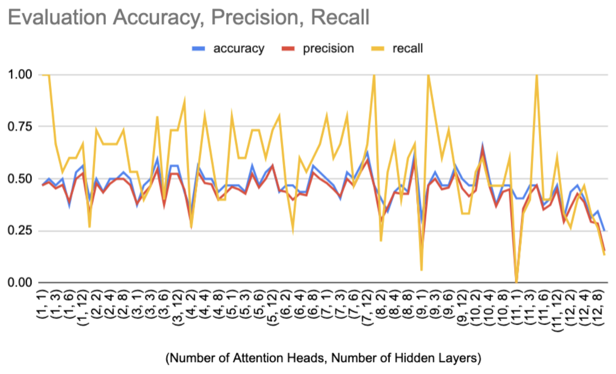


Figure 24: Fine-Tuning BERT Evaluation Metrics (own creation)

These graphs share many characteristics with Figure 23 and Figure 24. We can see the same patterns, somewhat stagnated for the training accuracy, precision, and recall. The accuracy, precision, and recall metrics have faltered down in smaller architectures, reflecting a deeper understanding of the input with a smaller amount of resources. In the evaluation metrics, we see slight changes. The accuracy, precision, and recall are more focused around the 50% point. In other words, the 50% precision and accuracy rate indicates that the model is correct 50% of the time, whereas the 50%

recall rate indicates that it properly identifies 50% of the expected output features.

These metrics indicate that the model is learning to understand the data and understand the key characteristics that relate the input data to the desired output features. The pre-training of the model has enabled a deeper understanding of the input data.

There is clear evidence of improvement from training the model from scratch to the pre-trained model. The pre-trained nature of fine-tuning allows smaller architectures to provide a better understanding of the input data. This ensures the model does not overfit the training data and is able to understand and generalize with greater ease.

10.2.3 BERT Result Discussion

Although there are great improvements concerning the BERT model trained from scratch, there are clear lagoons with both the trained and fine-tuned models. Various factors could have played a part in the functioning of this model.

One related factor can be the complexity of the task due to the type of data extracted. The features obtained were crowd-sourced from the AlternativeTo website. It is important to differentiate between abstractive and extractive. Abstractive tasks abstract the knowledge in the input text to generate the desired output, this desired output may or may not be contained in the input. An extractive task uses the input to generate the expected output.

Results obtained with BERT have shown that it learns to generalize very well over extractive tasks but is not able to reach the same metrics in abstractive tasks [55]. The nature of our data implies that we will be using an abstractive task. In this study, BERT has been proven to have a significant disadvantage.

The main reason for the incongruencies that BERT generates during abstractive tasks is the hallucination aspect of the model. As mentioned in the paper *On Faithfulness and Factuality in Abstractive Summarization* by Google researchers, ”these models are highly prone to hallucinate content that is unfaithful to the input document” [56]. Due to the limited scope of our fine-tuning dataset, the hallucination aspect might add context to the data that confuses the model. This aspect is therefore counterproductive for our task, focusing this model on a specific context will allow more specific hallucinations and therefore further confuse the input data.

A potential solution to this would be training the model in different formats. The task we are currently training on is a binary classification of the features, identifying whether features fit with the input text. This general contextualization might increase the hardship the model has to correctly identify the output. However, as previously seen, this model can also be trained using masked language modeling. In this case, we could use the next-word prediction until a stop token was found. The word-to-word nature of this training task would force the model to understand and manipulate the input data.

10.2.4 Training T5 Model

In the previous section we introduced the T5 model, this model is pre-trained on various text-to-text tasks. Therefore, we will be training this model on next-word prediction, expecting it to obtain the fine-tuned features. This initial graph shows the loss for the T5 model trained from scratch.

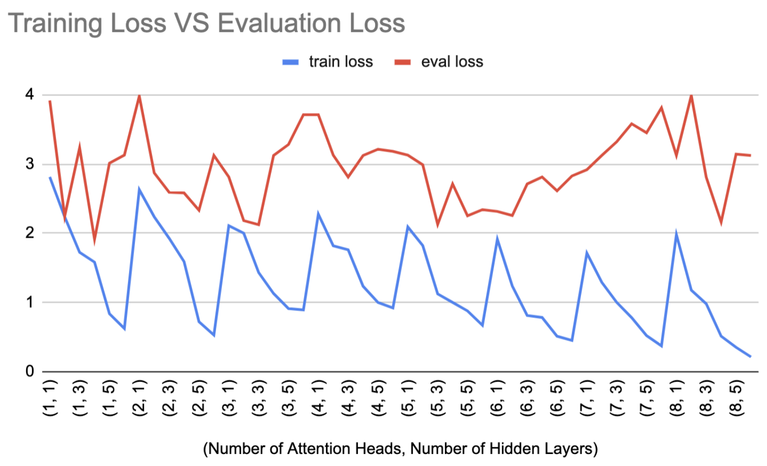


Figure 25: Trained T5 Training and Evaluation Loss (own creation)

The loss associated with the T5 training shows a clear pattern of decreasing losses while increasing architectures. However, the evaluation loss is quite erratic and does not follow the increases and decreases of the training loss. In certain cases, such as those of smaller architectures, the model cannot generalize past the scope of the training data. This is due to the fact that the model does not understand the main characteristics of the training data and is unable to reflect these on unseen data. However, as the architecture increases we see that the evaluation loss also increases, this might be due to overfitting our model. The latter architectures are very large with a small amount of training data. Therefore, it learns to understand very fine characteristics that the training data suggests to greatly improve the training metrics. Due to the capacity of the model, this decrease in the training metric does not reflect in the evaluation metrics.

As mentioned previously, the Rouge-1 score allows us to extract metrics for precision, recall, and F1. ROUGE-1 precision can be computed as the ratio of the number of unigrams in the expected input that can also be found in the model output. ROUGE-1 recall can be computed as the ratio of unigrams in the model output that also appear in the expected output. Finally, F1 is maintained as the harmonic mean between precision and recall. To further understand the model we will have a look at the precision and recall, Figure 26 and Figure 27 show these metrics for training and evaluation.

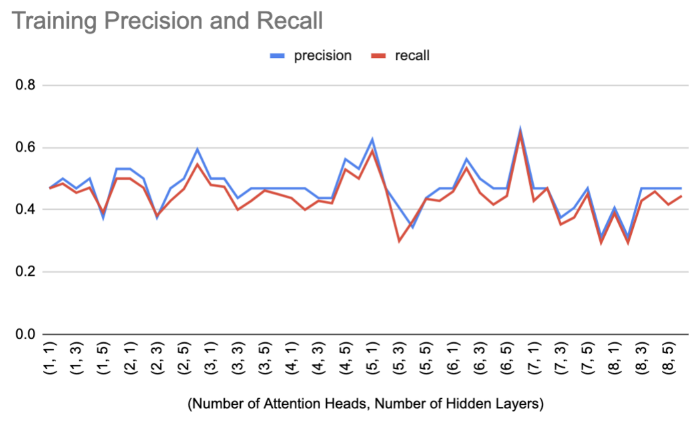


Figure 26: Trained T5 Training Metrics (own creation)

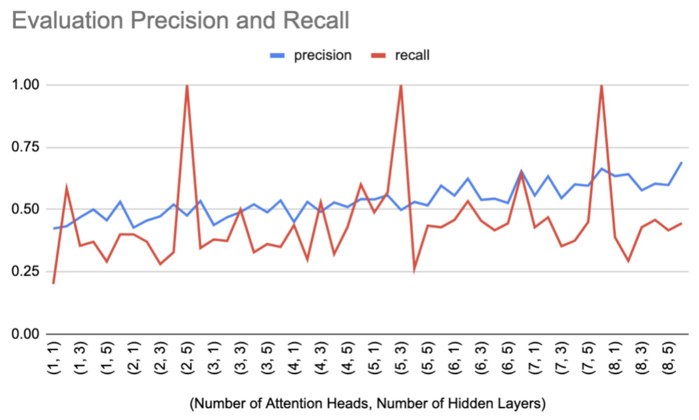


Figure 27: Trained T5 Evaluation Metrics (own creation)

We can clearly see the difference between the training and evaluation metrics. The training data follows a clear pattern, as the architecture increases the precision and recall increase with it. We can see there is very clear evidence of this with the increase in the number of hidden layers, when the amount of hidden layers drops the metrics tend to decrease with it. On the other hand, the metrics associated with the evaluation of the model tend to be erratic. The behavior associated with this, as explained in the BERT section is consistent with overfitting.

It is important to note that due to the early stopping metric that we have set, most overfitting is not caused by the train time the model has gone through but rather by the data and training task that it tries to comprehend. There are key characteristics in the training data, that is not found in the evaluation data. When training the model from scratch (without using the weights of the pre-trained model) this is due to a low amount of examples. Since it doesn't have a base understanding of previous text it generates all its assumptions from the input examples. To properly understand the text and extract proper knowledge from it, a model would require upwards of a thousand examples to extract proper knowledge. However, this step can be avoided with by pre-training the model with unstructured data and fine-tuning it to the specific task.

10.2.5 Fine-Tuning T5 Model

In the same way that using a pre-trained model adapted the learning that BERT had, we expect the T5 model to be able to easily generalize and quickly learn the task at hand. T5 has had great results in summarization [57] and key phrase generation [58] tasks in previous experimentation, therefore we expect the behavior to be similar for feature extraction.

For this study, we will get the weights of the pre-trained layers and modify the architecture of the T5 model by removing and adding layers. Due to the size of the T5 model and the number of parameters that it has been pre-trained on we are not able to increase the number of attention layers and the number of hidden layers to the same extent as the BERT model.

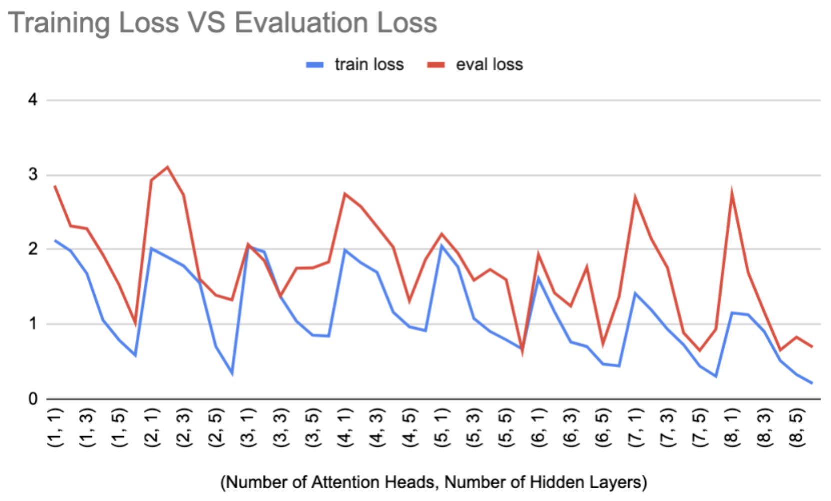


Figure 28: Fine-Tuned T5 Training and Evaluation Loss (own creation)

Figure 28 shows the training and evaluation loss corresponding to the fine-tuned T5 model. We can clearly see the difference between Figure 26 and Figure 27. The training loss has significantly decreased and the evaluation loss follows the same trend as the training loss. Therefore, using the pre-trained model has allowed our model to properly fit the data and understand the given patterns to generalize to unseen examples. This model can therefore understand the context and extract the desired features from the description. To further evaluate this model, Figure 29 and Figure 30 contain the ROUGE-1 recall and precision for the fine-tuned T5 training and evaluation.

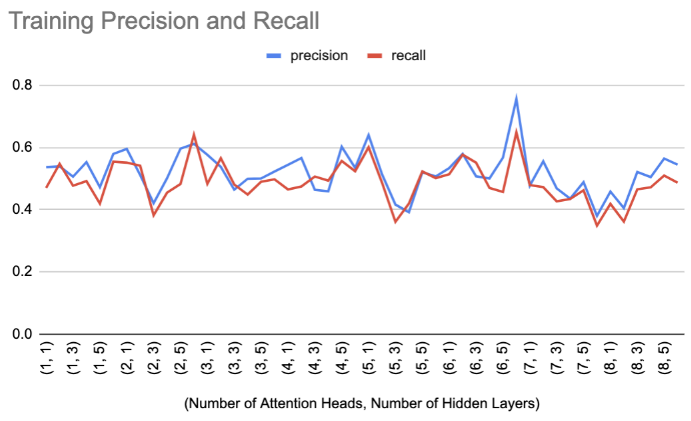


Figure 29: Fine-Tuning T5 Training Metrics (own creation)

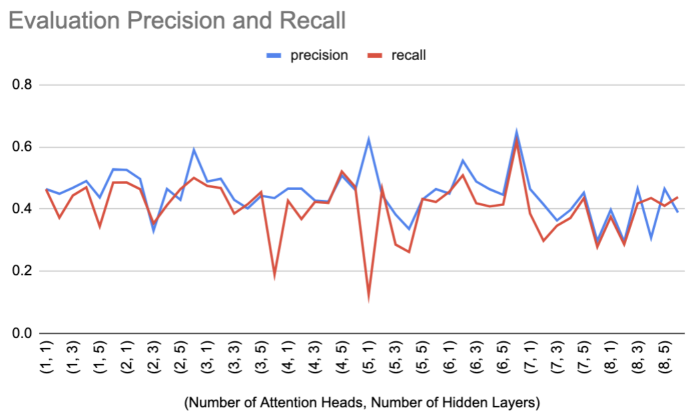


Figure 30: Fine-Tuning T5 Evaluation Metrics (own creation)

These graphs reinforce the distinction between the pre-trained T5 and the base T5. The metrics show great improvement and stabilization both from evaluation and training. We can clearly see that the training metrics have significantly increased, determining fewer false negatives (precision) and fewer false positives (recall) than in the previously seen trained model. There is also a significant increase and stabilization in the evaluation metrics. In Figure 29 we can see the erratic behavior that these metrics have. However, although not as good as the training metrics the fine-tuned T5 shows the evaluation metrics following the same patterns. In turn, the stabilization of these metrics means that the model is able to generalize patterns and obtain a good base from which to generate features. As discussed previously, this being an abstractive natural language task, it also means that the model understands the scope it is being used in and generates appropriately contextualized features.

10.2.6 T5 Result Discussion

The T5 is a model trained on text-to-text generation, it has shown great value in related tasks such as abstractive summarization and key phrase generation. Therefore, the introduction of this model to the study seems apparent. The T5, unsurprisingly, performed very well and generated very good metrics for both evaluation and training.

The T5 training versus fine-tuning results is a great highlight of the importance of fine-tuning for specific tasks. With the help of fine-tuning our model learns the base to which it corresponds (mobile applications domain), learns the task it has been requested to do (feature extraction) and generates proper, understandable features comparable to those that had been crowdsourced. Even so, the metrics obtained from the T5 model were far from ideal. In the state-of-the-art review, we saw plenty of models pre-trained on far more data.

The current trend in NLP suggests the more parameters a model is trained on the easier it will be for this model to pick up on the queues of our specific task. Therefore, in a less limited study, using larger text-to-text models can result in a great improvement in the training metrics.

10.2.7 BERT vs T5

Having explored both models in depth and compared their capabilities in a fine-tuning capacity and in a training capacity, we are now ready to compare these models side-by-side. The metric we will be used to evaluate the extracted features is ROUGE. We will be using three different ROUGE values that will allow an in-depth look at how these models perform:

- **ROUGE-1:** Overlap of unigrams between the model output and the expected output.
- **ROUGE-2:** Overlap of bigrams between the model output and the expected output.
- **ROUGE-L:** Longest Common Subsequences (LCS) "takes into account sentence level structure similarity naturally and identifies longest co-occurring in sequence n-grams automatically." [59]

These metrics are seen in the table below for BERT and T5. These models were chosen heuristically by the training and evaluation loss. Both of the selected models were fine-tuned.

		BERT	**T5**
ROUGE-1	Minimum	0.182	0.325
	Average	0.287	0.496
	Maximum	0.359	0.530
ROUGE-2	Minimum	0.0	0.210
	Average	0.129	0.354
	Maximum	0.241	0.412
ROUGE-L	Minimum	0.182	0.325
	Average	0.287	0.496
	Maximum	0.359	0.530

Table 22: Estimated Total Costs by Project Roles (own creation)

Having seen the loss metrics and the ROUGE metrics for the evaluation dataset, we can conclude that our T5 model outperforms the BERT model by a significant increase. As mentioned previously, this model still has a lot of room for improvement, but the results obtained are comparable to those in previous studies. We can further analyze these by comparing the individual metrics shown in the graph.

Although this initial evaluation gives us a computational view of how both of these models work, there are other real-world requirements that play a part in the current surge of machine-learning models. Therefore we will consider the following metrics as a measure of the real-world applicability of these models.

The ROUGE-1 metric contains the unigrams matching the model output and the expected output. Therefore, we are matching any word with any word. Features tend to contain similar words to express different aspects of the application. Therefore, expectedly so, the ROUGE-1 metric is high. In the BERT model the ROUGE-1 metric matches around 30% of the unigrams. This means around 30% of the words in the output is found in the expected output. Putting this into perspective, three out of ten words will be correctly matched. In the T5 the ROUGE-1 metric reaches This metric does not guarantee that the features are correctly identified, most features in our data have more than one token. This, therefore, is not a significant metric for our system.

The Rouge-2 metric shows bigrams, there is a big proportion of our feature distributions. Therefore, this metric might have a better perception. We can see a significant decrease in both the T5 and the BERT models, however, this metric shows a realistic overview of how many features are matched correctly to both models.

The ROUGE-L metric counts the longest sequence that is shared between both. This gives us an insight into the ordering of the extracted features. Whether the model keeps up the structure of the input or generates a different output sequence. This is not a crucial aspect of our system but is a good insight into the functioning of the model and the relation it generates between the words. Both the BERT model and the T5 model end up with the same ROUGE-L metric as ROUGE-1, this indicates the proper training of the model.

11 Conclusions

The current scope of NLP in the research field is bounded by Large Language Models (LLMs). LLMs have outperformed small models trained in specific tasks at performing these tasks. These models require a vast amount of computational capacity and memory storage. The training requirements for LLMs have made the creation of these systems a niche market for those companies with a great number of resources. In these terms, the research in the field of Natural Language Processing has been bounded by LLMs.

The research in this field has transformed into keeping an inside look into these models. Rather than attempting to create new smaller models that outperform LLMs, the field has focused on disseminating LLMs. The research is focused on finding security flaws in the models, attempting to find techniques to specify the models to different areas, and simplifying these to be more accessible among others.

This research has also been outsourced to many companies which require Natural Language solutions. This has made it more accessible for smaller companies to use these models in day-to-day solutions. The system made through the implementation of this paper extracts an easy-to-use pipeline to identify the best model for any specific Natural Language task. This simplifies the usage of these models and requires less knowledge of the training pipeline from part of the user.

11.1 Analysis of Completion of Objectives

The objectives for this study have been divided into theoretical objectives and practical objectives. These are divided into sub-objectives that granize the overhead objective and make easier points more accessible and explainable. Exploring the completion of these sub-objectives gives a more extensive look at the extent of accomplishment of the system.

11.1.1 Theoretical Sub-Objectives

Research state-of-the-art transformer models and architecture

This objective concluded in the subsection of *State of the Art Review* titles *Transformer Models Review*. This section reviews the training and expected uses of each model as well as the spatial capacity of these.

Research fine-tuning techniques for transformer models

This objective concluded in the subsection of *State of the Art Review* titles *Fine-Tuning Techniques Review*. This section includes a survey of the current techniques and the expectations and requirements of these.

Explore regularization techniques for fine-tuning deep learning models.

Regularization is used to make sure that the fine-tuning process does not overfit the small amount of data provided to the model. In order to do this we implemented Learning Rate Schedulers and Early Stopping callbacks, more information is found in the *Overview of the Machine Learning*

Pipeline Design section.

Compute Spatial and Temporal Complexities

The temporal complexities of the systems are analyzed in the fundamentals section, which explores the temporal complexities of RNNs and transformer models. The spatial complexity is calculated based on parameters with which the model has been pre-trained in.

Analyze Expected Loss and Accuracy of Models

Section 10.2 - Experiment Review and Results explores the model losses and gives a comparative view of the expected results. This has been somewhat generalized to the trends that training and evaluating give us and how this relates in the conceptual field.

11.1.2 Practical Sub-Objectives

Program studied transformer models and fine-tuning techniques

The chosen models have been programmed using the HuggingFace library as a base. The code can be found on GitHub, these models were programmed both for fine-tuning and training.

Generalize fine-tuning process to be applicable to different transformer models.

There are some fine-tuning techniques that we have been able to generalize. A great part of this generalization is due to the inheritance of the model architecture classes. Through this, our experiments are able to run simultaneously for both models. The base code that has been made for this paper is also able to be extended for other models seen in the *State of the Art Review* section. Although out of the scope of the currently implemented system, the other feasible models presented already have a base from which to train and can easily be transferable to the rest of the models.

Analyze the loss and accuracy of the fine-tuned model

After the creation of the pipeline, we ran experiments and analyzed the output of these in the *Experiment Review and Results* section. There have been slight variations of the metrics we will want to analyze, aside from loss and accuracy we want to measure ROUGE, precision, and recall.

Automatize ranking of models and fine-tuning techniques and draw conclusions on obtained results.

The implementation of automatic rankings of the model is implemented to automatically launch the best resulting option. The implementation of ranking is discussed in the *Overview of the Machine Learning Pipeline Design* section.

11.2 Analysis of Completion of Technical Competences

CCO1.1: To evaluate the computational complexity of a problem, know the algorithmic strategies which can solve it, and recommend, develop and implement the solution

which guarantees the best performance according to the established requirements. [In depth]

The nature of this study is to compile different techniques previously used to solve similar tasks to that proposed and generate a comprehensive solution for it. For this purpose, we evaluate the computational complexity of the state-of-the-art models and technologies being used within these models. These have been evaluated on a time and space complexity basis. The technique selection section speaks to the limitations of the user and the scope this study can cover. The model implementation and the specifics added to the training and testing are covered in the Overview of the Machine Learning Pipeline Development section. This section covers how we guarantee that training and testing take the least amount of computational power and generates a comprehensive solution to guarantee the best performance.

CCO1.3: To define, evaluate and select platforms to develop and produce hardware and software for developing computer applications and services of different complexities. [In depth]

One of the main objectives of the development of this study was to create a platform that contained a series of transformer models and fine-tuning techniques to select the best model for natural language processing tasks. To do this, we selected initial platforms for testing the model development, determined the scope and requirements of the system, and selected a proper platform for development, training, testing, and deployment. This is reflected in the *Planning* section and the *Modifications From Initial Planning* sections.

CCO2.1: To demonstrate knowledge about the fundamentals, paradigms, and the own techniques of intelligent systems, and analyze, design, and build computer systems, services, and applications that use these techniques in any applicable field. [Enough]

Intelligent systems are defined as "technologically advanced machines that perceive and respond to the world around them" [60]. A main characteristic of human interaction is text, in this sense, this study is attempting to understand the use of natural language and extract key information from this. The developed system designed for will study the use of transformer models for a specific context and task and deploy a ready-to-use model. The design of this system is detailed in the *Overview of the Machine Learning Pipeline Design.*

CCO2.2: Capacity to acquire, obtain, formalize and represent human knowledge in a computable way to solve problems through a computer system in any applicable field, in particular in the fields related to computation, perception, and operation in intelligent environments. [In depth]

The idea behind this system is to take human-made descriptions and crowdsourced features extracted from using an application and generate a model that can extract these features. This model will be ready to use and deployed in Google Cloud.

CCO2.4: To demonstrate knowledge and develop techniques about computational learning; to design and implement applications and systems that use them, includ-

ing those dedicated to the automatic extraction of information and knowledge from large data volumes. [Enough]

The models used in this system are trained in a large corpus of data. The large amount of data used for these systems makes them able to generalize to other tasks. In this paper, we analyze the use of models trained from large data volumes to fine-tune them using smaller, more specific, datasets. The output of this paper is a ready-to-use model that can generate key features from an application description.

CCO3.1: To implement critical code following criteria like execution time, efficiency, and security. [A little bit]

The code for training and testing has been developed with time and efficiency in mind, the details of how this has been attempted can be found in the *Overview of the Machine Learning Pipeline Design* section. In terms of security, we have abstracted a large part of the training and testing process to ensure that the model development and deployment do not have major data leakages. The data used for this system is publicly accessible and is therefore not subject to any important data leakages.

12 Future Work

This section covers the work that can still be done both developmentally and experimentally. These cover a vast amount of aspects somewhat explored during the paper but have increasing potential to be explored and added to the pipeline that this work culminates in.

The scope of this paper is limited to transformer models (T5 and BERT), using fine-tuning techniques to explore the usability of the system in domain-specific feature extraction.

Add models

The most amount of work that can be done towards generating a better pipeline is adding more models. The classes created for the models and experimentation can easily be generalized to different classifications and text-to-text training models. There are a few models proposed in the state-of-art section that propose new systems. Although out of scope for this project, they would be great additions to the pipeline. Models such as PEGASUS and GPT-3 have proven to be very useful for generalization, although they do carry a great necessity for computational power. If the user has this capacity these models will ensure great results, even better than those showcased in this paper.

Other models that could be taken into consideration are subsidiary models that stem from BERT. There are a few that have been mentioned previously such as RoBERTa or DestilBERT. These models could add an edge to the exploration of BERT and how this can be used for our specific task.

Add Transfer Learning

As mentioned previously, Transfer Learning gets the output from one model and uses this output through another model. This way the information extracted from one model can add to what the model understands and can achieve from the data. This includes adding layers previous to the model input and previous to the model output. This approach can achieve great contextualization of the data and make sure that the model understands the context it is focused in as well as the task at hand.

Add Monitoring Values

There are various values extracted for every model and for every model type. Although we output the number sometimes these can get overwhelming. Especially for non-technical users, these values will not have any worth. To make this task more visual and allow non-technical users (or users not specialized in machine learning) to understand these metrics better we can add visual monitoring values which are added while training and testing. This will show more visual graphs that will explain these numbers in a visual graphic. There are embedded systems within PyTorch that can allow for this. A commonly used monitoring tool is Tensorboard [61].

Exploration of different Learning Rate Schedulers

As mentioned previously, for simplicity and proper updating of the training parameters we have added a Learning Rate Scheduler. There are many other options for Learning Rate Schedulers that could be explored and experimented on. Different Learning Rate Schedulers might be more suitable for different problem types. There are two potential explorations within this: experimenting with different Learning Rate Schedulers or identifying types of problems within which each Learning Rate Scheduler would work best and breaking the experimentation by a problem.

In the first, within each task, we would run the experimentation with a set of different Learning Rate Schedulers and determine which one works best as a whole with the rest of the model architectures. This would increase the computation time exponentially. The other option would be identifying tasks, within our expected tasks, that would go well with the different learning rates. This would mean that we would identify one learning rate per expected type and build on this for the system. To do this, we would have to undergo an exploration of previous papers as well as hands-on experimentation to consolidate our knowledge before we can apply this to our system.

Heuristics for Model Training and Testing.

In the current implementation, we are running every model from the most basic to the most complex architecture selected by the user. In this way, we are running too many experiments where we could have already determined to stop any further experiments. As an example, when the model is very large it can easily overfit the training data. This means that we can stop increasing the model's size (the number of layers) and concentrate on other factors. Doing this will decrease the time required to run the experiments. Instead of running all experiments, in all models, with all possible architectures we will be selecting the best time to stop will be based on the output metrics of the models.

Compatibility with different deployment types.

Due to the simplicity of using Google Cloud for deployment, our current system only allows for this type of model deployment. A task that would allow for more users to use this system would be adding different deployment types. This could also affect the sustainability of our system, opting for more sustainable options when deploying and predicting.

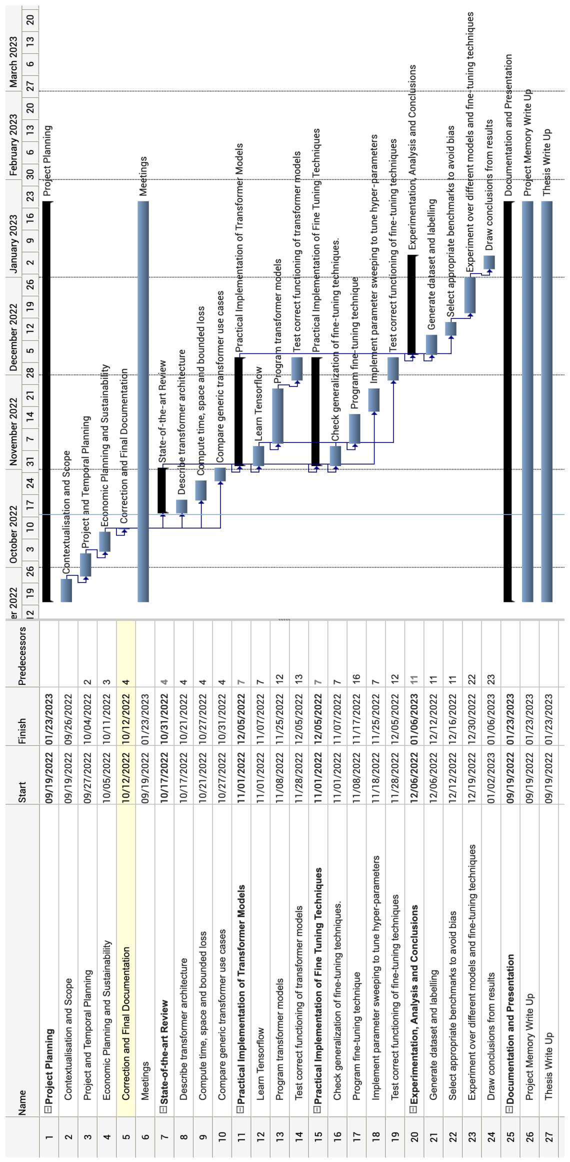
Exploration of Multi-modal and Computer Vision Applications

Another option is to make the pipeline extensible to not only Natural Language Processing tasks but also Multi-modal tasks and Computer Vision tasks. This can mean extending the model scope by adding models outside the transformer's scope. Therefore we would have to initially disseminate which task we would be working towards and create specific pipelines for these. Another option would be keeping the transformer constraint and adding tasks that focused on sequential data such as video and speech processing. Doing this would therefore require understanding what dataset we had and adding new data processing classes for each type.

Adjust Metrics for Text-to-Text Generation

The current implementation uses ROUGE as a metric for a text-to-text generation. As discussed, this metric was beneficial for our specific system, especially due to the ease of implementation and the metrics we could extract. The issue rises due to the difference in syntax format and word representations that we may find. For this reason, a different metric might be better suited. A few of these were presented previously, a good way to adjust these metrics would be to test a few and see how the model develops within them.

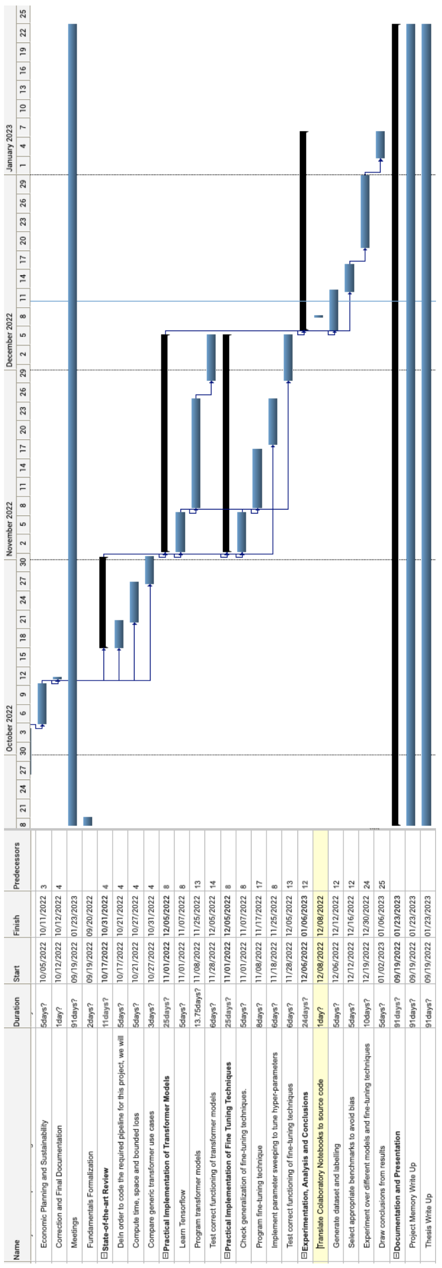
13 Annex A: Gnatt Chart



	Name	Start	Finish	Predecessors
1	**Project Planning**	**09/19/2022**	**01/23/2023**	
2	Contextualisation and Scope	09/19/2022	09/26/2022	
3	Project and Temporal Planning	09/27/2022	10/04/2022	2
4	Economic Planning and Sustainability	10/05/2022	10/11/2022	3
5	Correction and Final Documentation	10/12/2022	10/12/2022	4
6	Meetings	09/19/2022	01/23/2023	
7	**State-of-the-art Review**	**10/17/2022**	**10/31/2022**	4
8	Describe transformer architecture	10/17/2022	10/21/2022	4
9	Compute time, space and bounded loss	10/21/2022	10/27/2022	4
10	Compare generic transformer use cases	10/27/2022	10/31/2022	4
11	**Practical Implementation of Transformer Models**	**11/01/2022**	**12/05/2022**	7
12	Learn Tensorflow	11/01/2022	11/07/2022	7
13	Program transformer models	11/08/2022	11/25/2022	12
14	Test correct functioning of transformer models	11/28/2022	12/05/2022	13
15	**Practical Implementation of Fine Tuning Techniques**	**11/01/2022**	**12/05/2022**	7
16	Check generalization of fine-tuning techniques.	11/01/2022	11/07/2022	7
17	Program fine-tuning technique	11/08/2022	11/17/2022	16
18	Implement parameter sweeping to tune hyper-parameters	11/18/2022	11/25/2022	7
19	Test correct functioning of fine-tuning techniques	11/28/2022	12/05/2022	12
20	**Experimentation, Analysis and Conclusions**	**12/06/2022**	**01/06/2023**	11
21	Generate dataset and labelling	12/06/2022	12/12/2022	11
22	Select appropriate benchmarks to avoid bias	12/12/2022	12/16/2022	11
23	Experiment over different models and fine-tuning techniques	12/19/2022	12/30/2022	22
24	Draw conclusions from results	01/02/2023	01/06/2023	23
25	**Documentation and Presentation**	**09/19/2022**	**01/23/2023**	
26	Project Memory Write Up	09/19/2022	01/23/2023	
27	Thesis Write Up	09/19/2022	01/23/2023	

Figure 31: Planned Tasks - Gnatt Chart Large (own creation)

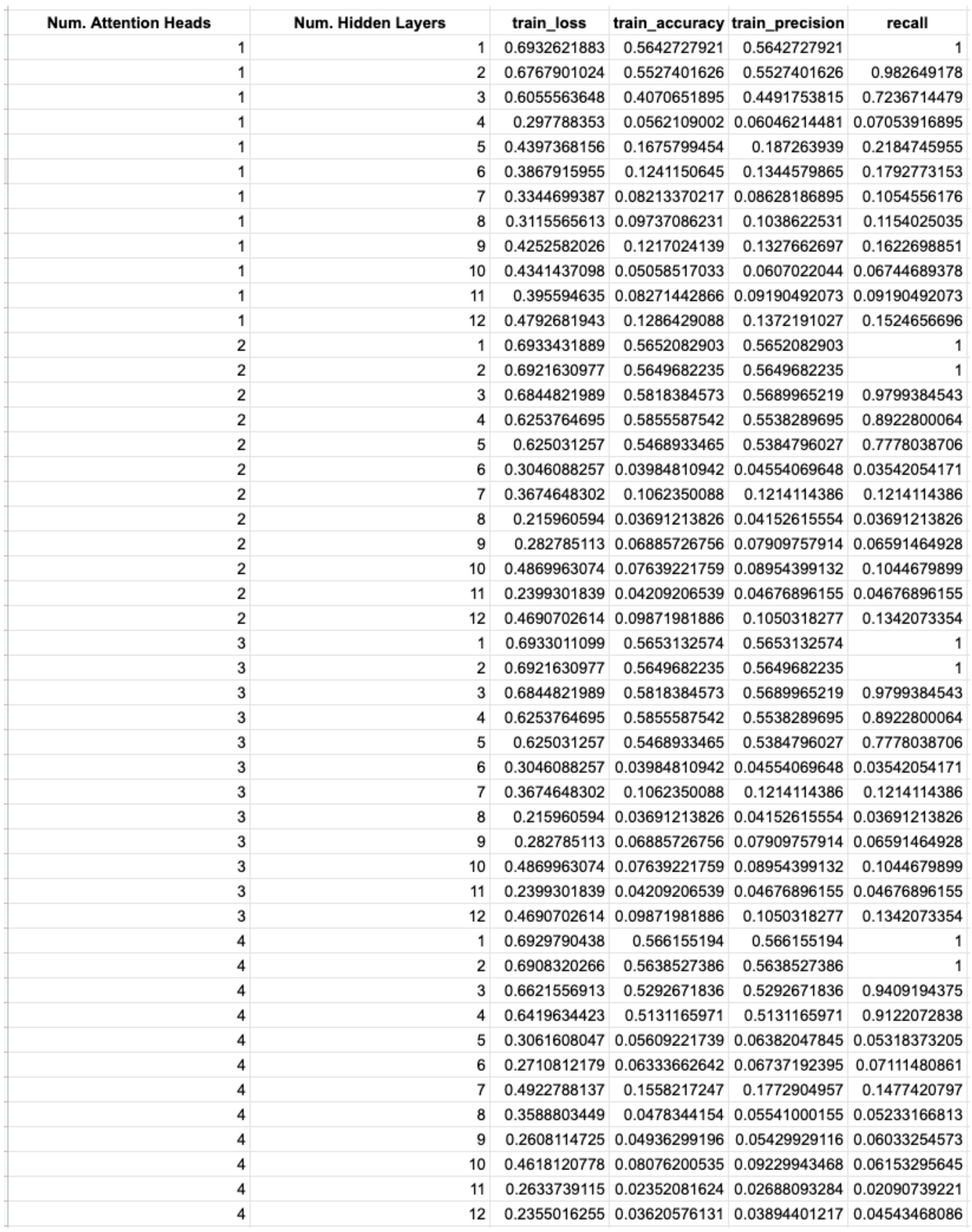
14 Annex B: Modified Gnatt Chart



Name	Duration	Start	Finish	Predecessors
Economic Planning and Sustainability	5days?	10/05/2022	10/11/2022	3
Correction and Final Documentation	1day?	10/12/2022	10/12/2022	4
Meetings	91days?	09/19/2022	01/23/2023	
Fundamentals Formalization	2days?	09/19/2022	09/20/2022	
⊟**State-of-the-art Review**	11days?	**10/17/2022**	**10/31/2022**	4
DeIn order to code the required pipeline for this project, we will	5days?	10/17/2022	10/21/2022	4
Compute time, space and bounded loss	5days?	10/21/2022	10/27/2022	4
Compare generic transformer use cases	3days?	10/27/2022	10/31/2022	4
⊟**Practical Implementation of Transformer Models**	25days?	**11/01/2022**	**12/05/2022**	8
Learn Tensorflow	5days?	11/01/2022	11/07/2022	8
Program transformer models	13.75days?	11/08/2022	11/25/2022	13
Test correct functioning of transformer models	6days?	11/28/2022	12/05/2022	14
⊟**Practical Implementation of Fine Tuning Techniques**	25days?	**11/01/2022**	**12/05/2022**	8
Check generalization of fine-tuning techniques.	5days?	11/01/2022	11/07/2022	8
Program fine-tuning technique	8days?	11/08/2022	11/17/2022	17
Implement parameter sweeping to tune hyper-parameters	6days?	11/18/2022	11/25/2022	8
Test correct functioning of fine-tuning techniques	6days?	11/28/2022	12/05/2022	13
⊟**Experimentation, Analysis and Conclusions**	24days?	**12/06/2022**	**01/06/2023**	12
Translate Colaboratory Notebooks to source code	1day?	12/08/2022	12/08/2022	
Generate dataset and labelling	5days?	12/06/2022	12/12/2022	12
Select appropriate benchmarks to avoid bias	5days?	12/12/2022	12/16/2022	12
Experiment over different models and fine-tuning techniques	10days?	12/19/2022	12/30/2022	24
Draw conclusions from results	5days?	01/02/2023	01/06/2023	25
⊟**Documentation and Presentation**	91days?	**09/19/2022**	**01/23/2023**	
Project Memory Write Up	91days?	09/19/2022	01/23/2023	
Thesis Write Up	91days?	09/19/2022	01/23/2023	

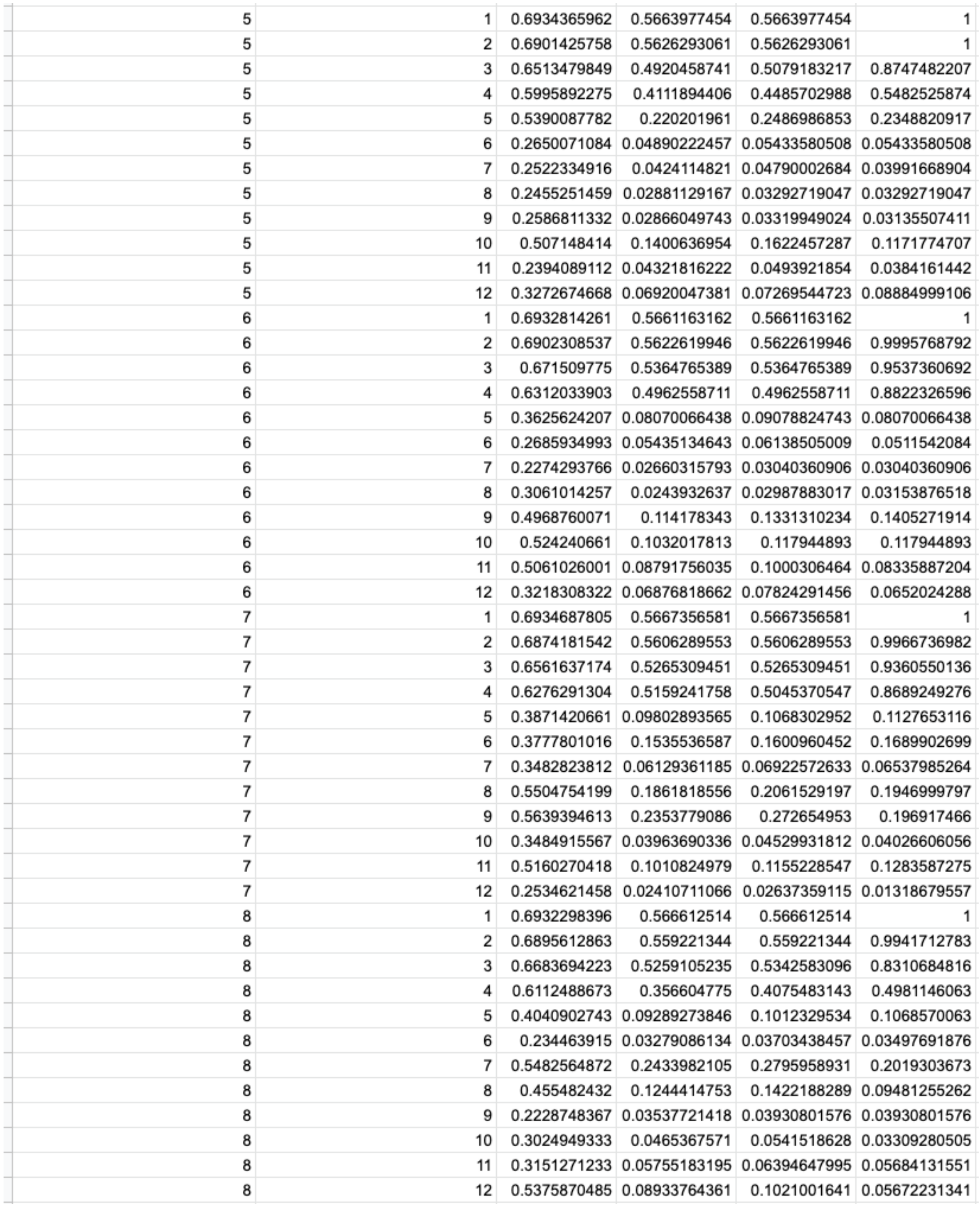
Figure 32: Planned Tasks - Gnatt Chart Large (own creation)

15 Annex C: Trained BERT data - Training Metrics



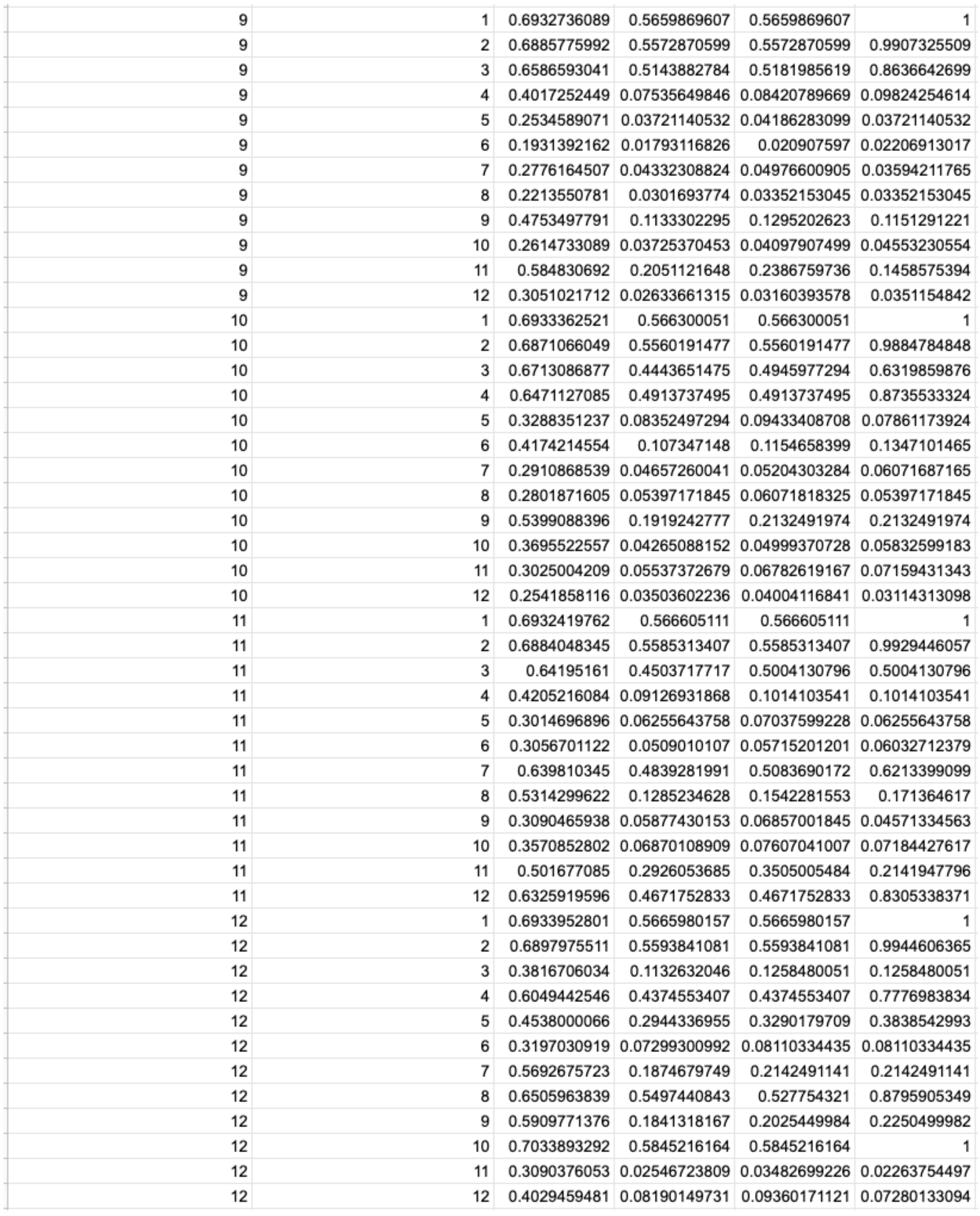
Num. Attention Heads	Num. Hidden Layers	train_loss	train_accuracy	train_precision	recall
1	1	0.6932621883	0.5642727921	0.5642727921	1
1	2	0.6767901024	0.5527401626	0.5527401626	0.982649178
1	3	0.6055563648	0.4070651895	0.4491753815	0.7236714479
1	4	0.297788353	0.0562109002	0.06046214481	0.07053916895
1	5	0.4397368156	0.1675799454	0.187263939	0.2184745955
1	6	0.3867915955	0.1241150645	0.1344579865	0.1792773153
1	7	0.3344699387	0.08213370217	0.08628186895	0.1054556176
1	8	0.3115565613	0.09737086231	0.1038622531	0.1154025035
1	9	0.4252582026	0.1217024139	0.1327662697	0.1622698851
1	10	0.4341437098	0.05058517033	0.0607022044	0.06744689378
1	11	0.395594635	0.08271442866	0.09190492073	0.09190492073
1	12	0.4792681943	0.1286429088	0.1372191027	0.1524656696
2	1	0.6933431889	0.5652082903	0.5652082903	1
2	2	0.6921630977	0.5649682235	0.5649682235	1
2	3	0.6844821989	0.5818384573	0.5689965219	0.9799384543
2	4	0.6253764695	0.5855587542	0.5538289695	0.8922800064
2	5	0.625031257	0.5468933465	0.5384796027	0.7778038706
2	6	0.3046088257	0.03984810942	0.04554069648	0.03542054171
2	7	0.3674648302	0.1062350088	0.1214114386	0.1214114386
2	8	0.215960594	0.03691213826	0.04152615554	0.03691213826
2	9	0.282785113	0.06885726756	0.07909757914	0.06591464928
2	10	0.4869963074	0.07639221759	0.08954399132	0.1044679899
2	11	0.2399301839	0.04209206539	0.04676896155	0.04676896155
2	12	0.4690702614	0.09871981886	0.1050318277	0.1342073354
3	1	0.6933011099	0.5653132574	0.5653132574	1
3	2	0.6921630977	0.5649682235	0.5649682235	1
3	3	0.6844821989	0.5818384573	0.5689965219	0.9799384543
3	4	0.6253764695	0.5855587542	0.5538289695	0.8922800064
3	5	0.625031257	0.5468933465	0.5384796027	0.7778038706
3	6	0.3046088257	0.03984810942	0.04554069648	0.03542054171
3	7	0.3674648302	0.1062350088	0.1214114386	0.1214114386
3	8	0.215960594	0.03691213826	0.04152615554	0.03691213826
3	9	0.282785113	0.06885726756	0.07909757914	0.06591464928
3	10	0.4869963074	0.07639221759	0.08954399132	0.1044679899
3	11	0.2399301839	0.04209206539	0.04676896155	0.04676896155
3	12	0.4690702614	0.09871981886	0.1050318277	0.1342073354
4	1	0.6929790438	0.566155194	0.566155194	1
4	2	0.6908320266	0.5638527386	0.5638527386	1
4	3	0.6621556913	0.5292671836	0.5292671836	0.9409194375
4	4	0.6419634423	0.5131165971	0.5131165971	0.9122072838
4	5	0.3061608047	0.05609221739	0.06382047845	0.05318373205
4	6	0.2710812179	0.06333662642	0.06737192395	0.07111480861
4	7	0.4922788137	0.1558217247	0.1772904957	0.1477420797
4	8	0.3588803449	0.0478344154	0.05541000155	0.05233166813
4	9	0.2608114725	0.04936299196	0.05429929116	0.06033254573
4	10	0.4618120778	0.08076200535	0.09229943468	0.06153295645
4	11	0.2633739115	0.02352081624	0.02688093284	0.02090739221
4	12	0.2355016255	0.03620576131	0.03894401217	0.04543468086

Figure 33: Trained BERT - Train Metrics (own creation)



5	1	0.6934365962	0.5663977454	0.5663977454	1
5	2	0.6901425758	0.5626293061	0.5626293061	1
5	3	0.6513479849	0.4920458741	0.5079183217	0.8747482207
5	4	0.5995892275	0.4111894406	0.4485702988	0.5482525874
5	5	0.5390087782	0.220201961	0.2486986853	0.2348820917
5	6	0.2650071084	0.04890222457	0.05433580508	0.05433580508
5	7	0.2522334916	0.0424114821	0.04790002684	0.03991668904
5	8	0.2455251459	0.02881129167	0.03292719047	0.03292719047
5	9	0.2586811332	0.02866049743	0.03319949024	0.03135507411
5	10	0.507148414	0.1400636954	0.1622457287	0.1171774707
5	11	0.2394089112	0.04321816222	0.0493921854	0.0384161442
5	12	0.3272674668	0.06920047381	0.07269544723	0.08884999106
6	1	0.6932814261	0.5661163162	0.5661163162	1
6	2	0.6902308537	0.5622619946	0.5622619946	0.9995768792
6	3	0.671509775	0.5364765389	0.5364765389	0.9537360692
6	4	0.6312033903	0.4962558711	0.4962558711	0.8822326596
6	5	0.3625624207	0.08070066438	0.09078824743	0.08070066438
6	6	0.2685934993	0.05435134643	0.06138505009	0.0511542084
6	7	0.2274293766	0.02660315793	0.03040360906	0.03040360906
6	8	0.3061014257	0.0243932637	0.02987883017	0.03153876518
6	9	0.4968760071	0.114178343	0.1331310234	0.1405271914
6	10	0.524240661	0.1032017813	0.117944893	0.117944893
6	11	0.5061026001	0.08791756035	0.1000306464	0.08335887204
6	12	0.3218308322	0.06876818662	0.07824291456	0.0652024288
7	1	0.6934687805	0.5667356581	0.5667356581	1
7	2	0.6874181542	0.5606289553	0.5606289553	0.9966736982
7	3	0.6561637174	0.5265309451	0.5265309451	0.9360550136
7	4	0.6276291304	0.5159241758	0.5045370547	0.8689249276
7	5	0.3871420661	0.09802893565	0.1068302952	0.1127653116
7	6	0.3777801016	0.1535536587	0.1600960452	0.1689902699
7	7	0.3482823812	0.06129361185	0.06922572633	0.06537985264
7	8	0.5504754199	0.1861818556	0.2061529197	0.1946999797
7	9	0.5639394613	0.2353779086	0.272654953	0.196917466
7	10	0.3484915567	0.03963690336	0.04529931812	0.04026606056
7	11	0.5160270418	0.1010824979	0.1155228547	0.1283587275
7	12	0.2534621458	0.02410711066	0.02637359115	0.01318679557
8	1	0.6932298396	0.566612514	0.566612514	1
8	2	0.6895612863	0.559221344	0.559221344	0.9941712783
8	3	0.6683694223	0.5259105235	0.5342583096	0.8310684816
8	4	0.6112488673	0.356604775	0.4075483143	0.4981146063
8	5	0.4040902743	0.09289273846	0.1012329534	0.1068570063
8	6	0.234463915	0.03279086134	0.03703438457	0.03497691876
8	7	0.5482564872	0.2433982105	0.2795958931	0.2019303673
8	8	0.455482432	0.1244414753	0.1422188289	0.09481255262
8	9	0.2228748367	0.03537721418	0.03930801576	0.03930801576
8	10	0.3024949333	0.0465367571	0.0541518628	0.03309280505
8	11	0.3151271233	0.05755183195	0.06394647995	0.05684131551
8	12	0.5375870485	0.08933764361	0.1021001641	0.05672231341

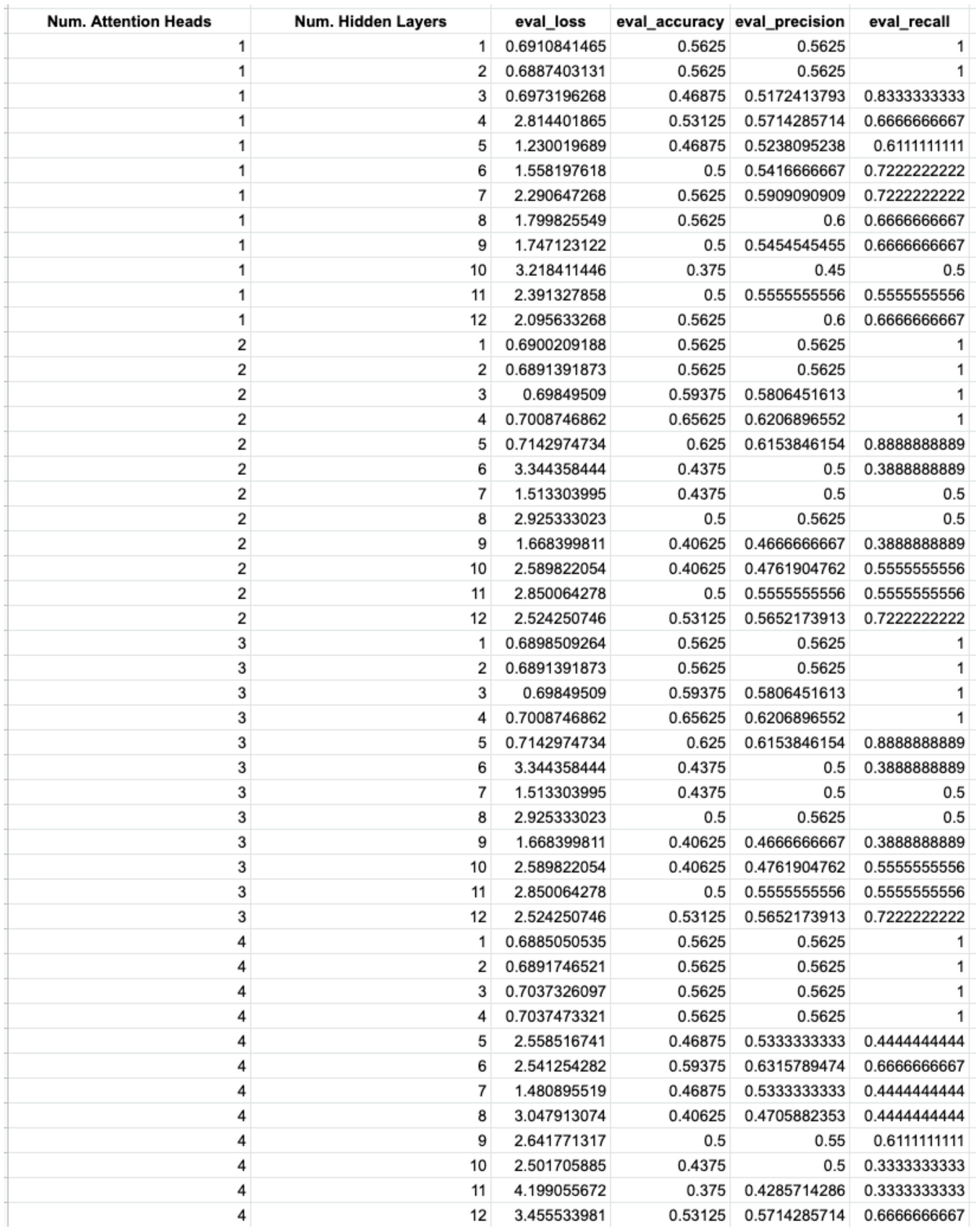
Figure 34: Trained BERT - Train Metrics (own creation)



9	1	0.6932736089	0.5659869607	0.5659869607	1
9	2	0.6885775992	0.5572870599	0.5572870599	0.9907325509
9	3	0.6586593041	0.5143882784	0.5181985619	0.8636642699
9	4	0.4017252449	0.07535649846	0.08420789669	0.09824254614
9	5	0.2534589071	0.03721140532	0.04186283099	0.03721140532
9	6	0.1931392162	0.01793116826	0.020907597	0.02206913017
9	7	0.2776164507	0.04332308824	0.04976600905	0.03594211765
9	8	0.2213550781	0.0301693774	0.03352153045	0.03352153045
9	9	0.4753497791	0.1133302295	0.1295202623	0.1151291221
9	10	0.2614733089	0.03725370453	0.04097907499	0.04553230554
9	11	0.584830692	0.2051121648	0.2386759736	0.1458575394
9	12	0.3051021712	0.02633661315	0.03160393578	0.0351154842
10	1	0.6933362521	0.566300051	0.566300051	1
10	2	0.6871066049	0.5560191477	0.5560191477	0.9884784848
10	3	0.6713086877	0.4443651475	0.4945977294	0.6319859876
10	4	0.6471127085	0.4913737495	0.4913737495	0.8735533324
10	5	0.3288351237	0.08352497294	0.09433408708	0.07861173924
10	6	0.4174214554	0.107347148	0.1154658399	0.1347101465
10	7	0.2910868539	0.04657260041	0.05204303284	0.06071687165
10	8	0.2801871605	0.05397171845	0.06071818325	0.05397171845
10	9	0.5399088396	0.1919242777	0.2132491974	0.2132491974
10	10	0.3695522557	0.04265088152	0.04999370728	0.05832599183
10	11	0.3025004209	0.05537372679	0.06782619167	0.07159431343
10	12	0.2541858116	0.03503602236	0.04004116841	0.03114313098
11	1	0.6932419762	0.566605111	0.566605111	1
11	2	0.6884048345	0.5585313407	0.5585313407	0.9929446057
11	3	0.64195161	0.4503717717	0.5004130796	0.5004130796
11	4	0.4205216084	0.09126931868	0.1014103541	0.1014103541
11	5	0.3014696896	0.06255643758	0.07037599228	0.06255643758
11	6	0.3056701122	0.0509010107	0.05715201201	0.06032712379
11	7	0.639810345	0.4839281991	0.5083690172	0.6213399099
11	8	0.5314299622	0.1285234628	0.1542281553	0.171364617
11	9	0.3090465938	0.05877430153	0.06857001845	0.04571334563
11	10	0.3570852802	0.06870108909	0.07607041007	0.07184427617
11	11	0.501677085	0.2926053685	0.3505005484	0.2141947796
11	12	0.6325919596	0.4671752833	0.4671752833	0.8305338371
12	1	0.6933952801	0.5665980157	0.5665980157	1
12	2	0.6897975511	0.5593841081	0.5593841081	0.9944606365
12	3	0.3816706034	0.1132632046	0.1258480051	0.1258480051
12	4	0.6049442546	0.4374553407	0.4374553407	0.7776983834
12	5	0.4538000066	0.2944336955	0.3290179709	0.3838542993
12	6	0.3197030919	0.07299300992	0.08110334435	0.08110334435
12	7	0.5692675723	0.1874679749	0.2142491141	0.2142491141
12	8	0.6505963839	0.5497440843	0.527754321	0.8795905349
12	9	0.5909771376	0.1841318167	0.2025449984	0.2250499982
12	10	0.7033893292	0.5845216164	0.5845216164	1
12	11	0.3090376053	0.02546723809	0.03482699226	0.02263754497
12	12	0.4029459481	0.08190149731	0.09360171121	0.07280133094

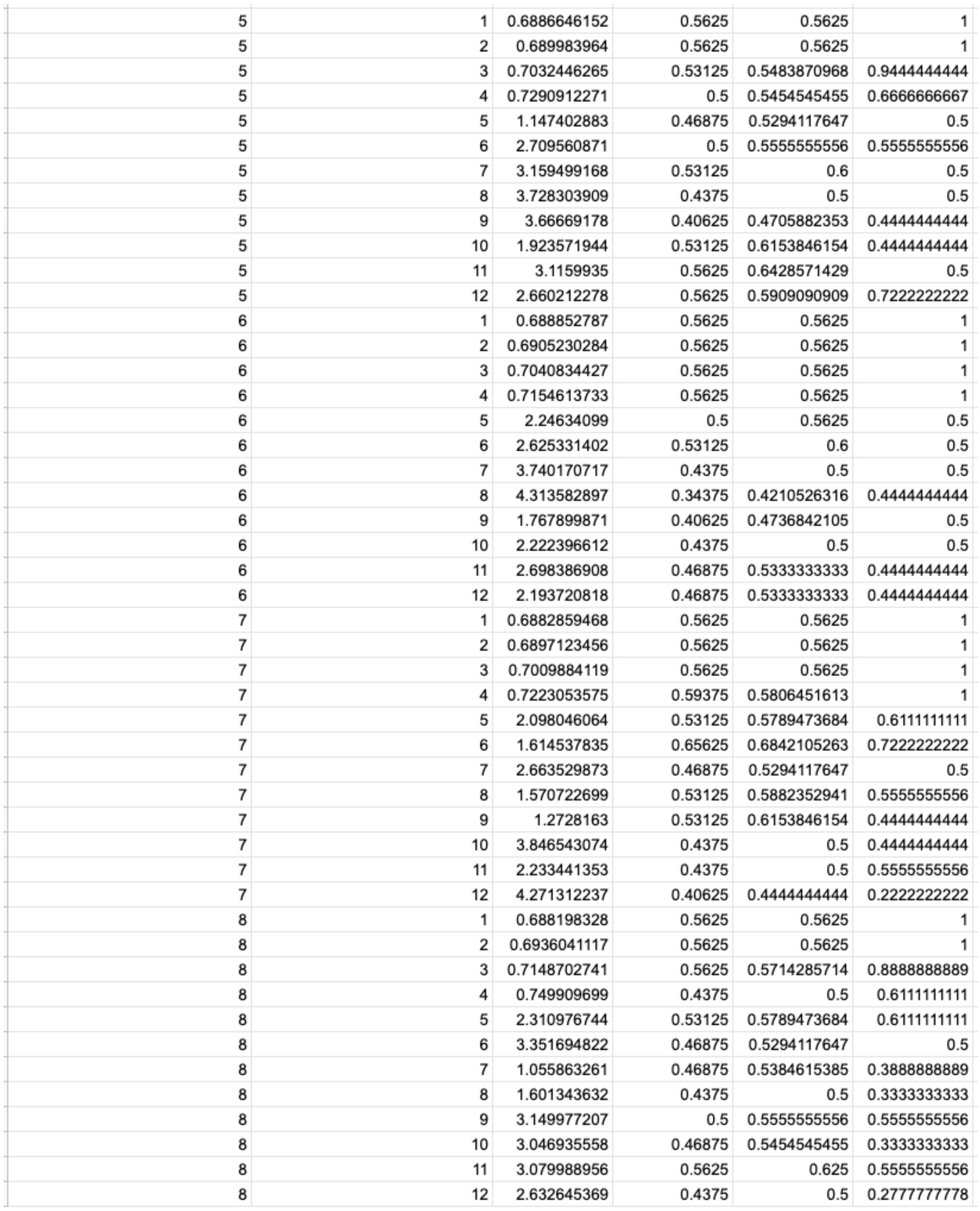
Figure 35: Trained BERT - Train Metrics (own creation)

16 Annex D: Trained BERT data - Evaluation Metrics



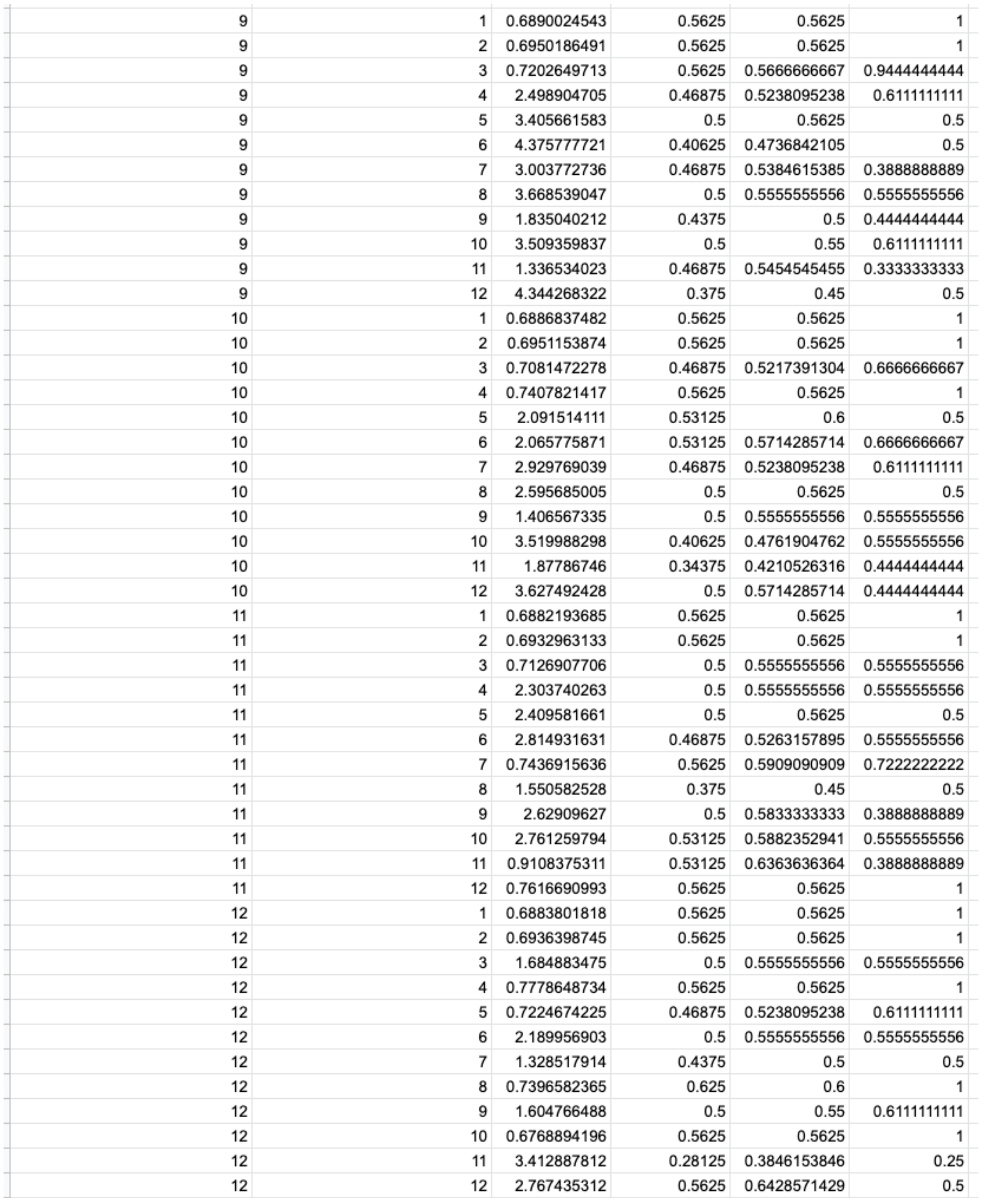
Num. Attention Heads	Num. Hidden Layers	eval_loss	eval_accuracy	eval_precision	eval_recall
1	1	0.6910841465	0.5625	0.5625	1
1	2	0.6887403131	0.5625	0.5625	1
1	3	0.6973196268	0.46875	0.5172413793	0.8333333333
1	4	2.814401865	0.53125	0.5714285714	0.6666666667
1	5	1.230019689	0.46875	0.5238095238	0.6111111111
1	6	1.558197618	0.5	0.5416666667	0.7222222222
1	7	2.290647268	0.5625	0.5909090909	0.7222222222
1	8	1.799825549	0.5625	0.6	0.6666666667
1	9	1.747123122	0.5	0.5454545455	0.6666666667
1	10	3.218411446	0.375	0.45	0.5
1	11	2.391327858	0.5	0.5555555556	0.5555555556
1	12	2.095633268	0.5625	0.6	0.6666666667
2	1	0.6900209188	0.5625	0.5625	1
2	2	0.6891391873	0.5625	0.5625	1
2	3	0.69849509	0.59375	0.5806451613	1
2	4	0.7008746862	0.65625	0.6206896552	1
2	5	0.7142974734	0.625	0.6153846154	0.8888888889
2	6	3.344358444	0.4375	0.5	0.3888888889
2	7	1.513303995	0.4375	0.5	0.5
2	8	2.925333023	0.5	0.5625	0.5
2	9	1.668399811	0.40625	0.4666666667	0.3888888889
2	10	2.589822054	0.40625	0.4761904762	0.5555555556
2	11	2.850064278	0.5	0.5555555556	0.5555555556
2	12	2.524250746	0.53125	0.5652173913	0.7222222222
3	1	0.6898509264	0.5625	0.5625	1
3	2	0.6891391873	0.5625	0.5625	1
3	3	0.69849509	0.59375	0.5806451613	1
3	4	0.7008746862	0.65625	0.6206896552	1
3	5	0.7142974734	0.625	0.6153846154	0.8888888889
3	6	3.344358444	0.4375	0.5	0.3888888889
3	7	1.513303995	0.4375	0.5	0.5
3	8	2.925333023	0.5	0.5625	0.5
3	9	1.668399811	0.40625	0.4666666667	0.3888888889
3	10	2.589822054	0.40625	0.4761904762	0.5555555556
3	11	2.850064278	0.5	0.5555555556	0.5555555556
3	12	2.524250746	0.53125	0.5652173913	0.7222222222
4	1	0.6885050535	0.5625	0.5625	1
4	2	0.6891746521	0.5625	0.5625	1
4	3	0.7037326097	0.5625	0.5625	1
4	4	0.7037473321	0.5625	0.5625	1
4	5	2.558516741	0.46875	0.5333333333	0.4444444444
4	6	2.541254282	0.59375	0.6315789474	0.6666666667
4	7	1.480895519	0.46875	0.5333333333	0.4444444444
4	8	3.047913074	0.40625	0.4705882353	0.4444444444
4	9	2.641771317	0.5	0.55	0.6111111111
4	10	2.501705885	0.4375	0.5	0.3333333333
4	11	4.199055672	0.375	0.4285714286	0.3333333333
4	12	3.455533981	0.53125	0.5714285714	0.6666666667

Figure 36: Trained BERT - Evaluation Metrics (own creation)



5	1	0.6886646152	0.5625	0.5625	1
5	2	0.689983964	0.5625	0.5625	1
5	3	0.7032446265	0.53125	0.5483870968	0.9444444444
5	4	0.7290912271	0.5	0.5454545455	0.6666666667
5	5	1.147402883	0.46875	0.5294117647	0.5
5	6	2.709560871	0.5	0.5555555556	0.5555555556
5	7	3.159499168	0.53125	0.6	0.5
5	8	3.728303909	0.4375	0.5	0.5
5	9	3.66669178	0.40625	0.4705882353	0.4444444444
5	10	1.923571944	0.53125	0.6153846154	0.4444444444
5	11	3.1159935	0.5625	0.6428571429	0.5
5	12	2.660212278	0.5625	0.5909090909	0.7222222222
6	1	0.688852787	0.5625	0.5625	1
6	2	0.6905230284	0.5625	0.5625	1
6	3	0.7040834427	0.5625	0.5625	1
6	4	0.7154613733	0.5625	0.5625	1
6	5	2.24634099	0.5	0.5625	0.5
6	6	2.625331402	0.53125	0.6	0.5
6	7	3.740170717	0.4375	0.5	0.5
6	8	4.313582897	0.34375	0.4210526316	0.4444444444
6	9	1.767899871	0.40625	0.4736842105	0.5
6	10	2.222396612	0.4375	0.5	0.5
6	11	2.698386908	0.46875	0.5333333333	0.4444444444
6	12	2.193720818	0.46875	0.5333333333	0.4444444444
7	1	0.6882859468	0.5625	0.5625	1
7	2	0.6897123456	0.5625	0.5625	1
7	3	0.7009884119	0.5625	0.5625	1
7	4	0.7223053575	0.59375	0.5806451613	1
7	5	2.098046064	0.53125	0.5789473684	0.6111111111
7	6	1.614537835	0.65625	0.6842105263	0.7222222222
7	7	2.663529873	0.46875	0.5294117647	0.5
7	8	1.570722699	0.53125	0.5882352941	0.5555555556
7	9	1.2728163	0.53125	0.6153846154	0.4444444444
7	10	3.846543074	0.4375	0.5	0.4444444444
7	11	2.233441353	0.4375	0.5	0.5555555556
7	12	4.271312237	0.40625	0.4444444444	0.2222222222
8	1	0.688198328	0.5625	0.5625	1
8	2	0.6936041117	0.5625	0.5625	1
8	3	0.7148702741	0.5625	0.5714285714	0.8888888889
8	4	0.749909699	0.4375	0.5	0.6111111111
8	5	2.310976744	0.53125	0.5789473684	0.6111111111
8	6	3.351694822	0.46875	0.5294117647	0.5
8	7	1.055863261	0.46875	0.5384615385	0.3888888889
8	8	1.601343632	0.4375	0.5	0.3333333333
8	9	3.149977207	0.5	0.5555555556	0.5555555556
8	10	3.046935558	0.46875	0.5454545455	0.3333333333
8	11	3.079988956	0.5625	0.625	0.5555555556
8	12	2.632645369	0.4375	0.5	0.2777777778

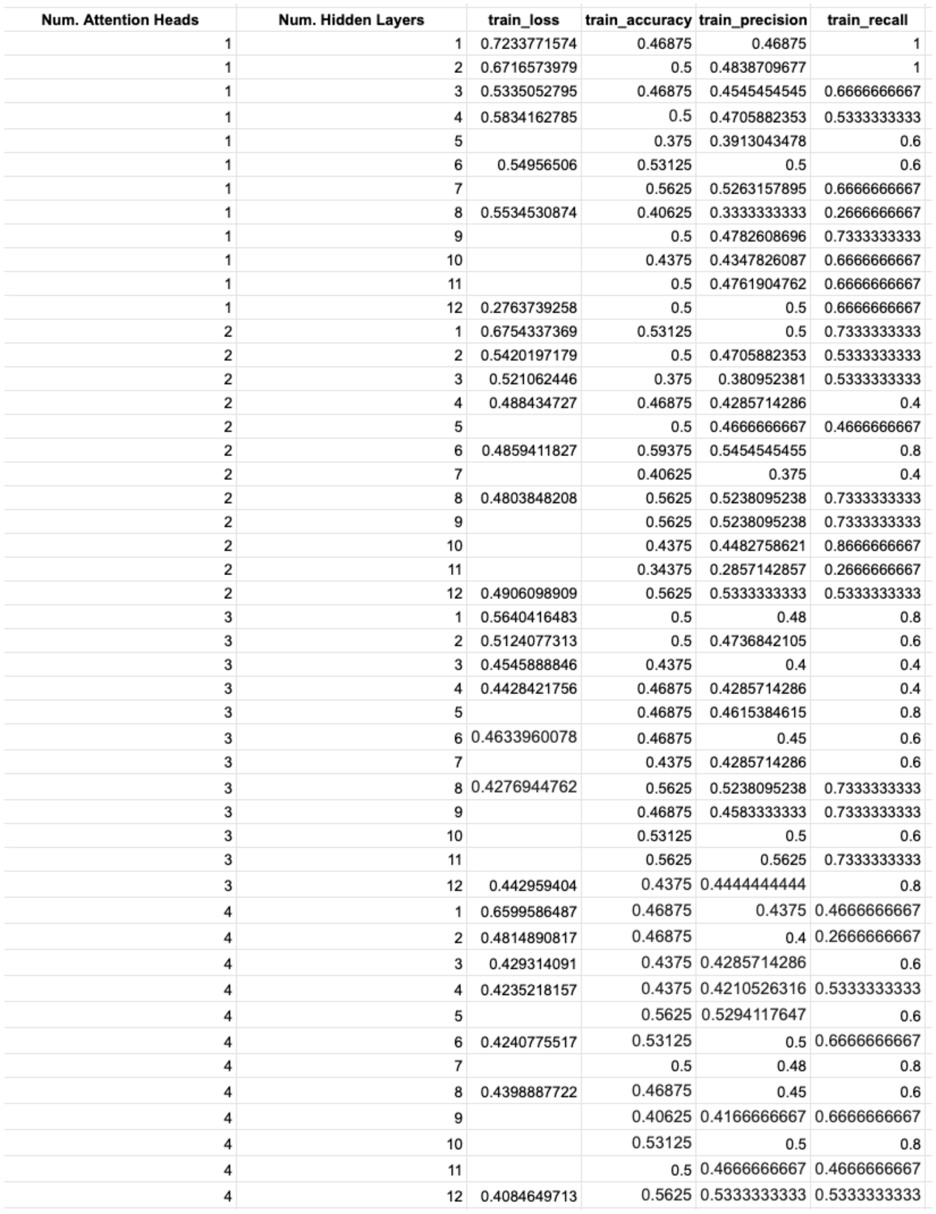
Figure 37: Trained BERT - Evaluation Metrics (own creation)



9	1	0.6890024543	0.5625	0.5625	1
9	2	0.6950186491	0.5625	0.5625	1
9	3	0.7202649713	0.5625	0.5666666667	0.9444444444
9	4	2.498904705	0.46875	0.5238095238	0.6111111111
9	5	3.405661583	0.5	0.5625	0.5
9	6	4.375777721	0.40625	0.4736842105	0.5
9	7	3.003772736	0.46875	0.5384615385	0.3888888889
9	8	3.668539047	0.5	0.5555555556	0.5555555556
9	9	1.835040212	0.4375	0.5	0.4444444444
9	10	3.509359837	0.5	0.55	0.6111111111
9	11	1.336534023	0.46875	0.5454545455	0.3333333333
9	12	4.344268322	0.375	0.45	0.5
10	1	0.6886837482	0.5625	0.5625	1
10	2	0.6951153874	0.5625	0.5625	1
10	3	0.7081472278	0.46875	0.5217391304	0.6666666667
10	4	0.7407821417	0.5625	0.5625	1
10	5	2.091514111	0.53125	0.6	0.5
10	6	2.065775871	0.53125	0.5714285714	0.6666666667
10	7	2.929769039	0.46875	0.5238095238	0.6111111111
10	8	2.595685005	0.5	0.5625	0.5
10	9	1.406567335	0.5	0.5555555556	0.5555555556
10	10	3.519988298	0.40625	0.4761904762	0.5555555556
10	11	1.87786746	0.34375	0.4210526316	0.4444444444
10	12	3.627492428	0.5	0.5714285714	0.4444444444
11	1	0.6882193685	0.5625	0.5625	1
11	2	0.6932963133	0.5625	0.5625	1
11	3	0.7126907706	0.5	0.5555555556	0.5555555556
11	4	2.303740263	0.5	0.5555555556	0.5555555556
11	5	2.409581661	0.5	0.5625	0.5
11	6	2.814931631	0.46875	0.5263157895	0.5555555556
11	7	0.7436915636	0.5625	0.5909090909	0.7222222222
11	8	1.550582528	0.375	0.45	0.5
11	9	2.62909627	0.5	0.5833333333	0.3888888889
11	10	2.761259794	0.53125	0.5882352941	0.5555555556
11	11	0.9108375311	0.53125	0.6363636364	0.3888888889
11	12	0.7616690993	0.5625	0.5625	1
12	1	0.6883801818	0.5625	0.5625	1
12	2	0.6936398745	0.5625	0.5625	1
12	3	1.684883475	0.5	0.5555555556	0.5555555556
12	4	0.7778648734	0.5625	0.5625	1
12	5	0.7224674225	0.46875	0.5238095238	0.6111111111
12	6	2.189956903	0.5	0.5555555556	0.5555555556
12	7	1.328517914	0.4375	0.5	0.5
12	8	0.7396582365	0.625	0.6	1
12	9	1.604766488	0.5	0.55	0.6111111111
12	10	0.6768894196	0.5625	0.5625	1
12	11	3.412887812	0.28125	0.3846153846	0.25
12	12	2.767435312	0.5625	0.6428571429	0.5

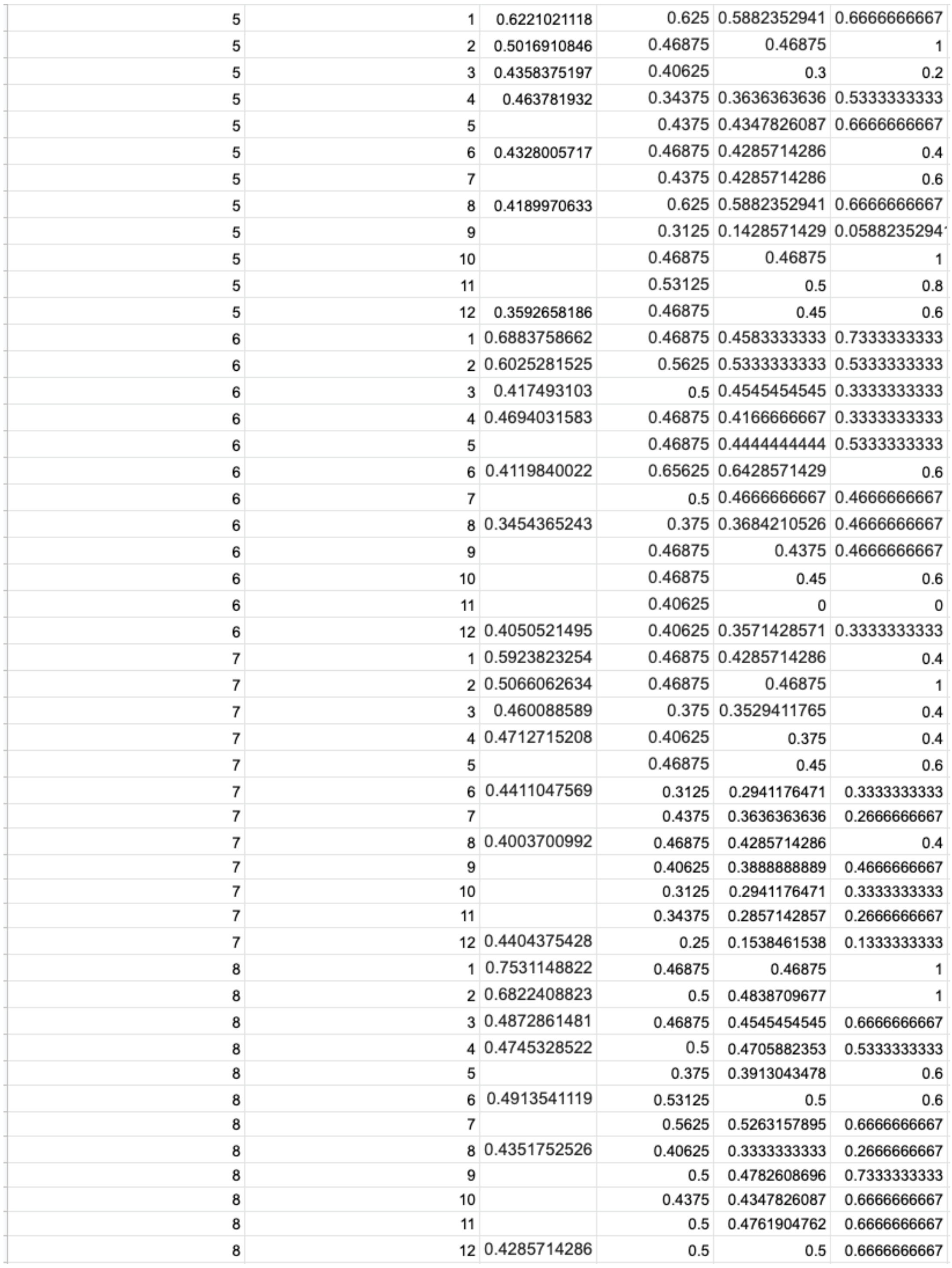
Figure 38: Trained BERT - Evaluation Metrics (own creation)

17 Annex E: Fine-Tuned BERT data - Evaluation Metrics



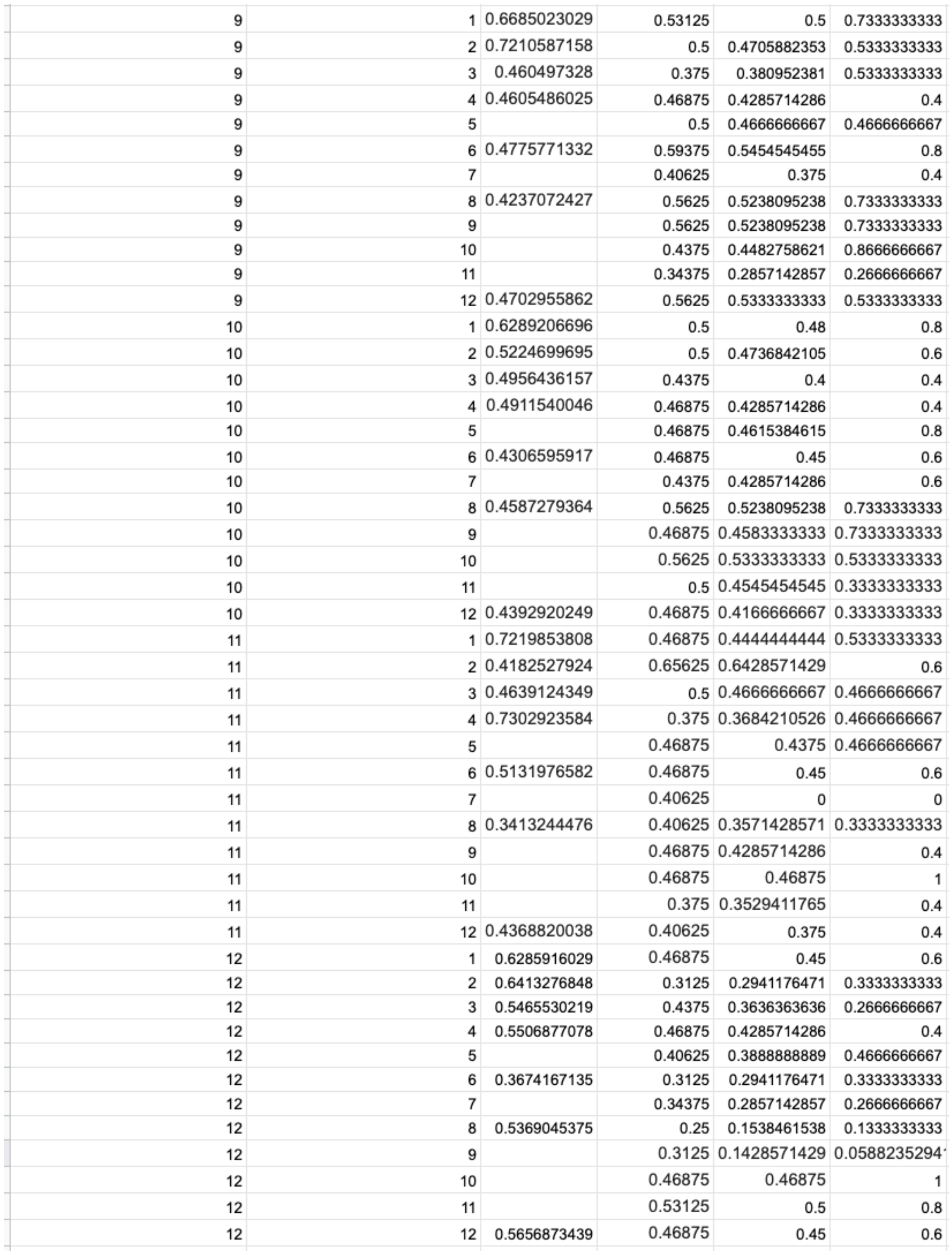
Num. Attention Heads	Num. Hidden Layers	train_loss	train_accuracy	train_precision	train_recall
1	1	0.7233771574	0.46875	0.46875	1
1	2	0.6716573979	0.5	0.4838709677	1
1	3	0.5335052795	0.46875	0.4545454545	0.6666666667
1	4	0.5834162785	0.5	0.4705882353	0.5333333333
1	5		0.375	0.3913043478	0.6
1	6	0.54956506	0.53125	0.5	0.6
1	7		0.5625	0.5263157895	0.6666666667
1	8	0.5534530874	0.40625	0.3333333333	0.2666666667
1	9		0.5	0.4782608696	0.7333333333
1	10		0.4375	0.4347826087	0.6666666667
1	11		0.5	0.4761904762	0.6666666667
1	12	0.2763739258	0.5	0.5	0.6666666667
2	1	0.6754337369	0.53125	0.5	0.7333333333
2	2	0.5420197179	0.5	0.4705882353	0.5333333333
2	3	0.521062446	0.375	0.380952381	0.5333333333
2	4	0.488434727	0.46875	0.4285714286	0.4
2	5		0.5	0.4666666667	0.4666666667
2	6	0.4859411827	0.59375	0.5454545455	0.8
2	7		0.40625	0.375	0.4
2	8	0.4803848208	0.5625	0.5238095238	0.7333333333
2	9		0.5625	0.5238095238	0.7333333333
2	10		0.4375	0.4482758621	0.8666666667
2	11		0.34375	0.2857142857	0.2666666667
2	12	0.4906098909	0.5625	0.5333333333	0.5333333333
3	1	0.5640416483	0.5	0.48	0.8
3	2	0.5124077313	0.5	0.4736842105	0.6
3	3	0.4545888846	0.4375	0.4	0.4
3	4	0.4428421756	0.46875	0.4285714286	0.4
3	5		0.46875	0.4615384615	0.8
3	6	0.4633960078	0.46875	0.45	0.6
3	7		0.4375	0.4285714286	0.6
3	8	0.4276944762	0.5625	0.5238095238	0.7333333333
3	9		0.46875	0.4583333333	0.7333333333
3	10		0.53125	0.5	0.6
3	11		0.5625	0.5625	0.7333333333
3	12	0.442959404	0.4375	0.4444444444	0.8
4	1	0.6599586487	0.46875	0.4375	0.4666666667
4	2	0.4814890817	0.46875	0.4	0.2666666667
4	3	0.429314091	0.4375	0.4285714286	0.6
4	4	0.4235218157	0.4375	0.4210526316	0.5333333333
4	5		0.5625	0.5294117647	0.6
4	6	0.4240775517	0.53125	0.5	0.6666666667
4	7		0.5	0.48	0.8
4	8	0.4398887722	0.46875	0.45	0.6
4	9		0.40625	0.4166666667	0.6666666667
4	10		0.53125	0.5	0.8
4	11		0.5	0.4666666667	0.4666666667
4	12	0.4084649713	0.5625	0.5333333333	0.5333333333

Figure 39: Fine-Tuned BERT - Train Metrics (own creation)



5	1	0.6221021118	0.625	0.5882352941	0.6666666667
5	2	0.5016910846	0.46875	0.46875	1
5	3	0.4358375197	0.40625	0.3	0.2
5	4	0.463781932	0.34375	0.3636363636	0.5333333333
5	5		0.4375	0.4347826087	0.6666666667
5	6	0.4328005717	0.46875	0.4285714286	0.4
5	7		0.4375	0.4285714286	0.6
5	8	0.4189970633	0.625	0.5882352941	0.6666666667
5	9		0.3125	0.1428571429	0.0588235294
5	10		0.46875	0.46875	1
5	11		0.53125	0.5	0.8
5	12	0.3592658186	0.46875	0.45	0.6
6	1	0.6883758662	0.46875	0.4583333333	0.7333333333
6	2	0.6025281525	0.5625	0.5333333333	0.5333333333
6	3	0.417493103	0.5	0.4545454545	0.3333333333
6	4	0.4694031583	0.46875	0.4166666667	0.3333333333
6	5		0.46875	0.4444444444	0.5333333333
6	6	0.4119840022	0.65625	0.6428571429	0.6
6	7		0.5	0.4666666667	0.4666666667
6	8	0.3454365243	0.375	0.3684210526	0.4666666667
6	9		0.46875	0.4375	0.4666666667
6	10		0.46875	0.45	0.6
6	11		0.40625	0	0
6	12	0.4050521495	0.40625	0.3571428571	0.3333333333
7	1	0.5923823254	0.46875	0.4285714286	0.4
7	2	0.5066062634	0.46875	0.46875	1
7	3	0.460088589	0.375	0.3529411765	0.4
7	4	0.4712715208	0.40625	0.375	0.4
7	5		0.46875	0.45	0.6
7	6	0.4411047569	0.3125	0.2941176471	0.3333333333
7	7		0.4375	0.3636363636	0.2666666667
7	8	0.4003700992	0.46875	0.4285714286	0.4
7	9		0.40625	0.3888888889	0.4666666667
7	10		0.3125	0.2941176471	0.3333333333
7	11		0.34375	0.2857142857	0.2666666667
7	12	0.4404375428	0.25	0.1538461538	0.1333333333
8	1	0.7531148822	0.46875	0.46875	1
8	2	0.6822408823	0.5	0.4838709677	1
8	3	0.4872861481	0.46875	0.4545454545	0.6666666667
8	4	0.4745328522	0.5	0.4705882353	0.5333333333
8	5		0.375	0.3913043478	0.6
8	6	0.4913541119	0.53125	0.5	0.6
8	7		0.5625	0.5263157895	0.6666666667
8	8	0.4351752526	0.40625	0.3333333333	0.2666666667
8	9		0.5	0.4782608696	0.7333333333
8	10		0.4375	0.4347826087	0.6666666667
8	11		0.5	0.4761904762	0.6666666667
8	12	0.4285714286	0.5	0.5	0.6666666667

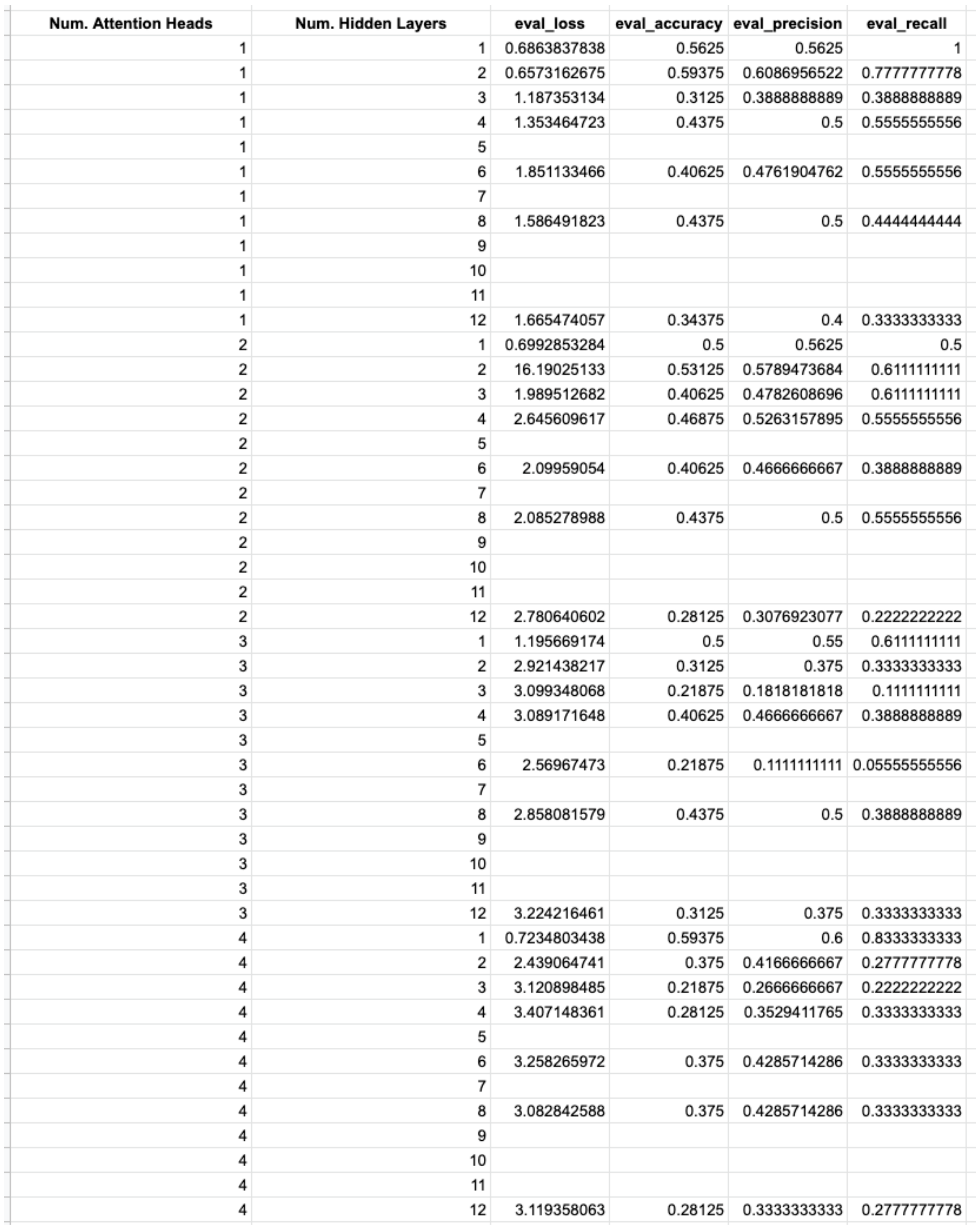
Figure 40: Fine-Tuned BERT - Train Metrics (own creation)



9	1	0.6685023029	0.53125	0.5	0.7333333333
9	2	0.7210587158	0.5	0.4705882353	0.5333333333
9	3	0.460497328	0.375	0.380952381	0.5333333333
9	4	0.4605486025	0.46875	0.4285714286	0.4
9	5		0.5	0.4666666667	0.4666666667
9	6	0.4775771332	0.59375	0.5454545455	0.8
9	7		0.40625	0.375	0.4
9	8	0.4237072427	0.5625	0.5238095238	0.7333333333
9	9		0.5625	0.5238095238	0.7333333333
9	10		0.4375	0.4482758621	0.8666666667
9	11		0.34375	0.2857142857	0.2666666667
9	12	0.4702955862	0.5625	0.5333333333	0.5333333333
10	1	0.6289206696	0.5	0.48	0.8
10	2	0.5224699695	0.5	0.4736842105	0.6
10	3	0.4956436157	0.4375	0.4	0.4
10	4	0.4911540046	0.46875	0.4285714286	0.4
10	5		0.46875	0.4615384615	0.8
10	6	0.4306595917	0.46875	0.45	0.6
10	7		0.4375	0.4285714286	0.6
10	8	0.4587279364	0.5625	0.5238095238	0.7333333333
10	9		0.46875	0.4583333333	0.7333333333
10	10		0.5625	0.5333333333	0.5333333333
10	11		0.5	0.4545454545	0.3333333333
10	12	0.4392920249	0.46875	0.4166666667	0.3333333333
11	1	0.7219853808	0.46875	0.4444444444	0.5333333333
11	2	0.4182527924	0.65625	0.6428571429	0.6
11	3	0.4639124349	0.5	0.4666666667	0.4666666667
11	4	0.7302923584	0.375	0.3684210526	0.4666666667
11	5		0.46875	0.4375	0.4666666667
11	6	0.5131976582	0.46875	0.45	0.6
11	7		0.40625	0	0
11	8	0.3413244476	0.40625	0.3571428571	0.3333333333
11	9		0.46875	0.4285714286	0.4
11	10		0.46875	0.46875	1
11	11		0.375	0.3529411765	0.4
11	12	0.4368820038	0.40625	0.375	0.4
12	1	0.6285916029	0.46875	0.45	0.6
12	2	0.6413276848	0.3125	0.2941176471	0.3333333333
12	3	0.5465530219	0.4375	0.3636363636	0.2666666667
12	4	0.5506877078	0.46875	0.4285714286	0.4
12	5		0.40625	0.3888888889	0.4666666667
12	6	0.3674167135	0.3125	0.2941176471	0.3333333333
12	7		0.34375	0.2857142857	0.2666666667
12	8	0.5369045375	0.25	0.1538461538	0.1333333333
12	9		0.3125	0.1428571429	0.0588235294
12	10		0.46875	0.46875	1
12	11		0.53125	0.5	0.8
12	12	0.5656873439	0.46875	0.45	0.6

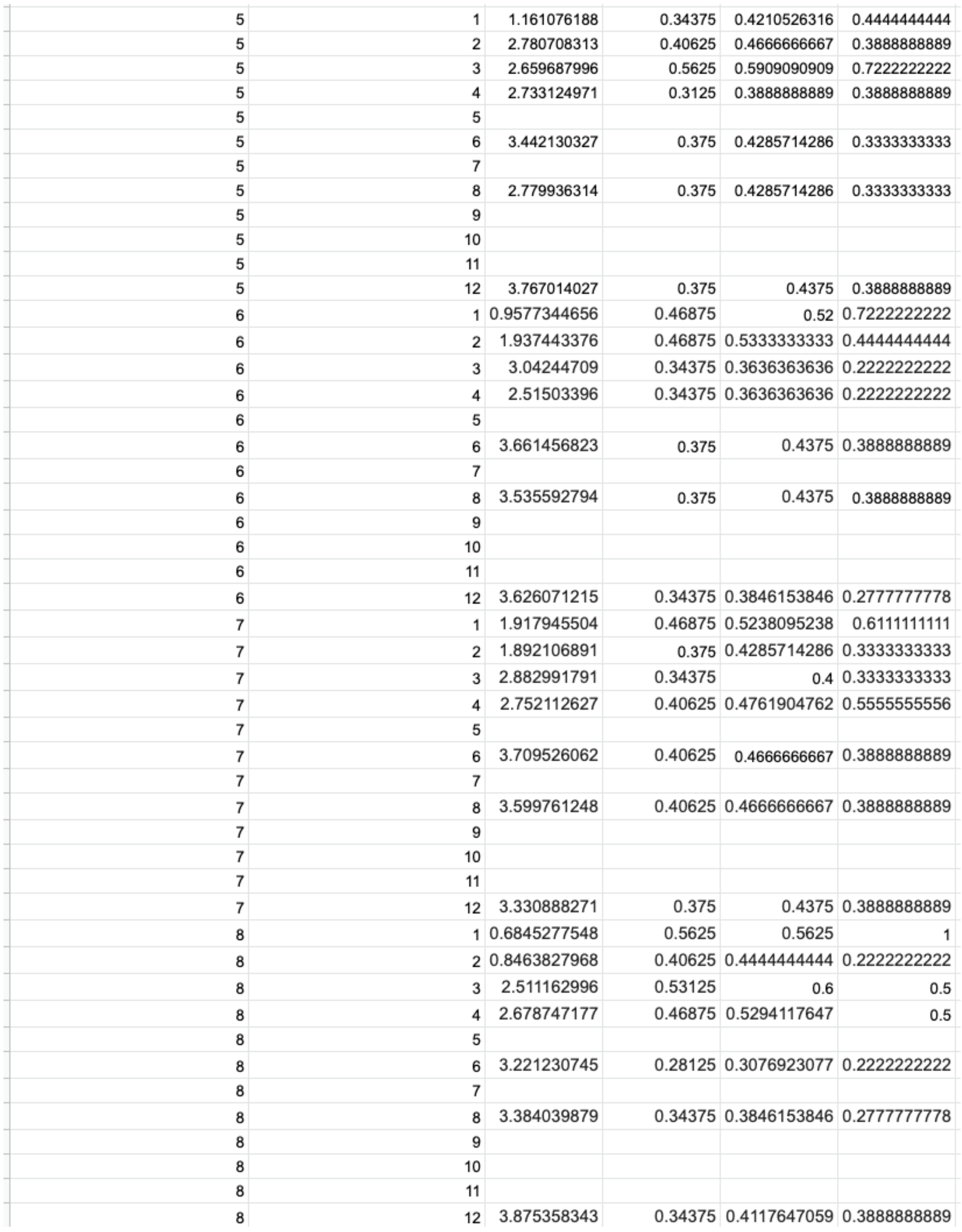
Figure 41: Fine-Tuned BERT - Train Metrics (own creation)

18 Annex F: Fine-Tuned BERT data - Evaluation Metrics



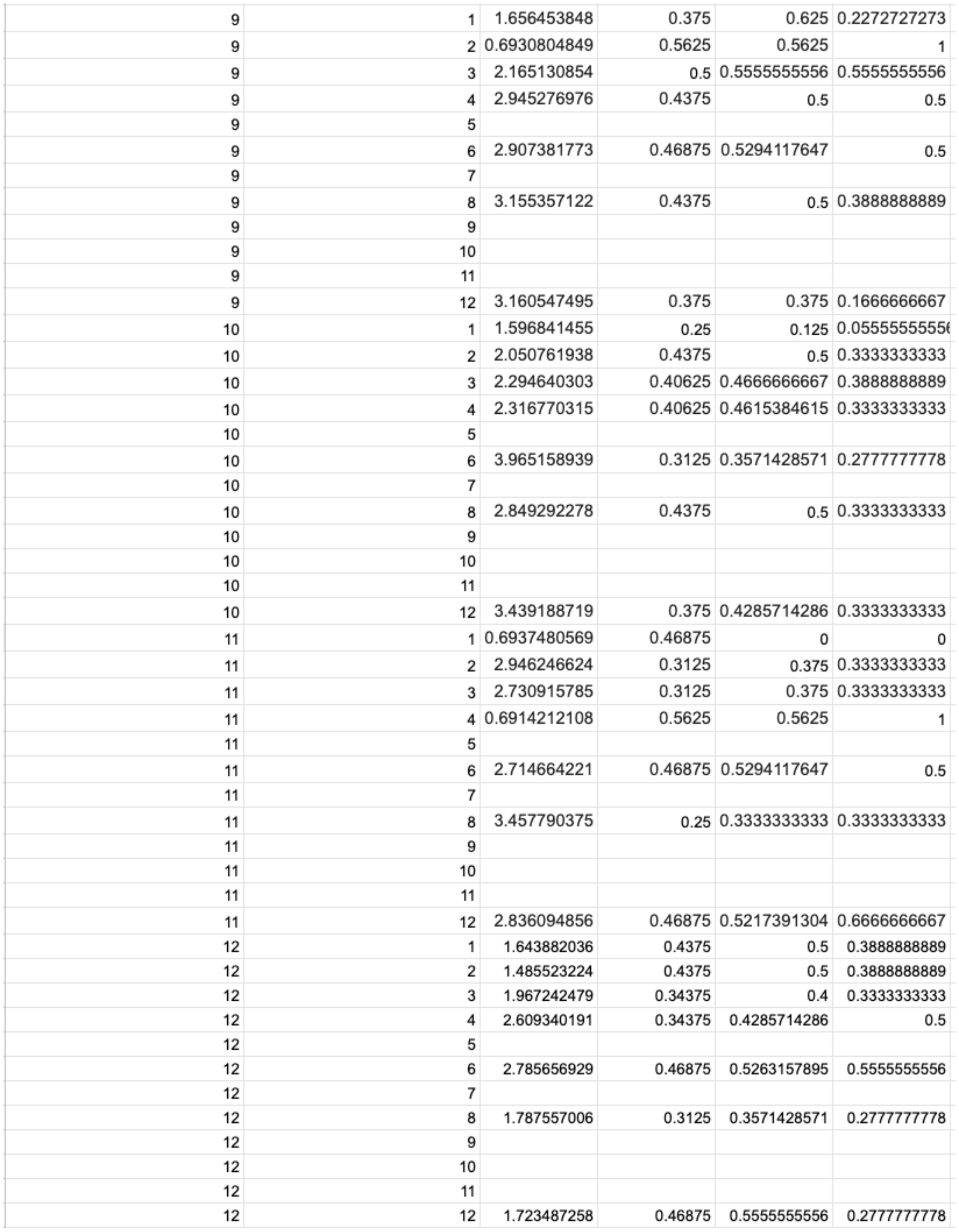
Num. Attention Heads	Num. Hidden Layers	eval_loss	eval_accuracy	eval_precision	eval_recall
1	1	0.6863837838	0.5625	0.5625	1
1	2	0.6573162675	0.59375	0.6086956522	0.7777777778
1	3	1.187353134	0.3125	0.3888888889	0.3888888889
1	4	1.353464723	0.4375	0.5	0.5555555556
1	5				
1	6	1.851133466	0.40625	0.4761904762	0.5555555556
1	7				
1	8	1.586491823	0.4375	0.5	0.4444444444
1	9				
1	10				
1	11				
1	12	1.665474057	0.34375	0.4	0.3333333333
2	1	0.6992853284	0.5	0.5625	0.5
2	2	16.19025133	0.53125	0.5789473684	0.6111111111
2	3	1.989512682	0.40625	0.4782608696	0.6111111111
2	4	2.645609617	0.46875	0.5263157895	0.5555555556
2	5				
2	6	2.09959054	0.40625	0.4666666667	0.3888888889
2	7				
2	8	2.085278988	0.4375	0.5	0.5555555556
2	9				
2	10				
2	11				
2	12	2.780640602	0.28125	0.3076923077	0.2222222222
3	1	1.195669174	0.5	0.55	0.6111111111
3	2	2.921438217	0.3125	0.375	0.3333333333
3	3	3.099348068	0.21875	0.1818181818	0.1111111111
3	4	3.089171648	0.40625	0.4666666667	0.3888888889
3	5				
3	6	2.56967473	0.21875	0.1111111111	0.05555555556
3	7				
3	8	2.858081579	0.4375	0.5	0.3888888889
3	9				
3	10				
3	11				
3	12	3.224216461	0.3125	0.375	0.3333333333
4	1	0.7234803438	0.59375	0.6	0.8333333333
4	2	2.439064741	0.375	0.4166666667	0.2777777778
4	3	3.120898485	0.21875	0.2666666667	0.2222222222
4	4	3.407148361	0.28125	0.3529411765	0.3333333333
4	5				
4	6	3.258265972	0.375	0.4285714286	0.3333333333
4	7				
4	8	3.082842588	0.375	0.4285714286	0.3333333333
4	9				
4	10				
4	11				
4	12	3.119358063	0.28125	0.3333333333	0.2777777778

Figure 42: Fine-Tuned BERT - Evaluation Metrics (own creation)



5	1	1.161076188	0.34375	0.4210526316	0.4444444444
5	2	2.780708313	0.40625	0.4666666667	0.3888888889
5	3	2.659687996	0.5625	0.5909090909	0.7222222222
5	4	2.733124971	0.3125	0.3888888889	0.3888888889
5	5				
5	6	3.442130327	0.375	0.4285714286	0.3333333333
5	7				
5	8	2.779936314	0.375	0.4285714286	0.3333333333
5	9				
5	10				
5	11				
5	12	3.767014027	0.375	0.4375	0.3888888889
6	1	0.9577344656	0.46875	0.52	0.7222222222
6	2	1.937443376	0.46875	0.5333333333	0.4444444444
6	3	3.04244709	0.34375	0.3636363636	0.2222222222
6	4	2.51503396	0.34375	0.3636363636	0.2222222222
6	5				
6	6	3.661456823	0.375	0.4375	0.3888888889
6	7				
6	8	3.535592794	0.375	0.4375	0.3888888889
6	9				
6	10				
6	11				
6	12	3.626071215	0.34375	0.3846153846	0.2777777778
7	1	1.917945504	0.46875	0.5238095238	0.6111111111
7	2	1.892106891	0.375	0.4285714286	0.3333333333
7	3	2.882991791	0.34375	0.4	0.3333333333
7	4	2.752112627	0.40625	0.4761904762	0.5555555556
7	5				
7	6	3.709526062	0.40625	0.4666666667	0.3888888889
7	7				
7	8	3.599761248	0.40625	0.4666666667	0.3888888889
7	9				
7	10				
7	11				
7	12	3.330888271	0.375	0.4375	0.3888888889
8	1	0.6845277548	0.5625	0.5625	1
8	2	0.8463827968	0.40625	0.4444444444	0.2222222222
8	3	2.511162996	0.53125	0.6	0.5
8	4	2.678747177	0.46875	0.5294117647	0.5
8	5				
8	6	3.221230745	0.28125	0.3076923077	0.2222222222
8	7				
8	8	3.384039879	0.34375	0.3846153846	0.2777777778
8	9				
8	10				
8	11				
8	12	3.875358343	0.34375	0.4117647059	0.3888888889

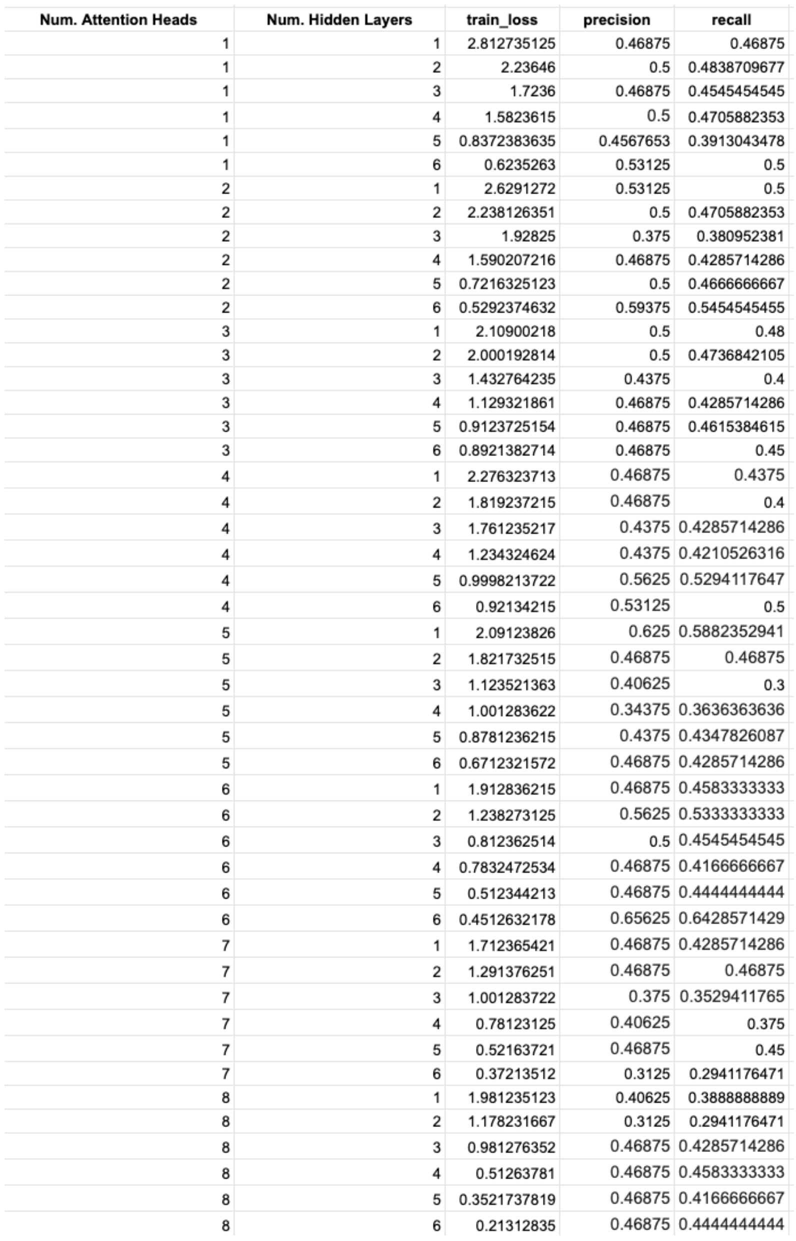
Figure 43: Fine-Tuned BERT - Evaluation Metrics (own creation)



9	1	1.656453848	0.375	0.625	0.2272727273
9	2	0.6930804849	0.5625	0.5625	1
9	3	2.165130854	0.5	0.5555555556	0.5555555556
9	4	2.945276976	0.4375	0.5	0.5
9	5				
9	6	2.907381773	0.46875	0.5294117647	0.5
9	7				
9	8	3.155357122	0.4375	0.5	0.3888888889
9	9				
9	10				
9	11				
9	12	3.160547495	0.375	0.375	0.1666666667
10	1	1.596841455	0.25	0.125	0.0555555555
10	2	2.050761938	0.4375	0.5	0.3333333333
10	3	2.294640303	0.40625	0.4666666667	0.3888888889
10	4	2.316770315	0.40625	0.4615384615	0.3333333333
10	5				
10	6	3.965158939	0.3125	0.3571428571	0.2777777778
10	7				
10	8	2.849292278	0.4375	0.5	0.3333333333
10	9				
10	10				
10	11				
10	12	3.439188719	0.375	0.4285714286	0.3333333333
11	1	0.6937480569	0.46875	0	0
11	2	2.946246624	0.3125	0.375	0.3333333333
11	3	2.730915785	0.3125	0.375	0.3333333333
11	4	0.6914212108	0.5625	0.5625	1
11	5				
11	6	2.714664221	0.46875	0.5294117647	0.5
11	7				
11	8	3.457790375	0.25	0.3333333333	0.3333333333
11	9				
11	10				
11	11				
11	12	2.836094856	0.46875	0.5217391304	0.6666666667
12	1	1.643882036	0.4375	0.5	0.3888888889
12	2	1.485523224	0.4375	0.5	0.3888888889
12	3	1.967242479	0.34375	0.4	0.3333333333
12	4	2.609340191	0.34375	0.4285714286	0.5
12	5				
12	6	2.785656929	0.46875	0.5263157895	0.5555555556
12	7				
12	8	1.787557006	0.3125	0.3571428571	0.2777777778
12	9				
12	10				
12	11				
12	12	1.723487258	0.46875	0.5555555556	0.2777777778

Figure 44: Fine-Tuned BERT - Evaluation Metrics (own creation)

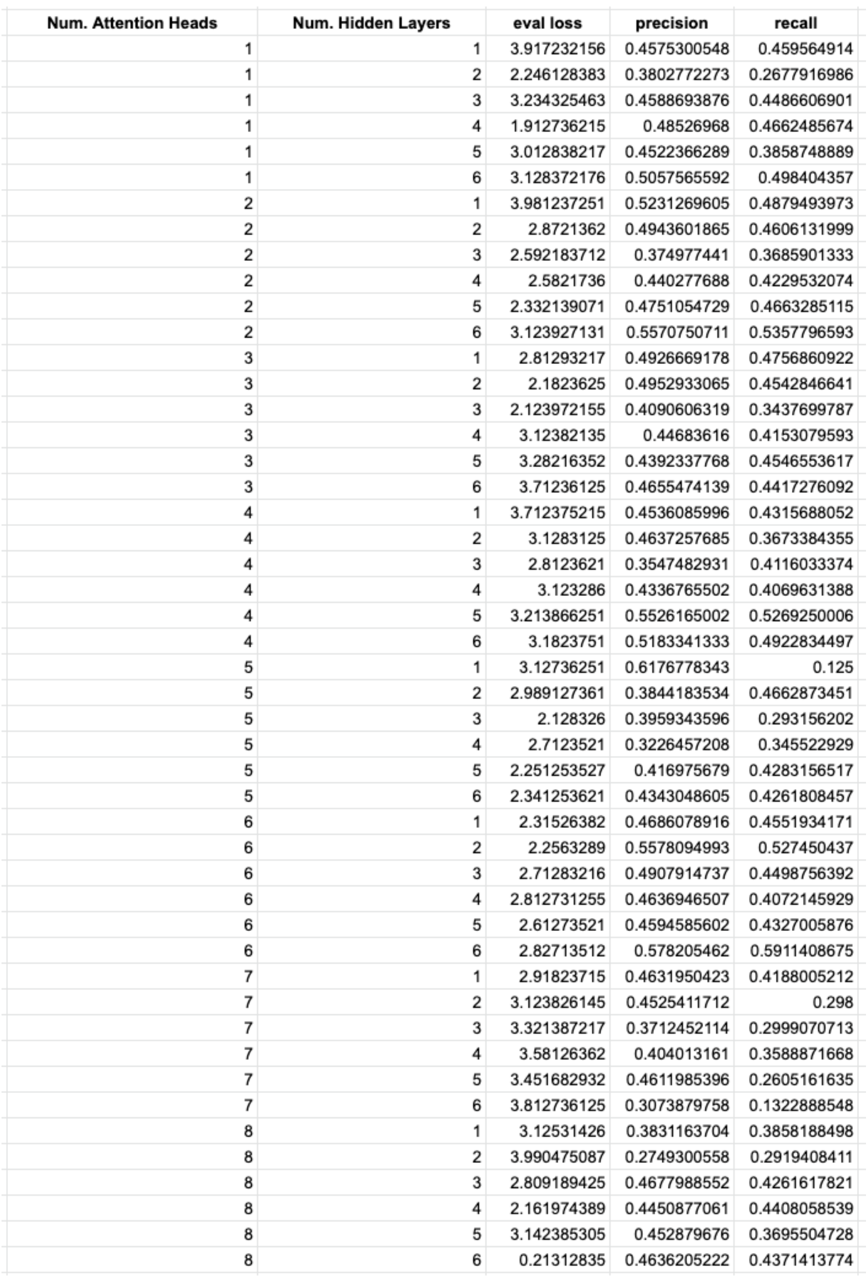
19 Annex G: Trained T5 data - Training Metrics



Num. Attention Heads	Num. Hidden Layers	train_loss	precision	recall
1	1	2.812735125	0.46875	0.46875
1	2	2.23646	0.5	0.4838709677
1	3	1.7236	0.46875	0.4545454545
1	4	1.5823615	0.5	0.4705882353
1	5	0.8372383635	0.4567653	0.3913043478
1	6	0.6235263	0.53125	0.5
2	1	2.6291272	0.53125	0.5
2	2	2.238126351	0.5	0.4705882353
2	3	1.92825	0.375	0.380952381
2	4	1.590207216	0.46875	0.4285714286
2	5	0.7216325123	0.5	0.4666666667
2	6	0.5292374632	0.59375	0.5454545455
3	1	2.10900218	0.5	0.48
3	2	2.000192814	0.5	0.4736842105
3	3	1.432764235	0.4375	0.4
3	4	1.129321861	0.46875	0.4285714286
3	5	0.9123725154	0.46875	0.4615384615
3	6	0.8921382714	0.46875	0.45
4	1	2.276323713	0.46875	0.4375
4	2	1.819237215	0.46875	0.4
4	3	1.761235217	0.4375	0.4285714286
4	4	1.234324624	0.4375	0.4210526316
4	5	0.9998213722	0.5625	0.5294117647
4	6	0.92134215	0.53125	0.5
5	1	2.09123826	0.625	0.5882352941
5	2	1.821732515	0.46875	0.46875
5	3	1.123521363	0.40625	0.3
5	4	1.001283622	0.34375	0.3636363636
5	5	0.8781236215	0.4375	0.4347826087
5	6	0.6712321572	0.46875	0.4285714286
6	1	1.912836215	0.46875	0.4583333333
6	2	1.238273125	0.5625	0.5333333333
6	3	0.812362514	0.5	0.4545454545
6	4	0.7832472534	0.46875	0.4166666667
6	5	0.512344213	0.46875	0.4444444444
6	6	0.4512632178	0.65625	0.6428571429
7	1	1.712365421	0.46875	0.4285714286
7	2	1.291376251	0.46875	0.46875
7	3	1.001283722	0.375	0.3529411765
7	4	0.78123125	0.40625	0.375
7	5	0.52163721	0.46875	0.45
7	6	0.37213512	0.3125	0.2941176471
8	1	1.981235123	0.40625	0.3888888889
8	2	1.178231667	0.3125	0.2941176471
8	3	0.981276352	0.46875	0.4285714286
8	4	0.51263781	0.46875	0.4583333333
8	5	0.3521737819	0.46875	0.4166666667
8	6	0.21312835	0.46875	0.4444444444

Figure 45: Trained T5 - Train Metrics (own creation)

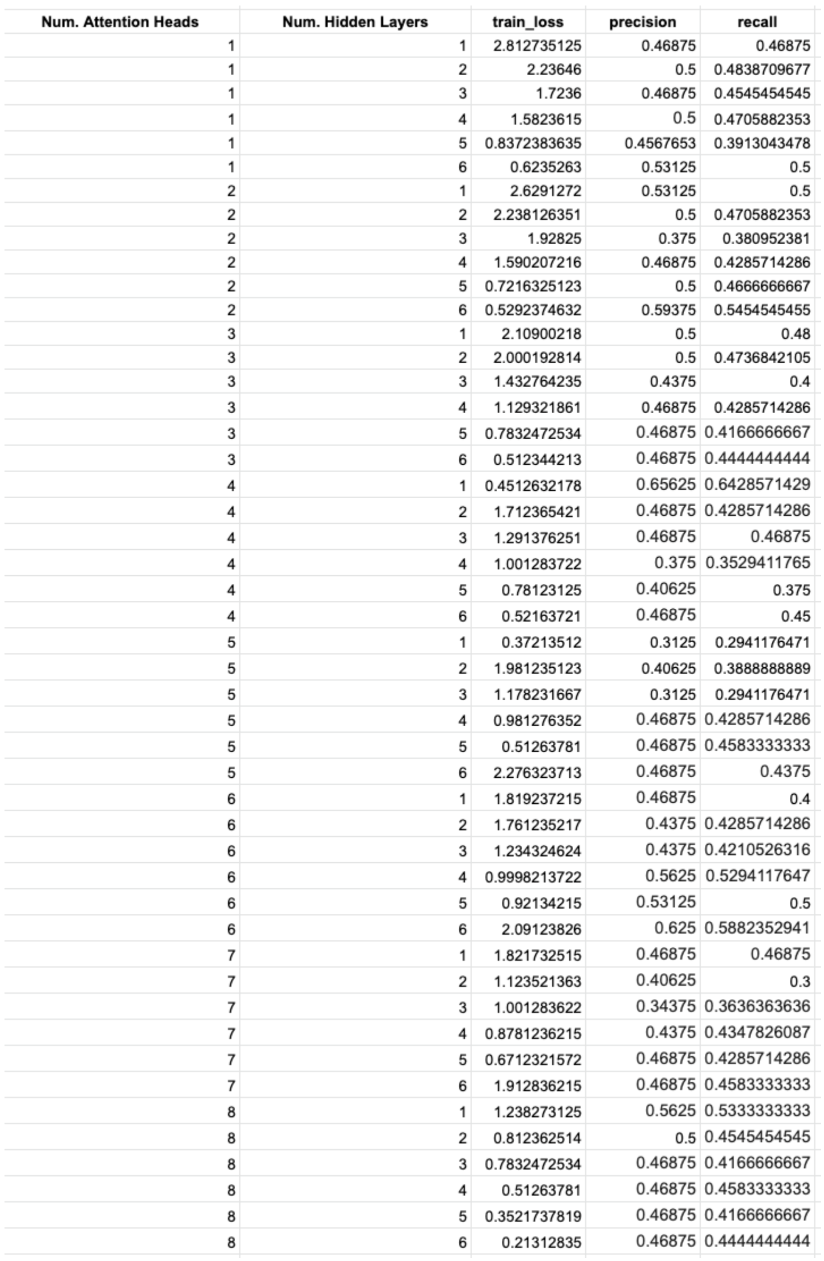
20 Annex H: Trained T5 data - Evaluation Metrics



Num. Attention Heads	Num. Hidden Layers	eval loss	precision	recall
1	1	3.917232156	0.4575300548	0.459564914
1	2	2.246128383	0.3802772273	0.2677916986
1	3	3.234325463	0.4588693876	0.4486606901
1	4	1.912736215	0.48526968	0.4662485674
1	5	3.012838217	0.4522366289	0.3858748889
1	6	3.128372176	0.5057565592	0.498404357
2	1	3.981237251	0.5231269605	0.4879493973
2	2	2.8721362	0.4943601865	0.4606131999
2	3	2.592183712	0.374977441	0.3685901333
2	4	2.5821736	0.440277688	0.4229532074
2	5	2.332139071	0.4751054729	0.4663285115
2	6	3.123927131	0.5570750711	0.5357796593
3	1	2.81293217	0.4926669178	0.4756860922
3	2	2.1823625	0.4952933065	0.4542846641
3	3	2.123972155	0.4090606319	0.3437699787
3	4	3.12382135	0.44683616	0.4153079593
3	5	3.28216352	0.4392337768	0.4546553617
3	6	3.71236125	0.4655474139	0.4417276092
4	1	3.712375215	0.4536085996	0.4315688052
4	2	3.1283125	0.4637257685	0.3673384355
4	3	2.8123621	0.3547482931	0.4116033374
4	4	3.123286	0.4336765502	0.4069631388
4	5	3.213866251	0.5526165002	0.5269250006
4	6	3.1823751	0.5183341333	0.4922834497
5	1	3.12736251	0.6176778343	0.125
5	2	2.989127361	0.3844183534	0.4662873451
5	3	2.128326	0.3959343596	0.293156202
5	4	2.7123521	0.3226457208	0.345522929
5	5	2.251253527	0.416975679	0.4283156517
5	6	2.341253621	0.4343048605	0.4261808457
6	1	2.31526382	0.4686078916	0.4551934171
6	2	2.2563289	0.5578094993	0.527450437
6	3	2.71283216	0.4907914737	0.4498756392
6	4	2.812731255	0.4636946507	0.4072145929
6	5	2.61273521	0.4594585602	0.4327005876
6	6	2.82713512	0.578205462	0.5911408675
7	1	2.91823715	0.4631950423	0.4188005212
7	2	3.123826145	0.4525411712	0.298
7	3	3.321387217	0.3712452114	0.2999070713
7	4	3.58126362	0.404013161	0.3588871668
7	5	3.451682932	0.4611985396	0.2605161635
7	6	3.812736125	0.3073879758	0.1322888548
8	1	3.12531426	0.3831163704	0.3858188498
8	2	3.990475087	0.2749300558	0.2919408411
8	3	2.809189425	0.4677988552	0.4261617821
8	4	2.161974389	0.4450877061	0.4408058539
8	5	3.142385305	0.452879676	0.3695504728
8	6	0.21312835	0.4636205222	0.4371413774

Figure 46: Trained T5 - Evaluation Metrics (own creation)

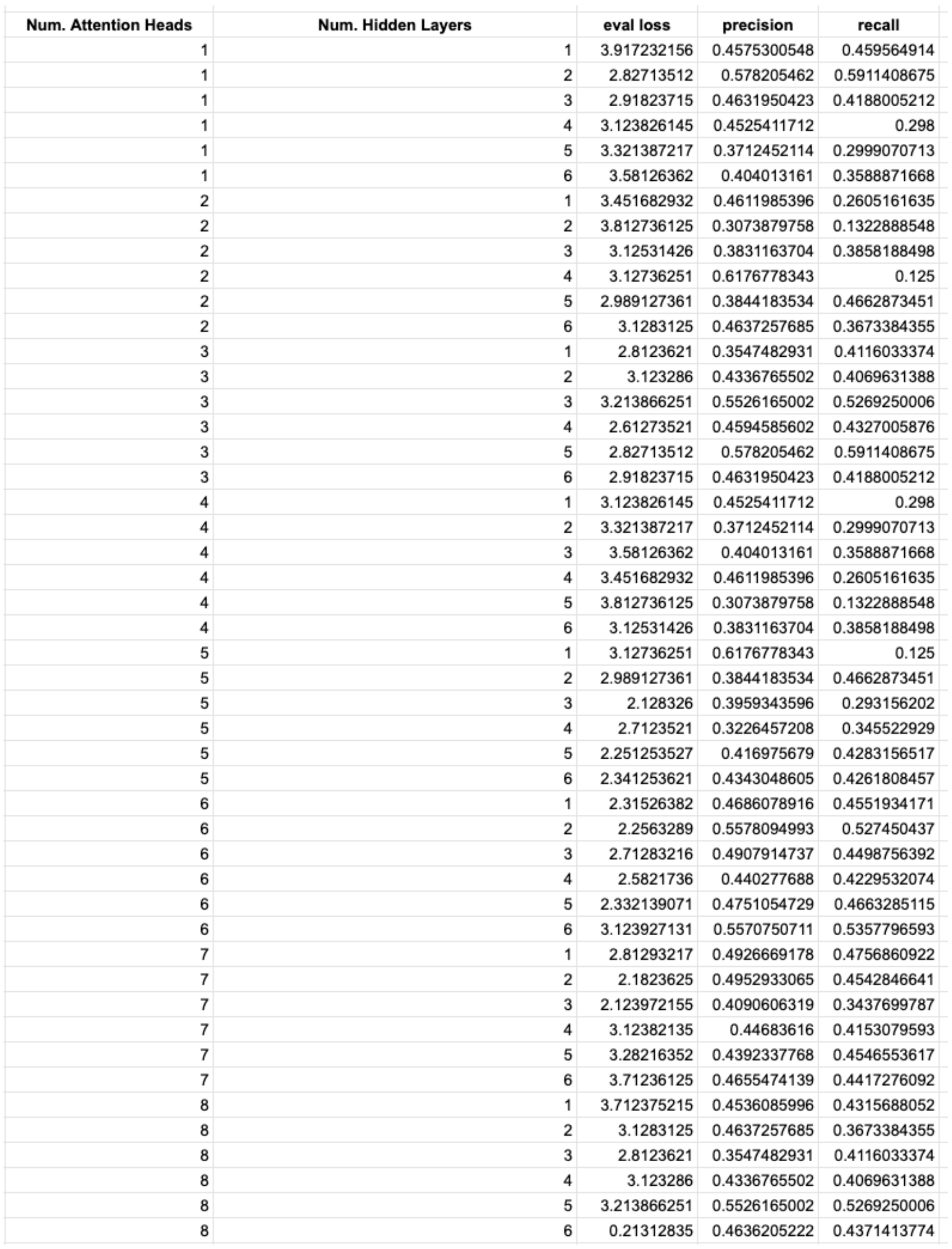
21 Annex I: Fine-Tuned T5 data - Evaluation Metrics



Num. Attention Heads	Num. Hidden Layers	train_loss	precision	recall
1	1	2.812735125	0.46875	0.46875
1	2	2.23646	0.5	0.4838709677
1	3	1.7236	0.46875	0.4545454545
1	4	1.5823615	0.5	0.4705882353
1	5	0.8372383635	0.4567653	0.3913043478
1	6	0.6235263	0.53125	0.5
2	1	2.6291272	0.53125	0.5
2	2	2.238126351	0.5	0.4705882353
2	3	1.92825	0.375	0.380952381
2	4	1.590207216	0.46875	0.4285714286
2	5	0.7216325123	0.5	0.4666666667
2	6	0.5292374632	0.59375	0.5454545455
3	1	2.10900218	0.5	0.48
3	2	2.000192814	0.5	0.4736842105
3	3	1.432764235	0.4375	0.4
3	4	1.129321861	0.46875	0.4285714286
3	5	0.7832472534	0.46875	0.4166666667
3	6	0.512344213	0.46875	0.4444444444
4	1	0.4512632178	0.65625	0.6428571429
4	2	1.712365421	0.46875	0.4285714286
4	3	1.291376251	0.46875	0.46875
4	4	1.001283722	0.375	0.3529411765
4	5	0.78123125	0.40625	0.375
4	6	0.52163721	0.46875	0.45
5	1	0.37213512	0.3125	0.2941176471
5	2	1.981235123	0.40625	0.3888888889
5	3	1.178231667	0.3125	0.2941176471
5	4	0.981276352	0.46875	0.4285714286
5	5	0.51263781	0.46875	0.4583333333
5	6	2.276323713	0.46875	0.4375
6	1	1.819237215	0.46875	0.4
6	2	1.761235217	0.4375	0.4285714286
6	3	1.234324624	0.4375	0.4210526316
6	4	0.9998213722	0.5625	0.5294117647
6	5	0.92134215	0.53125	0.5
6	6	2.09123826	0.625	0.5882352941
7	1	1.821732515	0.46875	0.46875
7	2	1.123521363	0.40625	0.3
7	3	1.001283622	0.34375	0.3636363636
7	4	0.8781236215	0.4375	0.4347826087
7	5	0.6712321572	0.46875	0.4285714286
7	6	1.912836215	0.46875	0.4583333333
8	1	1.238273125	0.5625	0.5333333333
8	2	0.812362514	0.5	0.4545454545
8	3	0.7832472534	0.46875	0.4166666667
8	4	0.51263781	0.46875	0.4583333333
8	5	0.3521737819	0.46875	0.4166666667
8	6	0.21312835	0.46875	0.4444444444

Figure 47: Fine-Tuned T5 - Train Metrics (own creation)

22 Annex J: Fine-Tuned T5 data - Evaluation Metrics



Num. Attention Heads	Num. Hidden Layers	eval loss	precision	recall
1	1	3.917232156	0.4575300548	0.459564914
1	2	2.82713512	0.578205462	0.5911408675
1	3	2.91823715	0.4631950423	0.4188005212
1	4	3.123826145	0.4525411712	0.298
1	5	3.321387217	0.3712452114	0.2999070713
1	6	3.58126362	0.404013161	0.3588871668
2	1	3.451682932	0.4611985396	0.2605161635
2	2	3.812736125	0.3073879758	0.1322888548
2	3	3.12531426	0.3831163704	0.3858188498
2	4	3.12736251	0.6176778343	0.125
2	5	2.989127361	0.3844183534	0.4662873451
2	6	3.1283125	0.4637257685	0.3673384355
3	1	2.8123621	0.3547482931	0.4116033374
3	2	3.123286	0.4336765502	0.4069631388
3	3	3.213866251	0.5526165002	0.5269250006
3	4	2.61273521	0.4594585602	0.4327005876
3	5	2.82713512	0.578205462	0.5911408675
3	6	2.91823715	0.4631950423	0.4188005212
4	1	3.123826145	0.4525411712	0.298
4	2	3.321387217	0.3712452114	0.2999070713
4	3	3.58126362	0.404013161	0.3588871668
4	4	3.451682932	0.4611985396	0.2605161635
4	5	3.812736125	0.3073879758	0.1322888548
4	6	3.12531426	0.3831163704	0.3858188498
5	1	3.12736251	0.6176778343	0.125
5	2	2.989127361	0.3844183534	0.4662873451
5	3	2.128326	0.3959343596	0.293156202
5	4	2.7123521	0.3226457208	0.345522929
5	5	2.251253527	0.416975679	0.4283156517
5	6	2.341253621	0.4343048605	0.4261808457
6	1	2.31526382	0.4686078916	0.4551934171
6	2	2.2563289	0.5578094993	0.527450437
6	3	2.71283216	0.4907914737	0.4498756392
6	4	2.5821736	0.440277688	0.4229532074
6	5	2.332139071	0.4751054729	0.4663285115
6	6	3.123927131	0.5570750711	0.5357796593
7	1	2.81293217	0.4926669178	0.4756860922
7	2	2.1823625	0.4952933065	0.4542846641
7	3	2.123972155	0.4090606319	0.3437699787
7	4	3.12382135	0.44683616	0.4153079593
7	5	3.28216352	0.4392337768	0.4546553617
7	6	3.71236125	0.4655474139	0.4417276092
8	1	3.712375215	0.4536085996	0.4315688052
8	2	3.1283125	0.4637257685	0.3673384355
8	3	2.8123621	0.3547482931	0.4116033374
8	4	3.123286	0.4336765502	0.4069631388
8	5	3.213866251	0.5526165002	0.5269250006
8	6	0.21312835	0.4636205222	0.4371413774

Figure 48: Fine-Tuned T5 - Evaluation Metrics (own creation)

References

[1] Hugging face – the ai community building the future.

[2] What are neural networks?

[3] Team rédac. Perceptron : Qu'est-ce que c'est et à quoi ça sert ?, Sep 2022.

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[5] Simeon Kostadinov. How recurrent neural networks work, Nov 2019.

[6] Home - colah's blog.

[7] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2019.

[8] State of the art nlp.

[9] Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J. Liu. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization, 2019.

[10] Sweta. Batch&nbsp;, mini batch and stochastic gradient descent, Aug 2020.

[11] Leonie Monigatti. A visual guide to learning rate schedulers in pytorch, Dec 2022.

[12] What is natural language processing?

[13] Natural language processing, Dec 2022.

[14] Marcello Politi. Fine-tuning for domain adaptation in nlp, May 2022.

[15] Why tensorflow.

[16] Pytorch.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[18] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

[19] Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training. 2018.

[20] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[21] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[22] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface's transformers: State-of-the-art natural language processing, 2019.

[23] Kexin Wang, Nils Reimers, and Iryna Gurevych. Tsdae: Using transformer-based sequential denoising auto-encoder for unsupervised sentence embedding learning, 2021.

[24] Kexin Wang, Nandan Thakur, Nils Reimers, and Iryna Gurevych. Gpl: Generative pseudo labeling for unsupervised domain adaptation of dense retrieval, 2021.

[25] Hasan Tercan, Alexandro Guajardo, and Tobias Meisen. Industrial transfer learning: Boosting machine learning in production. In *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, volume 1, pages 274–279, 2019.

[26] Yoshua Bengio. Deep learning of representations for unsupervised and transfer learning. In *Proceedings of the 2011 International Conference on Unsupervised and Transfer Learning Workshop - Volume 27*, UTLW'11, page 17–37. JMLR.org, 2011.

[27] Site built by: Salary.com. Benchmark electronics inc packaging development engineering senior manager salary.

[28] Ivy - the unified machine learning framework.

[29] Enterprise ml/nlp products and solutions for semantic search and question answering.

[30] Deepanshi. Artificial neural network: Beginners guide to ann, May 2021.

[31] Anjali Bhardwaj. What is a perceptron? – basics of neural networks, Oct 2020.

[32] Simeon Kostadinov. Understanding backpropagation algorithm, Aug 2019.

[33] Christian Bakke Vennerød, Adrian Kjærran, and Erling Stray Bugge. Long short-term memory rnn, 2021.

[34] Yuan Gao and Dorota Glowacka. Deep gate recurrent neural network, 2016.

[35] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation, 2015.

[36] Rick Merritt. What is a transformer model?, Sep 2022.

[37] Tianyang Lin, Yuxin Wang, Xiangyang Liu, and Xipeng Qiu. A survey of transformers, 2021.

[38] Prakhar Mishra. Understanding t5 modelnbsp;: Text to text transfer transformer model, May 2021.

[39] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways, 2022.

[40] Chalothon Chootong and Timothy K. Shih. Tech-talk-sum: Fine-tuning extractive summarization and enhancing bert text contextualization for technological talk videos. *Multimedia Tools and Applications*, 81(22):31295–31312, 2022.

[41] Alternativeto.

[42] Iso/iec awi ts 17847.

[43] Iso/iec cd 5259-2.

[44] Iso/iec cd 5259-3.

[45] Iso/iec cd 5259-4.

[46] Iso/iec cd 5259-5.

[47] Iso/iec ts 4213:2022.

[48] Iso/iec awi ts 17847.

[49] Github pipeline.

[50] Compute engine: Virtual machines (vms) nbsp;—nbsp; google cloud.

[51] Keras Team. Keras documentation: Callbacks api.

[52] Jason Brownlee. Understand the impact of learning rate on neural network performance, Sep 2020.

[53] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics.

[54] Prakhar Mishra. Automated metrics for evaluating the quality of text generation, Jul 2022.

[55] Yang Liu. Fine-tune bert for extractive summarization, 2019.

[56] Joshua Maynez, Shashi Narayan, Bernd Bohnet, and Ryan McDonald. On faithfulness and factuality in abstractive summarization, 2020.

[57] Diyah Puspitaningrum. A survey of recent abstract summarization techniques, 2021.

[58] Anna Glazkova and Dmitry Morozov. Applying transformer-based text summarization for keyphrase generation, 2022.

[59] Rouge (metric), Nov 2022.

[60] What are intelligent systems: Computer science amp; engineering.

[61] Tensorboard nbsp;: nbsp; tensorflow.