Universiteit Leiden

Master Computer Science

Resource-constrained Neural Architecture Search on DistilBERT for the task of question-answering

Name:	Andreas Paraskeva
Student ID:	s3252485
Date:	December, 2023
Specialisation:	Artificial Intelligence
1st supervisor:	Jan N. van Rijn
2nd supervisor:	João Pedro Correia dos Reis
3rd supervisor:	Suzan Verberne

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

The field of Automated Machine Learning and language models have witnessed significant research, with a considerable impact. Language models, typically built on transformer-based architectures, have achieved significant milestones in natural language processing. However, they come with a set of challenges, primarily stemming from their associated compute footprint. The application of AutoML to language models can lead to their democratization and foster further research, development, and usage of these architectures. We report on a case-study that does neural architecture search, with the aim of optimizing the backbone of the DistilBERT model in a resource-constrained environment (using a compute budget of $6\,500$ GPU-hours). We perform this neural architecture search on the DistilBERT model using an evolutionary algorithm. Despite the selection of a relatively small language model, we show that the cost of searching for a novel architecture for such models is immense and we propose several techniques to alleviate the linked cost of pre-training the candidate models. We propose a two-level hierarchical search space by incorporating macro-level architecture hyperparameters to a cell-based search. Furthermore, we present a segmented pipeline that allows for individual investigation and enhancement of its different components. Experimental results depict an efficiently diverse exploration method across generations. Additionally, findings reveal a strong correlation between pre-training performance and downstream task performance, suggesting its potential applicability as a cutoff criterion during model training. Finally, our learning curves analysis emphasizes the potential for efficient resource allocation through the adoption of an epoch-level stopping strategy, thus directing resources toward more promising candidate models. Future work should focus on scaling these insights to larger language models and a more diverse set of downstream tasks.

Contents

List of Figures

List of Tables

1 Introduction

The recent progress in the fields of **L**arge **L**anguage **M**odels (LLMs) [1] has captivated the attention of both researchers and users. Rapid breakthroughs in the field of **N**atural **L**anguage **P**rocessing (NLP) have been demonstrated through the release of the transformer architecture [2], and many models that followed - stemming from this architectural design - such as **B**idirectional **E**ncoder **R**epresentations from **T**ransformers (BERT) [3], versions of **G**enerative **P**re-trained **T**ransformer (GPT) [4], LLaMa [5] and OpenAssistant [6]. These have provided remarkable capabilities in terms of language comprehension and generation, whilst also paving the way for democratization in the field of NLP. Notably, platforms like HuggingFace [7] play a pivotal role by empowering developers, researchers, and organizations. Through the provision of access to pre-trained models, they facilitate the development of innovative applications and foster community-driven growth in NLP research.

The field of **Auto**mated **M**achine **L**earning (AutoML) [8] introduces the notion of utilizing experience gained from previously seen datasets and plays a supporting role in automating the identification of an appropriate machine learning pipeline, based on the task at hand. The fundamental objective is to democratize machine learning by enabling researchers to apply their expertise to more high-value tasks while automating routine processes. The exploration of better-performing architectures has been an extensively studied domain, highlighting the evident challenges in the design and tuning of neural networks. The constrained intuitive understanding of their effective design is an obstacle hindering the ability to explore architectures, as humans are generally less inclined to investigate more complex architectures. These have led to the inception of the field of **N**eural **A**rchitecture **S**earch (NAS) [9], where the task of designing an architecture is automated.

Our study delves into the complex realm of LLM, characterized by their transformer-based architecture and an extensive number of trainable parameters. The first objective is to dissect the macro-architecture of these models, deriving an effective search space encapsulating diverse models for potential exploration. Additionally, we seek to implement a search methodology that leverages a multi-objective optimization approach, with the primary goal being the optimization of the performance of the model on the downstream task of question-answering, whilst secondary minimizing the explored candidate models' size. Motivated by the convergence of AutoML and LLMs, our research acknowledges challenges [10]. Some of these challenges include the extreme computational cost of pre-training a base LLM, as well as the complexity of the application of an AutoML task which becomes even more relevant due to the necessity of its application across different lifecycle steps of an LLM. An LLM ideally should be improved in its pre-training, finetuning and inference phases. Associated with the learning phases are a considerable number of learning paradigms, thus the need for identifying or optimizing one is also evident. To address some of these challenges, we propose counter-measures, aiming to mitigate their impact and identify potential solutions for feasibility in a resource-constrained environment.

Through this research, we will report on insights derived from the application of NAS on language models in a resource-constrained environment. Due to the extensive computational costs, this research is conducted as a case-study to see whether a reasonable setup works, rather than a scientific study towards which of the various options perform best. In such a resource-constrained environment, this research utilized $6\,500$ GPU-hours of *NVIDIA A100* GPUs with 40GB VRAM. The need for utilizing these resources is associated with the pre-training phase, which utilizes large corpus of data and the benefit of higher GPU memory is evident in terms of training time. The proposed NAS methodology utilized the pre-trained model DistilBERT [1] from the HuggingFace platform, which is pre-trained on the task of **M**asked **L**anguage **M**odeling (MLM). While larger models exist, due to the computational cost of pre-training such models, we choose this relatively smaller language model of approximately 66 million parameters. As such, we will refer to DistilBERT as a language model, rather than an LLM. The (large) language models training procedure consists potentially of two phases, (i) a pre-training phase on a large corpus of (often unlabelled) text, which is typically expensive, and (ii) a fine-tuning phase on the task of interest, which is computationally much cheaper.

As previously mentioned, the first challenge for the application of NAS on language models is the associated need for pre-training the explored models. The proposed method for altering the model includes two searchable levels by adding macro-level architecture parameters to a cell-based search; derived from modules stemming from the architecture of the transformer encoder. These modules are more specifically subsets of the transformer encoder architecture or the transformer encoder itself. The search method is based on a Genetic Algorithm (GA) [11], and more specifically heavily influenced by **N**on-dominated **S**orting **G**enetic **A**lgoritm II (NSGA-II) [12] since we work towards optimizing generated models on the two previously mentioned objectives (performance and model size). Using the pre-trained model from HuggingFace we retrieve a set of weights for the original architecture, and any architectural changes applied to the model will only affect the weights of altered or added modules (which take randomly initialized weights). Another method to reduce the pre-training cost is to have a cut-off phase during the pre-training, where only half of the generated models will continue in the second phase of pre-training. The selection process follows the investigation of learning curves and only the best-performing models, in terms of loss on the MLM task, are selected. Another technique utilized to reduce the cost of pre-training is having two phases of pre-training, where the first one is focused on training solely modules that have been newly initialized, followed by training the whole architecture for calibration purposes.

The second challenge was the selection of appropriate datasets and accompanying learning paradigm for the task of pre-training. The pre-training phase uses the original

[1] This is the distilled version of Bert Base uncased provided by HuggingFace which is trained on English language using a **M**asked **L**anguage **M**odeling (MLM) objective.

datasets introduced for the training purposes of BERT and DistilBERT, namely BookCorpus [13] and English Wikipedia [14]. The training procedure follows the task of MLM, to ensure a rigorous and equitable comparison with the baseline model from HuggingFace. In regards to the finetuning phase, the **S**tanford **Qu**estion **A**nswering **D**ataset (SQUAD) [15] was utilized, in order to achieve benchmarking on the task of question-answering and compare the models' downstream task performance.

Aside from the innovations of the transformer architecture, one cannot refute the increasing complexity and huge number of trainable parameters accompanying it. These factors have necessitated a more automated and effective approach towards designing and optimizing their architecture, potentially identifying models with better performance and reduced compute footprint. This has raised the following question:

Can the application of neural architecture search, in a resource-constrained environment, aid the identification of better language model transformer-based architectures, in terms of performance on the task of question answering and the size of the model?

To address the research question, this study will focus on specific proposed options for the following tasks:

Search Space: Search space definition for transformer-based language models.

Multi-Objective Optimization: Approaching the problem as a multi-objective optimization, with a prioritization on a specific objective.

Cost efficient pre-training: Considerations and strategies aimed at the minimization of the cost of pre-training during the application of NAS.

This work proposes a novel search space for adapting the backbone of transformer encoders through the utilization of a segmented NAS pipeline. We highlight the potential applicability of learning curve analysis on language models and present experimental indications of a high correlation between the pre-training task and the downstream task, which indicates its applicability as a cutoff criterion for candidate models. Our findings and insights act as empirical evidence but more broad experiments and scaling to LLMs is necessary to solidify these.

This research has been conducted in collaboration with the company DEUS [2] (Amsterdam), and the research field aligns with their interest in identifying a good performing model for the task of question-answering, with the necessary considerations in terms of computational cost in its lifecycle. The aim is to make such a model's application to real-world scenarios feasible in resource-constrained environments, thus democratizing language models.

[2] https://www.deus.ai/

The outline of the thesis is conducted as follows. Section 2 provides an overview of the main architectural and methodology-related components that provide background knowledge for our research. Section 3 explores related work and research that influenced our decision process. Section 4 provides implementation details and the proposed methodology for answering our research questions. Section 5 covers details of our experimental setup, followed by Section 6 which depicts and analyzes the experimental results. Section 7 outlining the limitations and respective future work extensions, while Section 8 provides remarks and conclusions from our work.

2 Background

In this section, we will cover architectural and methodology-related components, which are deemed vital background knowledge of this project. We will expand upon the topics of evolutionary algorithms [11], the architecture of transformers [2], and neural architecture search [9].

2.1 Evolutionary Algorithms

Evolutionary Algorithms represent a class of adaptive computational search methods that rely on operations and techniques that are inspired by the process of natural and biological evolution [11]. The biological theory of evolution, based on the concept of natural selection, favors the best-fitted individuals, which are used to create the next generation through evolutionary operations. Similarly, evolutionary algorithms such as genetic algorithms, follow an iterative and sequential process of genetic operators and appropriate individual selection.

At the core of evolutionary algorithms is the concept of a *population*, collectively composed of candidate solutions. These candidates are represented through a suitable *genotype* (genetic encoding of the phenotype), depending on the specific problem domain. The next vital part of the evolutionary process is the definition of a *fitness evaluation function*, which is determined by our objective(s). The fitness value represents the quality of the individual and typically this should be optimized, by approaching it either as a maximization or minimization problem. As stated earlier promising individuals will be selected (according to their fitness values) in order to undergo the application of genetic operators and generate a new population (offspring). These operators can include *crossover/recombination* and *mutation*, and their application diverges the individuals from their ancestors. These also provide a balance between exploration and exploitation for traversing through the associated search space. It is essential to define carefully the selection process for both the mating selection (i.e. selection of fitted parents to create the offspring population) and next generation selection (i.e. individuals that compose the next generation of parents). Fig-

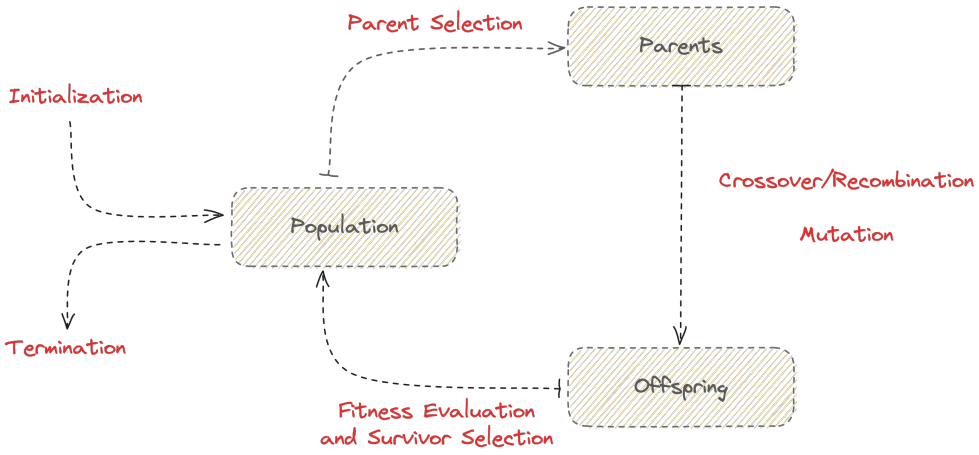


Figure 1: The general scheme of an Evolutionary Algorithm, represented in a flow chart format

ure 1 depicts the general scheme that is followed in an evolutionary algorithm, including the previously mentioned operators and processes.

2.1.1 Non-dominated Sorting Genetic Algorithm II (NSGA-II)

Amongst the vast list of evolutionary algorithms, is the NSGA-II [12]. This was developed to counter some of the initial shortcomings of early evolutionary algorithms. It has specifically emerged to tackle multi-objective optimization problems, where more than one objective needs to be optimized at the same time, with possibly conflicting interests. This can be depicted in a multi-dimensional space with the set of solutions known as the Pareto front.

The main idea behind the NSGA-II approach centers around the handling of non-dominated solutions and the iterative identification of these sets of solutions. It categorizes sequentially the non-dominated solution into fronts, removing the currently selected set and moving into the identification of the next one. It also introduces the concept of the crowding distance (distance of candidate to neighboring solutions), to prioritize the selection of the most diverse solution (less crowded ones, i.e. with the closest neighboring candidates furthest away). This provides the ability to preserve diversity by encouraging the spread of selected solutions, i.e. varying in their objective scores compared to other individuals in the same Pareto rank. Finally, its other substantial contribution was the addition of the aspect of elitism, where the best candidate solutions, called the elites, in each generation, are inserted into the next generation. This is achieved by selecting candidates for the next generation selection in order of Pareto front ranking and in case the population to be selected is smaller than the pool, the last front will undergo selection utilizing the crowding distance

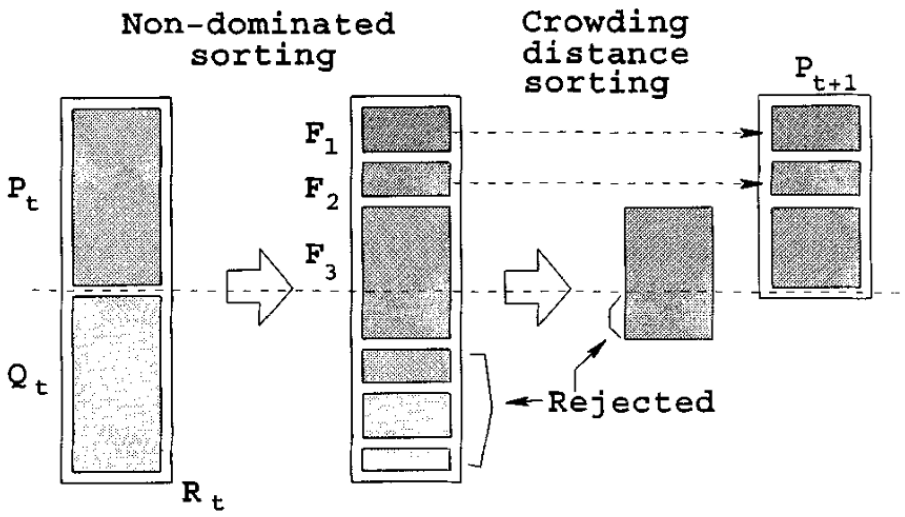


Figure 2: The outline of the NSGA-II next-generation selection procedure. R_t represents the combined population of P_t and Q_t, representing the parent and offspring population of generation t respectively. F_i represent the set of solutions included in the i$^{\text{th}}$ front in descending order of importance. Figure taken from [12].

as well. An outline of the next-generation selection procedure is depicted in Figure 2.

2.2 Transformers

Arguably, one of the most influential papers in deep learning research presented the architecture of transformers and attention layers [2]. This powerful neural architecture has been widely used in different NLP tasks, including language modeling, text generation, and text summarization. This relied on the concept of self-attention, providing a selective focus on different parts of the input sequence during the output generation. Moreover, the ability of transformers to process the whole input sequence at once allowed for computing relations between all pairs of the input tokens. In this section we will go over all the main parts that the transformers consist of, and how these are incorporated into its architectural design.

2.2.1 Architectural overview

In contrast to **R**ecurrent **N**eural **N**etwork (RNNs) and **L**ong **S**hort-**T**erm **M**emory (LSTM) models, that were previously **s**tate-**o**f-**t**he-**a**rt (SOTA), transformers have the ability to parallelize the processing of information across the whole input sequence. This allows them to identify relationships between the words and provide long-range dependencies across

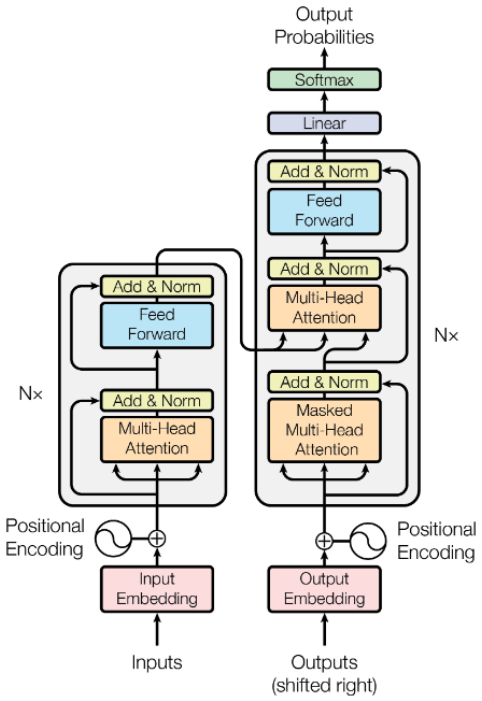


Figure 3: The architecture of the transformer is represented. The left part of this diagram depicts the transformer encoder, whilst the right one is the decoder. Figure taken from [2].

them. Moreover, transformers are autoregressive, meaning that the sequentially generated output tokens become part of the input in the next step.

Transformer-based models comprise of multiple stacked encoder layers, enabling the model to capture complex patterns and refine the representations of the input. The output representation of the encoder stack, known as the context representation is fed as input onto the decoder stack. In contrast to the encoders, decoders have a masked self-attention, which ensures visibility of information preceding the current position during the generation by masking tokens that would follow.

2.2.2 Attention Mechanism

The attention mechanism is a computational mechanism that provides the model with a selective focus on the input sequence during the output generation. Its purpose is the mapping of queries and a set of key-value pairs to an output. This is calculated as follows:

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) \cdot V$$

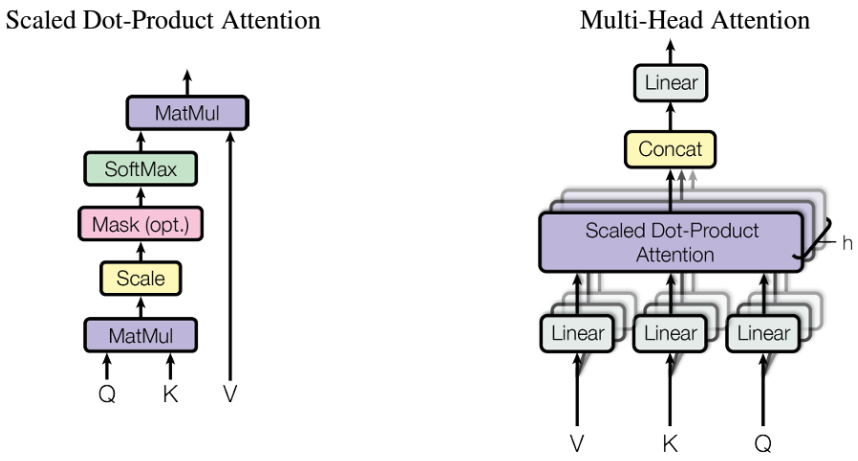


Figure 4: (left)Scaled Dot-Product Attention. (right)Multi-head attention consists of several attention layers running in parallel. Figure taken from [2]

where K represents the set of keys, V is the set of values and Q is the set of queries, and all of the above are vector representations (derived from the input sequence). d_k is the dimension of the key vectors. The softmax function normalizes the dot product between Q and K, resulting in the set of attention weights. It is worth noting that these attention weights are calculated based on the similarity of query and key, and can be utilized as both self-attention and cross-attention, capturing dependencies within and across sequences. The introduced "Scaled Dot-Product Attention" is visualized on the left side of Figure 4.

Naturally extending the concept of self-attention to multi-head self-attention enhances the ability of the model to encapsulate information of different types of dependencies across the input. Through the usage of multiple sets of the previously mentioned query (Q), keys (K), and values (V) matrices, the model can process input parts simultaneously. The following equation depicts its calculation process:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{Head}_1, \text{Head}_2, \ldots, \text{Head}_H) \cdot W^O$$

where Head_i represents the i^{th} attention head, with each attention head retaining its own set of weights. H represents the number of attention heads and W^O is a weight matrix that combines the outputs from all the attention heads. Each attention head can specialize in capturing different relationships across the data and has been proven effective on many NLP tasks. The multi-head self-attention can be visualized on the right side of Figure 4.

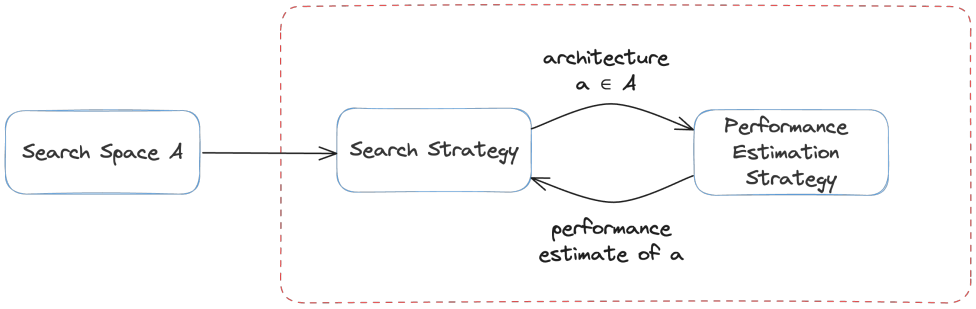


Figure 5: An overview of neural architecture search according to its three dimensions (Figure based on [17]). The search strategy iteratively selects an architecture (α) from the predefined search space A. The performance estimation strategy will output a prediction of its ultimate task. This will then be used to select the next architecture according to the prior knowledge.

2.2.3 Feed-Forward Neural Network

Each of the encoder and decoder layers contains a fully connected **F**eed-forward **N**eural **N**etwork (FNN). The original architecture of the FNN consists of two linear transformations and a RELU [16] activation function. This follows the multi-head self-attention, allowing the model to direct its focus on the local interaction of the data and thus enabling the learning of more complex features.

2.3 Neural Architecture Search

The high complexity of **N**eural **N**etworks (NNs), combined with the limited intuition in terms of designing them, presents an obstacle towards efficient and effective architectural design choices. This, in turn, constricts the explored search space, since humans are less likely to manually identify more complex and diverse architectures. Due to this, the need for more effective and possibly automated design choices is evident.

Neural Architecture Search has emerged as a subfield of **Auto**mated **M**achine **L**earning (AutoML) [8], which is the process of automating the steps of a machine learning pipeline. NAS focuses on the processes of designing the optimal neural network for a specified task. The main principle which guides the search and consequently the choice of architecture is the effect on the model's performance. Several concepts need to be introduced in order to provide a general overview of what components compile as NAS methodology. NAS methods are typically categorized based on three dimensions: the *search space*, the *search strategy*, and a *performance estimation strategy* [17] (see Figure 5). These components are explored further in this research project and thus will be explained in the subsequent sections.

2.3.1 Search space

The search space, in terms of a NAS algorithm, represents the realm of possible neural network configurations that would be likely to be explored during the execution of our algorithm. This can be defined based on various choices that would define the architecture of the NN, such as number of layers, type of layers, type of connections, activation functions, hyperparameters, etc. The defined search space aims to encapsulate valid and possible solutions whilst also limiting choices and as a result, constricting its size, since the aim would be to be able to explore a considerate portion of it.

A careful definition of the search space is a fundamental step in the process of NAS methodologies, and failure to provide this would result in unproductive exploration with the likelihood of identifying a superior architecture reduced. An architecture such as the transformer can be considered complex with a broader search space. Architectural decisions which would determine the possible candidates and the search space require thorough investigation.

2.3.2 Search strategy

Following the definition of the search space, we need to determine an intricate way to traverse it. This is achieved through the search strategy, which is an optimization technique, used to identify promising candidate architectures. These can typically lie under two main categories: black-box optimization techniques [18, 19] and one-shot techniques [20]. The focus of our research is black-box optimization techniques and more specifically investigation of the principles of EA.

During the efficient exploration of the search space, we would typically be interested in enhancing the performance of our baseline model, however, the model size would also be of importance. We can view these two objectives as our optimization objectives in a multi-objective setting. NSGA-II would operate as a "black-box optimizer" which would iteratively generate candidate architectures and through investigation of the two objective goals (under Pareto fronts) would determine the next generation of candidates and guide the search accordingly.

2.3.3 Performance estimation strategy

The performance estimation strategy refers to a technique that can be applied in order to retrieve an estimation of the ultimate model performance, which is one if not the sole optimization objective. The most simple approach would be following the standard procedure for training the model and evaluating it. This is undoubtedly extremely expensive, as proven by prior research as well [18], and would render the application of NAS on language models in resource-constrained environments infeasible. Some of the techniques that have

been researched and employed typically in other domains have been lower fidelity estimates [21] (data or epoch level), weight inheritance from parent model [22] and learning curve analysis and extrapolation [23]. The aforementioned techniques are implemented and evaluated in our research, in an attempt to optimize the cost of pre-training and derive insights on their potential applicability to language models.

3 Related work

This section will cover some of the key papers in the two respective fields, LLMs and NAS, which have provided breakthroughs and thus encouraged further research in these areas. Firstly, we will explore the recent work in the field of LLMs and NAS separately, and the analysis of literature for NAS targeted towards language models will follow.

3.1 Large Language Models

As previously highlighted, the transformer architecture [2] and the introduction of the attention mechanism have revolutionized the field of NLP, with breakthroughs in various tasks, including language modeling, text generation, and text summarization. The self-attention mechanism (see Section 2.2.2) allowed for the identification of relations across all the token embeddings in the input sequence in a parallelized fashion. This has permitted the processing, in terms of dependencies, for longer sequences, compared to previously state-of-the-art models which relied on **R**ecurrent **N**eural **N**etworks (RNNs) [24], **C**onvolution **N**eural **N**etworks (CNNs) [25] or **L**ong-**S**hort **T**erm **M**emory (LSTMs) [26] models.

The transformer architecture paved the way for the creation of LLMs since they are typically transformer-based. Among the most influential LLMs, the GPT [4] (a decoder-based transformer model) has showcased the potential of unsupervised pre-training and finetuning. Other models such as BERT [3] utilized bidirectional transformers (more specifically stacked transformer encoders) and a masked language model objective to capture rich contextual understanding of words from the input sequence. It was pre-trained on a large corpus based on the BookCorpus [13] and English Wikipedia [14] datasets. In the past, BERT achieved state-of-the-art results across various natural language processing tasks.

Text-**t**o-**T**ext **T**ransfer **T**ransformer (T5) [27] provided a single model for a diverse set of tasks such as classification, translation, summarization, and question answering. It utilized once again a transformer-based architecture and through casting all tasks as text-to-text transformations, it achieved the generation of output text given the respective input text.

These models have been extremely successful and impactful in the field of NLP. However, their deployment is computationally expensive and resource-demanding, making their applicability in many real-world applications infeasible. Due to our resource-constrained

environment, experimentation in this research will focus on a distilled [28] version of such a model. More specifically, the focus is DistilBERT [29], like the name suggests, a distilled version of BERT that aims to reduce the computational footprint while retaining substantial performance gains. This leverages knowledge distillation techniques, utilizing BERT as the teacher model. The reduced model size allowed for more flexibility in terms of the application of NAS, making the exploration of stemmed architectures feasible.

3.2 Neural Architecture Search

Designing a neural network can be a hard and time-consuming process. The limited intuition for designing them is an obstacle that restricts the explored architectures. These have led to the inception of the field of NAS, where the designing of an architecture is automated. One of the most influential papers in the field of NAS has been a paper where reinforcement learning has been integrated into NAS [18]. The main idea revolves around the utilization of an RNN controller, which is responsible for generating a string representing an NN (e.g. CNN). This architecture represents the child network, which is then trained and evaluated on the validation set. This performance metric is fed into a policy-based reinforcement agent as the reward, which in turn updates the parameters of the controller model. This iterative procedure is repeated to explore and more effectively identify a good child network. This was however limited to CNN or RNN-based models. Another obvious limitation or drawback of such an implementation would be the fact that it requires each child network to be fully trained which can be extremely computationally and time-wise expensive.

A recent approach to counter this black-box optimization search has been a one-shot search, where essentially one model is fully trained and the final model is a sub-network of the so-called *one-shot model* (also known as *supernetwork*). A famous approach following this is the **D**ifferentiable **Ar**chi**t**ure **S**earch (DARTS) [20], where continuous relaxation of the search space enables the use of gradient descent. It allows for the treatment of the search domain as a continuous space thus making it differentiable. This in turn allows for the identification of the local optimum model. Evolutionary algorithm-based methods were proposed to exploit evaluated candidate models for effective population maintenance and evolution [19, 12].

3.3 Neural Architecture Search on Language Models

Research in the area of CNN and RNN models has been quite extensive, with a heavy focus on the area of computer vision. Although limited, research in the area of transformers has now been steadily growing, with branching to NLP-related tasks. The *AutoTrans* [30] research paper focused on the discovery of good transformer-based architecture by automating the design choices in the model. This involves (i) setting the layer norm (which in turn

affects the training stability), (ii) where to put encoder attention, (iii) the number of layers, and many more. To achieve this they have employed an RL strategy through ENAS [31]. To further boost the performance and speed up the architectural search, they have utilized specialized parameter-sharing strategies for multi-head attention. The architecture search consists of two main phases. Initially, the child model with shared parameters (ω) is trained. The training of the θ parameters of the RNN controller follows, which is achieved by optimizing the expected reward by using REINFORCE [32].

A very interesting approach, dubbed as Primer [33], utilized a NAS algorithm based on an evolutionary algorithm that would search for a decoder-only auto-regressive language model. Two of the main modifications have been squaring ReLU activations and adding a depthwise convolution layer after each Q, K, and V projection in self-attention. These modifications should be applicable to other scenarios as well. The search space is defined as a collection of TensorFlow [34] applications (primitives) and evolution is applied to search for the model architectures. This research possesses interesting findings, however, it is accompanied by limitations. The main limitation would be the constricted application to TensorFlow, however, this can be countered by deriving the respective primitives for PyTorch [35] or other machine learning frameworks. Our focus has been a hierarchical-based search space that deviated from the idea of primitives, however, the identification of elemental blocks that contribute to the architectural design is an obvious similarity.

AutoTinyBERT [36] is an approach that focuses on the automatic optimization of hyperparameters, producing LLMs based on the architecture of BERT tailored towards resource-constrained environments. This approach applied NAS and hyperparameter tuning, by automatically generating different configurations that define the architecture of the transformer encoders and the number of stacked transformers; resulting in a homogenous architecture. Inspired by this, we define custom modules that serve as integral elements of the transformer encoder. This extends the possible exploration compared to solely relying on the configuration setup provided by HuggingFace [7] which can be viewed as a hyperparameter optimization task. It allows for independent configuration of the encoder blocks, whilst also endorsing deviations with the possible addition of such modules.

AutoBERT-Zero [37] is another evolutionary algorithm-based approach that focuses on the exploration of the backbone of the BERT model, by utilization of primitive mathematical operations in the intra-level aimed at the exploration of attention structures. On the inter-layer, they used convolutional layers to learn local dependencies. They proposed **O**peration **P**riority **N**eural **A**rchitecture **S**earch (OP-NAS) algorithm, to balance the tradeoff of exploration and exploitation, and another technique called Bi-Branch Weight Sharing in order to minimize the need for pre-training by sharing weights of modules of the supernet.

Through researching the above solutions, we can identify different techniques that aim to achieve a more balanced training procedure and also efficiently determine a search space and search method for traversing said space. It was determined that NAS through the usage of an evolutionary algorithm-based back-box optimization method enabled optimization

with retained knowledge of explored candidates. *AutoTinyBERT* [36] focused on a macro-search space that contained limited expressiveness but depicted very promising results. Inspired by the *AutoTinyBERT*, we developed a more expressive hierarchical search space, that defined primitive components of the transformer encoder backbone and added macro-level architecture hyperparameters to mutate the current architecture in a modular approach. Additionally, we retain weights associated with unaffected layers post-mutation [22] and investigate the effectiveness of epoch-level multi-fidelity in our NAS method. Changes can be applied independently to each of the modules resulting in diverse transformer encoders within the same model. Finally, inspired by techniques shared in *AutoBERT-Zero* [37], we can add modules into the encoder architecture, like the Feed Forward Neural Network, to potentially encourage more stability during training.

4 Proposed methods

In the context of NAS and its application to a specific model, two essential steps are the definition of an appropriate search space and an accompanying search method which should explore models within this search space. In the following section, we will provide an overview of both the search space definition and details of the search method, seen in Sections 4.1 and 4.2 respectively. It should be highlighted that due to the extensive computational costs, we conduct a case study to derive insights, rather than compare various alternative options for the definition of the search space and search strategy. It was necessary to perform simulations and small-case experiments, before the actual execution, to ensure that the pipeline executes successfully. The performance estimation strategy in our approach is involved in the intermediate step in the two-phase pre-training which is outlined in Section 4.2.4.

4.1 Search Space Definition

To provide elaborate and carefully defined search space for our NAS method applied to DistilBERT, we first needed to explore and investigate its architecture. It comprises a stack of transformer encoder blocks which consist of the following counterparts:

Multi-head self-attention allows the model to focus its attention on different parts of the input sequence simultaneously. This component allows the model to capture information encapsulating different relationships within the data and provide a comprehensive representation of those relationships.

Layer Normalization This component is used to stabilize and speed up the training through normalizing activation within each layer. It is placed in between the multi-head self-attention and feed-forward neural network layers.

Feed-forward Neural Network layer (FNN) This component consists of two linear transformations with an activation function in between (in this case GELU). This architecture allows for the modeling of non-linear relationships in the data and the representation of high-level features.

Output Layer The final component of the transformer block takes the output of the FNN and combines it with the output of the multi-head self-attention mechanism. This is then fed through a linear representation which represents the final output of the encoder block and consequently the input of the transformer encoder block to follow.

Analysis of these critical architectural components has led to the decision of a modular component-based approach for the definition of the search space. As explained above, the multi-head self-attention allows the model to focus on different parts of the input sequence simultaneously, capturing relationships within the data. The FNN introduces non-linearity, allowing the capture of intricate patterns and high-level features. Arguably the two aforementioned components are the most crucial parts of the transformer encoder and we introduce enhanced searchability to a cell-based approach by incorporating macro-level architecture hyperparameters; thereby incorporating two levels of searchability in a hierarchical search space. Therefore, we utilize the most important components of the transformer encoder block - including the multi-head self-attention, the FNN, and the transformer encoder as a whole - and by definition of a parameter space, we create blocks of these components. The utilization of these blocks and their integration in our mutation function will be explained more in-depth in the subsequent Section 4.2.1. The transformer encoder block is an identical copy of the block in the original model of DistilBERT. The remaining modules do retain the same general structure (as the original model) but their respective parameter options are outlined hereafter.

1. Multi-head Self-attention:

   Number of attention heads defines the number of attention heads that are included in the associated component. Each attention head focused on different aspects or patterns within the data allowing parallelization. More attention heads allow for the capturing of a richer set of information and understanding of complex patterns in the data, however, it comes with an associated computational cost and memory usage.

2. Feed-forward Neural Network:

   activation function defines the activation to be used in between the linear layers of the FNN. The options are [*ReLU* [16], *GELU* [38]], in contrast to the original option being solely *GELU*.

intermediate size defines the dimensionality of the hidden layers of the FNN. If there are more than two stacked linear layers then we can alter the intermediate size of the middle layers.

number of layers defines the number of stacked linear layers (minimum value of 2).

All of the hyperparameters mentioned above have a parameter space that spans relatively close to the original values of the model of DistilBERT and follows similar settings to the AutoTinyBert[36].

4.2 Genetic Algorithm Pipeline

After formulating the search space, the next logical and crucial step was the definition of the search method. This is based on techniques from genetic algorithms and more specifically the NSGA-II approach. The complete pipeline of the proposed search method is depicted in Figure 6. This can be visualized in incremental steps, which are outlined in detail in the subsequent sections.

The choice for NSGA-II was based on the fact that it has been proven empirically and through literature studies that it can be an effective way to apply evolutionary algorithms to a multi-objective problem. Modifications applied to the original algorithm, driven by intuition, empirical analysis derived from small-scale experiments, and the limitations imposed by the available project resources. Pivotal decisions at various stages of the pipeline, as well as this adaptation, will be analyzed and explained in subsequent sections.

4.2.1 Mutation function

The mutation function of the GA-based pipeline is important since it will create our candidate models and guide the traversal through the search space with considerations towards the exploration and exploitation trade-off. The modules of *Multi-head self-attention*, *Feed-forward Neural Network* and *transformer encoder block* are used to make changes to existing models and create the new candidate models of our search method.

Our first option would be to add or remove probabilistically the entirety of the encoder block. This is achieved through either the addition or deletion of the encoder block from the associated dictionary and re-indexing the remaining encoder blocks. The HuggingFace [7] package allows for easy adaptation to the configuration where a parameter determines the number of encoder blocks to be included. Moreover, through this adaptation, we can easily retain the remaining possible changes that are indicated in our model dictionary which represents the new model.

We can probabilistically change existing modules of *Multi-head self-attention* or *FNNs* by switching existing ones with newly generated ones. For this we randomly create one of

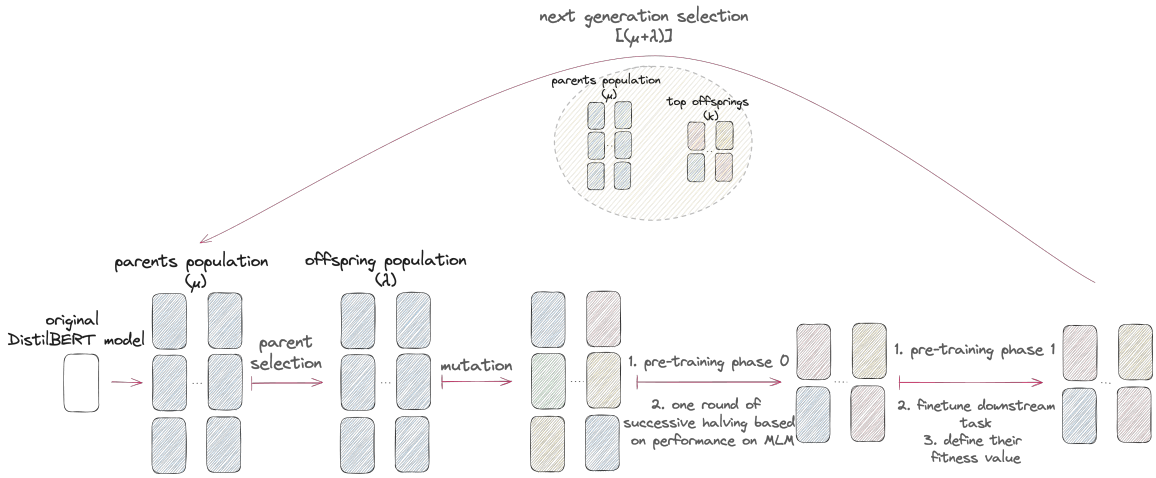


Figure 6: The NAS pipeline outlines the operational steps for the proposed exploration method, based on genetic algorithms. The original DistilBERT model undergoes mutations to create the initial parents' population of size μ. Mating selection is then performed by investigating the Pareto front to choose the best-fitted parents. These selected parents undergo further mutations to generate the stemmed offspring population, which requires pre-training. Candidate models are pre-trained, and the top-performing half is chosen based on their masked language modeling loss. Subsequently, the selected models undergo additional pre-training and fine-tuning. Finally, we compute their objective scores and choose the next-generation parents through plus-selection.

the previously mentioned blocks based on the hyperparameters explained in Section 4.1. Each module generation is independent of the previous one, featuring random instantiation of hyperparameters drawn from the parameter space. These hyperparameters may deviate from the original values whilst also impacting factors such as the number of stacked linear layers and the associated activation functions within the Feed-Forward Neural Network (FNN). These generated modules, combined with the addition or alteration of the FNN module and adjustments to the Multi-head self-attention, collectively contribute to the diversification of the search space, deviating from the original transformer encoder and DistilBERT architecture.

The probabilities of the occurrence of each of the aforementioned mutations have been empirically set to the following values:

Encoder 0.2 with a corresponding likelihood of either adding or removing the encoder block established as $\frac{2}{3}$ and $\frac{1}{3}$, respectively.

FNN 0.4 with a corresponding likelihood of either adding or altering the existing FNN block established as $\frac{2}{3}$ and $\frac{1}{3}$, respectively.

Multi-head Self-attention 0.4 probability of altering the existing Multi-head Self-attention module.

Further experimentation with these hyperparameters is encouraged, but the associated computational with each experimental run is immense.

The encoder mutation will be applied first, and the encoder list will be traversed in a random order, ensuring an equal probabilistic chance for the mutation of a module across all encoders. This allows for experimentation with dynamic mutation probabilities, even though they have not been currently implemented. To ensure that the newly instantiated modules will be identifiable during the training process, we include a flag in the genotype indicating whether altered modules have experienced training yet or not.

4.2.2 Population Initialization

The first step towards running a genetic algorithm is the initialization of the parent population. It was essential to ensure that the generated population arising from the original design of DistilBERT, possessed a diverse set of alterations ensuring that the generated models are deviating from each other and the baseline. To achieve this, the mutation function was utilized with the provision of more aggressive mutation probabilities. Moreover, empirical analysis of the individuals generated was applied, ensuring that the generated individuals are diverse. Analysis of the instantiated modules, as well as the size of models generated assured the application of different mutations.

The motivation for seeking a sufficiently varied population is to encourage the exploration of a wider search space. This decision is considered essential based on the fact that the limited resource environment restricts further the exploration phase, thus initializing on a broader spectrum of the search space, whilst exploiting the respective areas with future generations is vital.

4.2.3 Mating Selection

The mating selection process is heavily based on the NSGA-II algorithm, to achieve the creation of the offspring for the next generation. Due to the need to prioritize the objective of boosting the performance of the candidate models on the downstream task of question-answering, adaptations have been made to remove the balance priority that was pre-established.

Similarly to the original approach, we need to first assess the Pareto dominance of the candidates (according to the two objectives) and iteratively categorize them into non-dominated Pareto fronts. To do this, we identify the set of non-dominated solutions, place them in a new front, remove these candidates, and repeat the process until all of the individuals have been split into fronts. The lower the Pareto front index the higher the importance of the individual. Following this, we calculate the crowding distance of the individuals in

each front, but we utilize only one of the two axes. We calculate the distance from neighboring candidates based on the axis representing the model size objective. This promotes diversity in the set of solutions based on their model size, whilst of course, we aim to optimize the performance of the model.

The NSGA-II approach incorporated the concept of elitist selection, where it directly selected individuals in order of Pareto fronts, which were inserted in the next generation. Binary tournament selection was employed, with the selection criterion being the crowding distance. This leads to favoring candidates with higher crowding distance, endorsing higher diversity. Intuitively, it was decided to adapt the selection process and remove the elitist approach. Instead, pure tournament selection was effectively in place, where a random subset of the population was selected (of size k) and the best-fitted individual was selected. This was determined firstly by the Pareto front-ranking, and in the case of a draw the crowding distance was used as the decisive criterion.

The above decision was justified by the fact that elitist selection favors exploitation during the generation of offspring. This is guaranteed by the selection of the best individuals to be used for the generation of offspring. This will likely otherwise lead to faster convergence, but based on the fact that we lack the "crossover" function, we need to encourage exploration with other techniques. In contrast to this, tournament selection intrinsically leads to a more diverse generation of candidates by randomly selecting the individuals who will compete against each other. The decision of the hyperparameter k was set to 3 in an attempt to solidify the enforced diversity. The decision is grounded in literature and small-scale experimentation, indicating that a small value of k leads to more exploration and induced robustness in a noisy environment (such as the one under consideration). The decision for a small value for this hyperparameter is also grounded on the fact that we have small population sizes.

4.2.4 Pre-training Phases

In an attempt to further stabilize and possibly speed up the pre-training process, this was split into two distinct and sequential stages. The generated offspring that would require to be pre-trained retain partially pre-trained weights from the parent which they stem from. Undeniably, this knowledge relies on the original architecture of the ancestor, however, based on the fact that changes incrementally occur per individual, it is safe to assume that these weights can be re-used [22] as the ground basis and treated as existing knowledge.

Taking into account the aforementioned assumptions, it was decided to set up the two phases of pre-training as follows. The first phase will freeze any retained modules, enforcing training to take place exclusively on the newly initiated modules (resulting from the mutation function). This in theory should allow for more stable and efficient training, since by freezing weights the number of parameters to be updated is reduced. This makes training faster whilst also reducing the demand for computational resources. This process

of pre-training will hereafter be referred to as *pre-training phase* 0. This is applied to the complete datasets and a total of four epochs.

Due to the change in architectural design, the need for training and adjusting the whole network is clear. The next phase of pre-training involves the entirety of the model, thus all of the trainable parameters are unfrozen. However, in order to effectively reduce the need for pre-training explored models, the application of selection criteria was enforced. At the end of the *pre-training phase* 0, the trained models undergo a cutoff process, where only the top half of the population will be selected to proceed in the second phase of pre-training. The selection criterion is their respective performance in the task of MLM. More specifically, the models with the lowest evaluation loss at the end of training are selected. This is achieved through the investigation of recorded losses and associated learning curves.

The selected models proceed to enter the second phase of pre-training, so-called *pre-training phase* 1. As previously stated, these are trained for all the possible trainable parameters. This is done to calibrate the layers of the network (across existing and newly instantiated modules) and further train the retrieved models. This has been set to 8 epochs with usage of the full dataset. Due to the restricted resource environment, a fixed and equal distribution of computational resources was applied. The associated risk of partial pre-training is acknowledged, but in efforts to counter this, the selected hyperparameters of pre-training as well as the number of epochs were empirically determined during experimental training of a variety of architectures (created through our mutation function).

4.2.5 Finetuning on the downstream task

To retrieve the performance on the downstream task of question-answering, we finetune the selected candidate models for 3 epochs using the SQUAD dataset. Next, we evaluate them using the test set and record their performance on the two typically benchmarked metrics for this dataset, $F1$ and **E**xact **M**atch (EM). Since both scores are on the same scale and present equal importance in the benchmarking of this dataset, we then proceed to average them and use their average as the performance objective metric of our candidates. Each model is now associated with performance and size objective scores, which are used in the following step, the next-generation selection.

4.2.6 Next Generation Selection

For the process of selecting the next generation of parents, two techniques that could have been followed were plus-selection and comma-selection. The comma selection, also represented as (μ, λ), allows selection for the next generation of parents exclusively from the offspring population of the current generation. This selection approach will aid the convergence to a global optimum and maintain diversity in the population from one generation to the next because it completely replaces the current population with offspring. However, in

the case of low parent and offspring population sizes, μ and λ respectively, this can also result in the loss of good genetic material, meaning that a bad generation of offspring can result in losing all of the previously attained progress.

In contrast to this, plus-selection also indicated as $(\mu + \lambda)$, provides the capability of selecting the next generation of parents from the current generation's parents and offspring. This provides a form of elitism since it retains individuals potentially from the previous generation of parents thus preventing loss of good genetic material, ensuring that the best-performing candidates have a higher likelihood of surviving from one generation to the next. This was even more vital in our case, due to the small population sizes, essentially running the risk of generating a bad batch of offspring and losing the progress.

The next generation of each population is selected in a deterministic way, in which the best μ individuals are selected out of the μ parents and λ offspring $(\mu + \lambda)$. This is done by selecting the top individuals from the pool of parents and offspring, based on the Pareto rank. This allows for the best candidates to be selected for the next generation of parents, whilst also taking advantage of the benefits accompanying the NSGA-II approach (see Section 2.1.1). Essentially the Pareto front is created for the pool of individuals, we calculate the Pareto rank and crowding distances, and finally, proceed to select the top individuals the same way that NSGA-II implemented their next-generation selection phase.

5 Experiments

The problem statement and research question have been outlined in Section 1. To align with the scope of this project, the experimental setup has been designed to address the following key questions through its execution:

Pareto front investigation Is there sufficient exploration of candidate models based on the conflicting objective scores (as determined through Pareto front analysis)?

Multi-fidelity optimization Is multi-fidelity optimization on an epoch level effective on language models?

Cutoff criterion Is the pre-training loss an effective cutoff criterion and a good performance indicator for the downstream task?

Due to the low number of generations, it is essential to assess the performance trend across generations, which would in turn provide insight into the potential success of this research project and future research routes.

5.1 Baselines

Our baseline model is the original pre-trained model of DistilBERT, retrieved from the HuggingFace platform. Our baseline model is the original pre-trained DistilBERT model from HuggingFace with an EM score of 62.0 and $F1$ of 75.9, thus an average of approximately 69.0. This is used as the basis for the creation of stemmed models, resulting from the application of NAS. The training phase of these models follows three distinct phases (see Section 4) and the trained and finetuned models are compared to the original DistilBERT model after it has been finetuned on the task of question-answering.

To ensure a fair comparison between the generated models and the baseline, a standardized training procedure was followed. For the finetuning task, the performance of the models is compared on the metrics of F1 score [39] and EM score, which are the typical metrics for benchmarking on *SQUAD*. The models are also compared based on their respective model sizes. Intermediate and final model evaluation is also based on the learning curves of the loss retrieved from the pre-training phase on the task of MLM.

5.2 Datasets

The DistilBERT model as well as the subnetworks generated, stemming from the original architecture, require two stages of training. The first stage was the (partial) pre-training of the models on the task of masked language modeling. To clarify, what is meant by partial pre-training, is the fact that models undergo the normal pre-training procedure, but the weights of the pre-trained parent model are loaded and the mutations will only initialize weights for the affected layers. Layers and modules that have not been altered through the mutation phase retain their weights [22]. The pre-training phase will aid the model in achieving a deeper understanding of language semantics and syntax. The associated trained weights provide the foundation for linguistic capabilities, which enable the model to learn a downstream task, such as question-answering; thus finetuning it effectively in the second stage of training. These two previously mentioned tasks rely on the appropriate selection of datasets to achieve the respective goals. The chosen datasets are explored in more detail in the following section.

5.2.1 Datasets for Masked Language Modeling

Our models followed training under the task of MLM to achieve the proper understanding of language. This means that some of the tokens are probabilistically masked, and the model is assigned the task of predicting the hidden token. Two datasets were used to achieve this, namely the *English Wikipedia* dataset [14] and the *BookCorpus* dataset [13].

The *English Wikipedia* dataset consists of a large collection of textual content extracted from the articles existing in the English version of Wikipedia. Only text passages are

extracted, whereas lists, tables, and headers are ignored. As per the justification in the original paper of BERT [3], it was crucial to consider a document-level, instead of a shuffled sentence-level one (such as *One Billion Word* benchmark). This was done to retrieve contiguous sentences and therefore aid the language understanding capabilities.

The *BookCorpus* Dataset was also used, which provided the textual content retrieved from a considerable number of books across many genres, and surrounding themes. The different styles of writing and structures provided a complementary context in terms of linguistics whilst also typically providing longer forms of writing.

The combination of the two datasets provides a comprehensive and extensive sample of textual content, encapsulating linguistic patterns, language syntax, semantics, and contextual relationships across sentences. This can act as the baseline for further enhancing the captured knowledge of the model and sufficiently provide the building blocks for moving into more complex downstream NLP tasks. Another crucial factor for the decision of the previously mentioned datasets was the alignment with the training procedure of the baseline, in an attempt to provide a fair and meaningful comparison of achieved performances across models.

The pre-processing steps for the task of MLM involve the aforementioned datasets, *English Wikipedia* and *BookCorpus*, therefore the first step was retrieving and concatenating them. These are purely textual datasets, with the raw format of the text. Pre-processing and filtering steps can typically influence the performance of the trained model, either positively or negatively. Since the focal point of our research is the application of NAS, we aimed to mitigate any potential effects that can be introduced by third-party factors, thus it was decided to not include pre-processing steps for filtering the dataset.

A standardized pre-processing step in terms of tokenization was followed. The textual information was converted to tokens. A maximum length of 512 tokens was set and sequences longer than that were truncated and split into chunks. The default tokenizer (from HuggingFace) parameters for padding and application of stride were used. The tokenized dataset was also randomly shuffled at the end of its processing.

5.2.2 Datasets for Question-Answering

The **S**tandford **Qu**estion **A**nswering **D**ataset (SQUAD) [15], is well known and widely used dataset in the field of NLP. It consists of a wide and diverse collection of Wikipedia articles, providing the necessary means to assess the ability of a language model in comprehension of context and extraction of accurate answers. Each instance of the dataset presents a passage from the Wikipedia article (with the title of said article). This passage also referred to as *context* is accompanied by a set of questions that should be answered from said passage.

For our purposes, we used *SQUADv1.1* which consists of questions that have at least one answer provided. The goal would be to provide questions spanning in complexity and topics, for the finetuned models to be able to train on extracting the correct answer to a

question (assuming that the appropriate context has been provided). To offer a concise and clear depiction of the dataset's structure, the following representation is provided:

- **Context:** Portion of text from a Wikipedia article, where the answer to the question can be extracted from
- **Question Answer Sets:**
  - **Question:** The question in text format
  - **Answers:** A set containing the possible answers (minimum size 1)
  - **Answer Positions:** A list of indices, indicating which character position, in the context, the answer starts at
  - **Question ID:** A unique identifier to the question
- **Title:** The Wikipedia article's title
- **Dataset role:** which dataset portion it belongs to: [validation or training]

In regards to the *SQUAD v1.1* dataset, there was no filtering step for discarding any of the records. The provided dataset from the HuggingFace platform [40] provides the train and validation set. After loading the two datasets, the train set was split to create the test set, with the train set retaining 80% and the test set 20%. Next, the necessary steps were taken for tokenization, truncation, padding, and application of stride. The first step was tokenizing the long sequences of textual information, which provided the *context*, *question*, and *answer* pairs. The text was split into smaller units, known as tokens, consisting of typically words or subwords. We defined a maximum sentence length of 384 [7], to be able to handle longer sequences of text, by truncating them. Stride is used to define the number of overlapping tokens between two successive chunks of tokens, where the truncation has been applied.

Additionally, special characters are used, $[CLS]$ and $[SEP]$, to separate the context and question in the tokenized data. This can be visualized as follows:

$$[CLS] \text{ question } [SEP] \text{ context } [SEP]$$

Information related to the starting and ending positions of the answers has also been used, to create the proper mapping. The start indices and length of answers from the original dataset will be used to index the respective start and end tokens encapsulating the answer. The model will thus be tasked with predicting one start and end logit per token. These predictions will be used to determine which tokens encapsulate the answer, within the provided context. An example of such tokenization [3] is depicted in Figure 7.

[3] Based on the HuggingFace NLP course https://huggingface.co/learn/nlp-course/chapter7/7?fw=tf

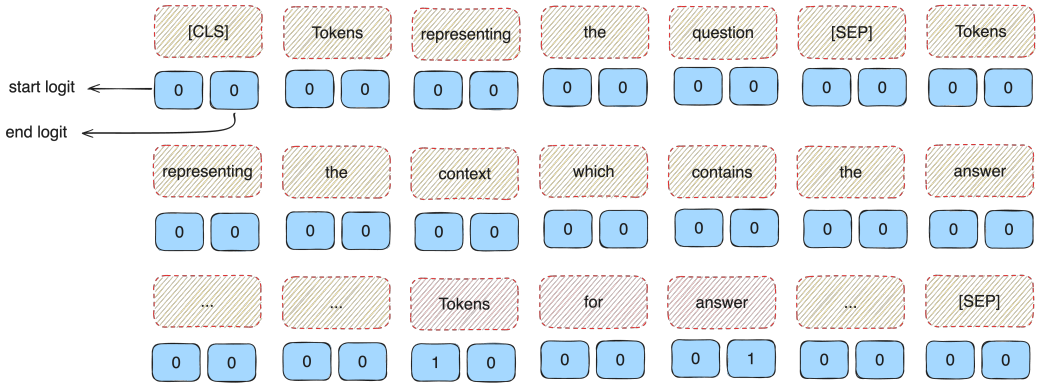


Figure 7: Depiction of the tokenized version of SQUAD for a context, question pair. The yellow rectangles depict the tokens representing the words of the respective part of either the question or the context. The pair of logits below them represent the start and end logits which are to be predicted for the individual tokens. The answer tokens are highlighted with red rectangles, whilst the start and end tokens are also represented through the positive logits in respective positions.

5.3 Experimental setup

The aim of the experimental setup is the validation of the objectives of the thesis, expanded in Section 1, with consideration of the computational cost associated with them. The provided hardware (by the SURF Cooperative) specifications can be seen in Table 1.

Due to the extreme associated cost of pre-training a language model as well as the restricted resources available (in terms of computation and time), it was rendered infeasible to repeat experimental runs, and all of the provided results and trained models are based on a single execution of the pipeline for three generations. Presented results, in Section 6, are aimed towards answering the research questions and provide indications that would foster further research and experimental runs that would solidify conclusions towards decisions associated with the proposed methodology.

The empirical model validation is grounded on the basis of standardized methods of training and datasets, with the usage of pre-determined metrics for comparison, such as the F1 and EM scores on the task of question answering. Furthermore, a comparison of intermediate models' performance is achieved through a comparison of the learning curves and achieved loss on the MLM task, on the point of reference during the training. The restrictive and finite budget led to the decision to allow models to be trained until a predetermined number of epochs (rather than until convergence). In this way, we ensure equitable assessment of the performance of the model and equal distribution of budget amongst them.

One of the key setup hyperparameters involved in any genetic algorithm is the definition of the parents and offspring population. These were set intuitively and through small-scale

Lenovo Node Type	CPU SKU	CPU cores per Node	Accelerator(s)	DIMMs	Total memory per Node (per core)
ThinkSystem SD650-N v2	Intel Xeon Platinum 8360Y (2x) 36 Cores/Socket 2.4 GHz (Speed Select SKU) 250W	72	NVIDIA A100 (4x) 40 GiB HMB2 memory	16 x 32 GiB 3200 MHz, DDR4	512GiB 160GiB HMB2 (7.111 GiB)

Table 1: Hardware specification of a single node, available on the Snellius cluster, provided by SURF Cooperative.

experiments and simulation runs. Due to the high expense of pre-training models, it was evident that both populations should naturally be low in size. The parents' population size (μ) was set to a value of 8, whilst the offspring population (λ) was set to 12. Additionally, it should be noted that due to the usage of Pareto fronts, these values had to be high enough to formulate ideal unambiguous Pareto sets.

The experimental setup is split into several sections, following the pipeline of our methodology. Descriptions and hardware utilization for each section can be seen in Table 2. This provides in detail node utilization of each step and details for the respective process. Each of the processes depicted in the table (aside from the population initialization) is repeated in full per generation of our pipeline. Breaking down the execution into the outlined stages within the pipeline provided finer control over the process. Segmentation and independent execution of the stages enabled us to assess intermediate results, allowing flexibility towards accommodating potential errors. This in turn minimizes the risk of invalidating the entire run, which is especially crucial in resource-intensive executions within a resource-constrained environment such as this one.

Process Name	Node Utilization	Description
Initialize Population	1/4	The mutation function was used - with higher probabilities of each individual mutation taking place - in order to create diverse set of individuals stemming from the original DistilBERT architecture
Offspring Generation	1/4	Parents are selected based on the custom NSGA-II mating selection approach and pairs of parents are created until a population of size λ (offspring size) is created. These are then mutated in order to create the offspring population
Pre-training phase 0	2/4	During this process of pre-training the individuals, only the newly initialized modules in the models retain unfrozen weights and undergo training for 4 epochs on the task of MLM.
Models Cutoff	1/4	Investigate the learning curves of the *pre-training phase 0*, and select half of the models with the smallest loss which should continue onto *pre-training phase 1*.
Pre-training phase 1	2/4	The selected models have all of their weights unfrozen and continue training for 8 epochs on the task of MLM.
Finetune	2/4	The selected models undergo finetuning on the task of question answering, consisting of 3 epochs.
Evaluate	1/4	The selected models are evaluated on the task of question answering and recorded metrics of F1 and EM are saved.
Next Generation Selection	1/4	Using plus-selection process, the best candidates out of the population are selected to form the next generation parents population of size μ.

Table 2: Outline of the experimental steps which are repeated throughout our experimental execution. The first column presents the name of the process, followed on the next column by the node utilization used to run the respective step. Detailed description of individual nodes can be seen in Table 1. The third and last column provides a description of the process with associated details if applicable.

6 Results

This section provides the collected data from the pipeline execution and supplementary experiments. The experimental analysis is presented in three distinct key areas of investigation, namely *Pareto front investigation*, *Multi-fidelity optimization* and *Cutoff criterion*. These will be expanded upon in the subsections to follow.

6.1 Pareto front investigation

The generation of Pareto fronts involves an iterative and repeating process. A candidate solution dominates another if it is at least as good as any other solution in the pool in all objectives and strictly better in at least one objective. The identification of non-dominated solutions categorizes them in the current Pareto front ranking. These solutions are then excluded from the set, and the process is repeated until no candidate solutions remain. Higher-numbered Pareto fronts (see Figure 8) are less significant, as they are dominated by candidates from lower-numbered fronts.

Figure 8 illustrates the plotted Pareto fronts across the three generations. The type of population is represented by different shapes, with offspring candidates denoted by squares and the parent population by circles. Coloration is employed to distinguish the Pareto front rankings for each candidate. Analysis of the progress across generations indicates little improvement, as evident by the predominant selection of candidates for the next generation from the set of parents. Additionally, it should be stated that the original model of DistilBERT has not been outperformed, with our best model achieving comparable performance. However, the diverse set of data points in the Pareto fronts indicates effective exploration, which is crucial for capturing the different trade-offs between the typically conflicting objectives that are under investigation. This, coupled with the earlier statement, reinforces the need for plus-selection for selecting the next generation of parents. Lack of inclusion of the parents in the candidate pool would amount to the loss of good genetic material and loss of progress across generations.

The lack of improvement may stem from the relatively low number of explored configurations both within generations and across the whole execution of the pipeline. Another factor hindering the improvement could be the predetermined allocation of budget, which is uniform across all candidates and defined by the number of epochs of pre-training. Moreover, the absence of convergence as depicted in Figure 9 may affect the proper exploration of potentially promising candidates, since the possibility of their selection for future generations is limited.

Our limited generation of models is emphasized when considering the findings of the authors of [37], where exploration was extended to more than 750 candidates; where we were able to only explore dozens of candidates within our compute budget. This highlights the necessity of seeking strategies to expand population pool sizes. Such a strategy could be

the utilization of learning curve [23] analysis, which would allow for effective allocation of resources to more promising candidates, thus ensuring full convergence of a smaller portion of models.

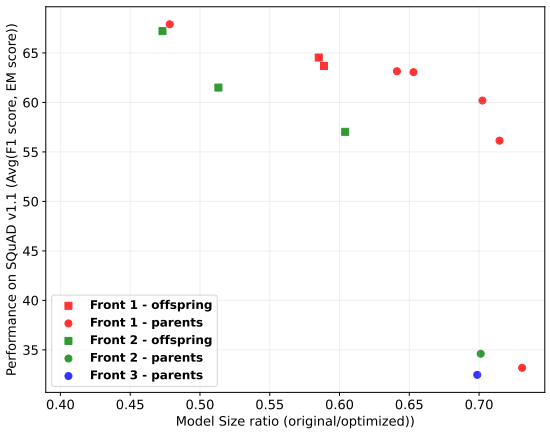
6.2 Multi-fidelity optimization

Figure 9 presents the offspring population of the third and last generation. In order to assess the effectiveness of our decision to employ model training cutoff, we proceed to fully train all of the models of the last generation to retrieve the final performance of all of the offspring candidates. The offspring population underwent training through both phases of pre-training as well as finetuning on the downstream task. Two models that experienced exploding gradients have been excluded from the figure. The examination of the MLM loss learning curves provides insights into the effectiveness of the cutoff criterion between the two training phases and raises the question of whether a multi-fidelity optimization approach, particularly on an epoch level, would be a valuable strategy for further exploration. The analysis of the learning curves reveals that the majority of the selected models (depicted by solid lines) maintain their dominance even after the cutoff criterion at around $18\,000$ steps. Notably, candidate *model* 0 which was the worst performer at our cutoff point managed to outperform a significant number of models, including two of the initially selected models that performed less effectively. However, it is important to highlight that despite this improvement, *model* 0 does not manage to achieve performance comparable to the best-performing models. This data suggests the potential merit of conducting a learning curve analysis on LLMs. This can potentially allow the exploration of more models, leveraging selection mechanisms such as hyperband [41], thereby enhancing the exploration of the search space.

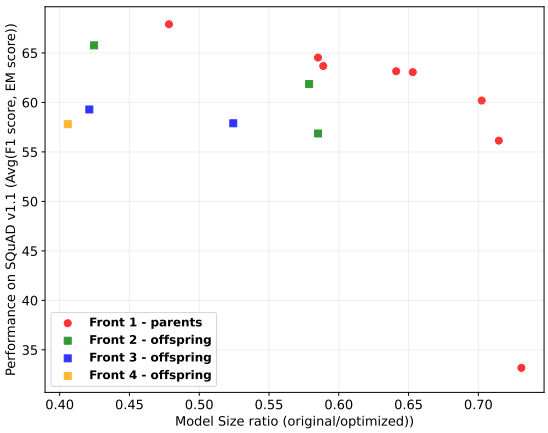
6.3 Cutoff criterion

To conduct a more thorough evaluation of the effectiveness of our cutoff criterion, we investigate the correlation between the MLM loss during pre-training and the performance on the downstream task. We record the pre-training loss on the task of MLM at the cutoff point (between the two phases of pre-training) and the final pre-training loss of our offspring models in the last generation. We then plot them against the performance on the downstream task of question answering. The rationale for monitoring and recording the pre-training loss both during and after training, while also examining its correlation with downstream task performance, stems from the lack of insights into the ultimate model performance throughout the training process. Observing Figure 10 provides empirical indications that the loss during pre-training can be an effective cutoff criterion for model selection. The presented correlation for the final pre-training loss and the cut-off point pre-training loss in regard to the downstream task performance score is -0.771 and -0.789,

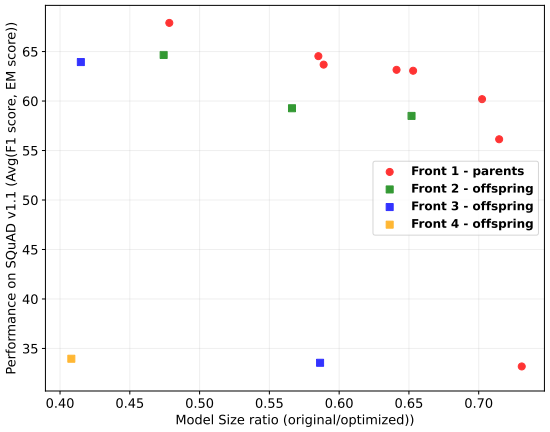
respectively. These results demonstrate a strong correlation, affirming the relevance of the pre-training metric score as an indicator of our downstream task performance. This is crucial because, during the training phase of the model, we lack knowledge of its future performance on the downstream task, which should be optimized. Establishing an effective stopping criterion is necessary for optimizing the allocation of available resources to the more promising candidates.



(a) Generation 1 - Pareto fronts



(b) Generation 2 - Pareto fronts



(c) Generation 3 - Pareto fronts

Figure 8: Pareto fronts were formulated for the primary objective of the downstream task of question answering (average of F1 and Exact Match scores), and the secondary objective of the ratio of the original model size to the candidate model size. Higher numbered Pareto front ranking indicates models of less significance since they are dominated by the models in lower numbered Pareto fronts. Pareto front rankings are indicated by coloration choices specified in the legends, while population type is depicted by the shape of the data point. Circles represent candidates in the parent population, while squares depict the parents.

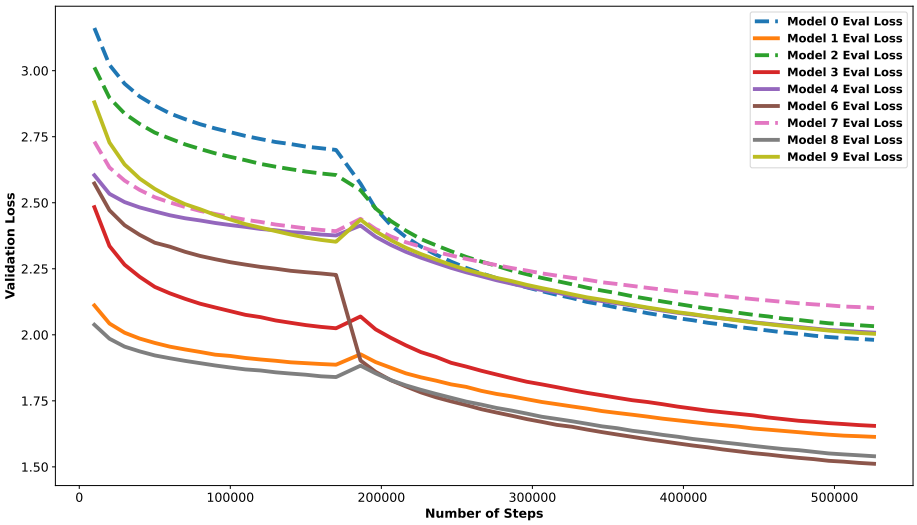


Figure 9: Masked language modeling task loss captured during the two phases of pre-training for the offspring population of generation 3. The solid lines represent the selected models that have moved to *pre-training phase 1*, whilst the dotted lines represent the models that were otherwise cut off.

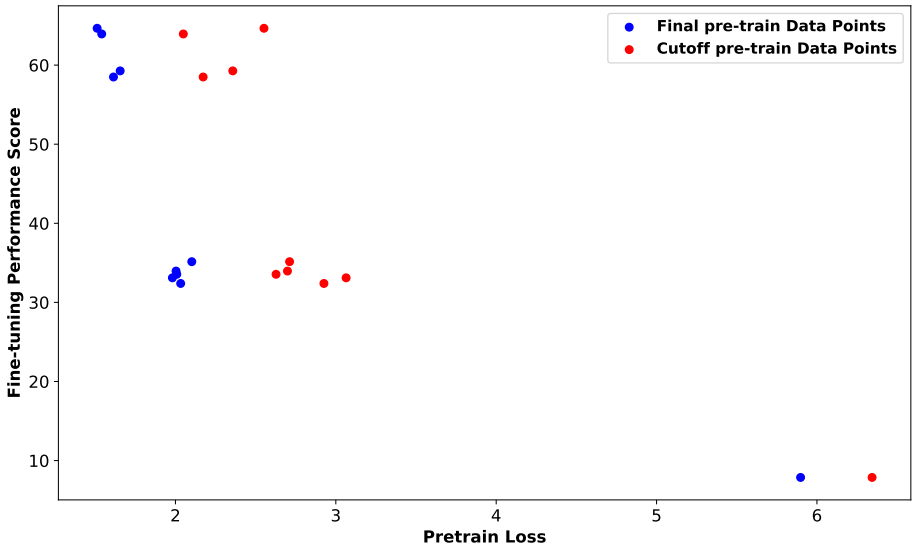


Figure 10: Presenting the correlation between the retrieved loss during the pre-training phase, and the downstream task performance score (post pre-training). The blue dots represent the final pre-training masked language modeling loss, whilst the red dots depict this loss at the cutoff point.

7 Limitations and Future work

A clear drawback of this current project is the lack of extensive execution of the program. The restricted number of generations of EA, stemming from the limitation of resource and time acquisition, is not sufficient to reach convergence of the search procedure. We highly encourage the continuation of the execution of the pipeline to reach convergence or possibly better-resulting models. However, before committing additional resources, we should first explore further avenues for enhancing the training efficiency of the pipeline, particularly focusing on the pre-training component. While there is an abundance of research dedicated to parameter-efficient fine-tuning techniques [42], there are limited insights on how to enhance the efficiency of the pre-training phase. It would be worthwhile to investigate further methodologies that leverage the retained weights of the unaffected layers in the model. This can potentially enable us to expedite the pre-training process, thus allowing a more extensive exploration of the search space without increasing the resource expenditure.

The current implementation of the pipeline explores the optimization of two objective metrics, i.e. the performance on the downstream task and the model size. However, there are a multitude of performance indicators, and their careful assessment should highlight their respective importance according to our task at hand. The Pareto front optimization for a multi-objective setting allows for the potential addition of more objectives such as inference time, energy consumption, memory usage, and so on. The exploration of more downstream tasks will also have a positive impact and indicate the generalizability of candidate models but a selection of a proxy task should be in place to avoid increasing the dimensionality of our objective space unnecessarily.

Our experiments indicate that utilizing the pre-training loss as a termination criterion during the training process is a viable path. The strong correlation with the final performance on the downstream task, which is our objective of interest, highlights its potential to act as a performance indicator during the training procedure (see Figure 10). A pivotal decision for our research was to utilize learning curves for the early cutting of models which depicted less potential. This was necessary in order to minimize the computational for the pre-training phase of the models, and our insights suggest that learning curve analysis on language models is worth investigating. Further research on multi-fidelity optimization techniques (on an epoch level), such as hyperband [41], is highly recommended since it holds the potential for more efficient resource allocation. This can result in the generation of more models per generation, thus expanding the explored search space.

Due to the high associated cost of the pre-training phase, another training technique that would be worth investigating would be the application of knowledge distillation [28]. The usage of a pre-trained (typically bigger) model, as the teacher, might aid the student model (our retrieved models) to achieve possibly a higher performance score whilst potentially utilizing fewer resources. Therefore investigation and comparison of self-training and knowledge distillation, for both pre-training and finetuning on the downstream task would

be beneficial.

Another point worth exploring in our proposed pipeline is the possible addition of a crossover function. Typically, in an evolutionary algorithm, we induce exploration through the crossover of the selected parents of the pair, whilst the mutation function provides exploitation capabilities. Both are very important for the efficient identification of a good candidate, however, the complexity of NNs renders the implementation of the crossover function more difficult. In this work, the balance between exploration and exploitation is provided through the mutation function in combination with the selection mechanism which favors diversity but the effective addition of the crossover would be highly beneficial. Undoubtedly, many considerations need to be in place, that would allow the application of crossover without further increasing the need for pre-training.

Our research focused exclusively on the extraction of the answer, assuming that the model has been provided the correct context. However, it cannot be guaranteed that knowledge is neither universal nor static across time. Investigation of the application of NAS on a model responsible for the retrieval part of the context would be interesting research. This can potentially lead to the creation of a generative model augmented by the retrieval of information, both of which have been optimized through NAS. This could also highlight the respective importance of the extractive, generative, and retrieval parts which are applied towards answering a given question.

Further research could explore alternative options for the NAS methodology. An area that would be beneficial to investigate is the usage of a dynamic mutation probability. In theory, this should provide the possibility of an adapting mutation rate through the progression of the execution. This will ultimately allow the algorithm to respond to changes in the optimization landscape or the population's characteristics. This provides a seamless switch between exploration and exploitation, with the prospect of leading to better convergence and solution quality. The associated limitation is the increased associated complexity introduced by the requirement of tuning and the necessary consideration for the respective adaptations.

Finally, we believe that subsequent research efforts should prioritize the extension of these findings to encompass larger language models and a broader array of downstream tasks, thereby enhancing the generalizability of the insights.

8 Conclusion

In this work, we have performed a neural architecture search towards DistilBERT, ranging over $6\,500$ GPU-hours on NVIDIA A100 GPUs with 40GBs of VRAM. We propose a novel hierarchical search space to adapt the backbone of the transformer encoder. This is accompanied by the integration of a segmented NSGA-II-based pipeline, serving as our exploration strategy. In contrast to traditional holistic pipelines, this segmentation facilitates

individualized analysis and improvement of its components, thus enhancing the optimization process. We indicate components of the pipeline that are important to undergo further investigation, where dedicated efforts towards their optimization will potentially improve the retrieved results and resource expenditure.

Our findings highlight a compelling correlation between the performance metric in the pre-training phase and the downstream task. Additionally, there are indications of the applicability of learning curve analysis for language models, and consequently potential effectiveness of a multi-fidelity optimization approach on an epoch level. We emphasize the importance of conducting further empirical investigations to strengthen these observations. Analysis of multiple downstream tasks will aid the generalization of our insights, whilst investigation of larger language models is also paramount towards ensuring potential scaling of conclusions to LLMs. Based on the above, we conclude that the application of NAS, in a resource-constrained environment, can possibly aid the identification of better transformer-based language models, but it is deemed necessary to allocate resource and research efforts toward optimizing parts of the pipeline prior to further execution (especially on large language models).

Acknowledgements This work has been (partly) financed by the Dutch Research Council (NWO) and has used the Dutch national e-infrastructure, with the support of the SURF Cooperative, using grant no. EINF-5916.

References

[1] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. A survey of large language models. *Computing Research Repository, CoRR*, abs/2303.18223, 2023.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the International Conference on Advances in Neural Information Processing Systems, NeurIPS*, pages 5998–6008, 2017.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL*, pages 4171–4186, 2019.

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

[5] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *Computing Research Repository, CoRR*, abs/2302.13971, 2023.

[6] Andreas Köpf, Yannic Kilcher, Dimitri von Rütte, Sotiris Anagnostidis, Zhi-Rui Tam, Keith Stevens, Abdullah Barhoum, Nguyen Minh Duc, Oliver Stanley, Richárd Nagyfi, Shahul ES, Sameer Suri, David Glushkov, Arnav Dantuluri, Andrew Maguire, Christoph Schuhmann, Huu Nguyen, and Alexander Mattick. OpenAssistant conversations – Democratizing large language model alignment. *Computing Research Repository, CoRR*, abs/2304.07327, 2023.

[7] 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. Transformers: State-of-the-art natural language processing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing: System Demonstrations, EMNLP*, pages 38–45, 2020.

[8] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated Machine Learning: Methods, Systems, Challenges*. Springer International Publishing, 2019.

[9] Colin White, Mahmoud Safari, Rhea Sukthanker, Binxin Ru, Thomas Elsken, Arber Zela, Debadeepta Dey, and Frank Hutter. Neural architecture search: Insights from 1000 papers. *Computing Research Repository, CoRR*, abs/2301.08727, 2023.

[10] Alexander Tornede, Difan Deng, Theresa Eimer, Joseph Giovanelli, Aditya Mohan, Tim Ruhkopf, Sarah Segel, Daphne Theodorakopoulos, Tanja Tornede, Henning Wachsmuth, and Marius Lindauer. AutoML in the age of large language models: Current challenges, future opportunities and risks. *Computing Research Repository, CoRR*, abs/2306.08107, 2023.

[11] Thomas Bäck. *Evolutionary algorithms in theory and practice - evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, 1996.

[12] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[13] Yukun Zhu, Ryan Kiros, Richard S. Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE International Conference on Computer Vision, ICCV*, pages 19–27, 2015.

[14] Yumo Xu and Mirella Lapata. Weakly supervised domain detection. *Transactions of the Association for Computational Linguistics*, 7:581–596, 2019.

[15] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing, EMNLP*, pages 2383–2392, 2016.

[16] Abien Fred Agarap. Deep learning using rectified linear units (ReLU). *Computing Research Repository, CoRR*, abs/1803.08375, 2018.

[17] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Computing Research Repository, CoRR*, abs/1808.05377, 2018.

[18] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *Proceedings of the International Conference on Learning Representations, ICLR*, 2017.

[19] Zheng Jian, Han Wenran, Zhang Ying, and Ji Shufan. EENAS: An efficient evolutionary algorithm for neural architecture search. In *Proceedings of The Asian Conference on Machine Learning, PMLR*, volume 189, pages 1261–1276, 2023.

[20] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *Proceedings of the International Conference on Learning Representations, ICLR*, 2019.

[21] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: stochastic neural architecture search. In *International Conference on Learning Representations, ICLR*, 2019.

[22] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka I. Leon-Suematsu, Jie Tan, Quoc V. Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *Proceedings of the International Conference on Machine Learning, ICML*, volume 70, pages 2902–2911, 2017.

[23] Felix Mohr and Jan N. van Rijn. Learning curves for decision making in supervised machine learning - A survey. *Computing Research Repository, CoRR*, abs/2201.12150, 2022.

[24] Robin M. Schmidt. Recurrent neural networks (RNNs): A gentle introduction and overview. *Computing Research Repository, CoRR*, abs/1912.05911, 2019.

[25] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *Computing Research Repository, CoRR*, abs/1511.08458, 2015.

[26] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997.

[27] 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. *Journal of Machine Learning Research, (JMLR)*, 21:140:1–140:67, 2020.

[28] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *Computing Research Repository, CoRR*, abs/1503.02531, 2015.

[29] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *Computing Research Repository, CoRR*, abs/1910.01108, 2019.

[30] Wei Zhu, Xiaoling Wang, Yuan Ni, and Guotong Xie. Autotrans: Automating transformer design via reinforced architecture search. In *Proceeding of the International Conference on Natural Language Processing and Chinese Computing, (NLPCC)*, volume 13028, pages 169–182, 2021.

[31] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *Proceedings of the International Conference on Machine Learning, ICML*, 2018.

[32] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

[33] David R. So, Wojciech Manke, Hanxiao Liu, Zihang Dai, Noam Shazeer, and Quoc V. Le. Searching for efficient transformers for language modeling. In *Proceedings of the International Conference on Advances in Neural Information Processing Systems, NeurIPS*, pages 6010–6022, 2021.

[34] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Gregory S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian J. Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Józefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Gordon Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul A. Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda B. Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *Computing Research Repository, CoRR*, abs/1603.04467, 2016.

[35] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Proceedings of the International Conference on Advances in Neural Information Processing Systems, NeurIPS*, pages 8024–8035, 2019.

[36] Yichun Yin, Cheng Chen, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. AutoTinyBERT: Automatic hyper-parameter optimization for efficient pre-trained language models. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics and the International Joint Conference on Natural Language Processing, ACL/IJCNLP*, pages 5146–5157, 2021.

[37] Jiahui Gao, Hang Xu, Han Shi, Xiaozhe Ren, Philip L. H. Yu, Xiaodan Liang, Xin Jiang, and Zhenguo Li. AutoBERT-Zero: Evolving BERT backbone from scratch. *Proceedings of the AAAI Conference on Artificial Intelligence, AAAI*, pages 10663–10671, 2022.

[38] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (GELUs). *Computing Research Repository, CoRR*, abs/1606.08415, 2016.

[39] Marina Sokolova, Nathalie Japkowicz, and Stan Szpakowicz. Beyond accuracy, f-score and roc: A family of discriminant measures for performance evaluation. In *Proceedings of the Australian Joint Conference on Artificial Intelligence, AJCAI*, 2006.

[40] Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, Joe Davison, Mario Sasko, Gunjan Chhablani, Bhavitvya Malik, Simon Brandeis, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, Angelina McMillan-Major, Philipp Schmid, Sylvain Gugger, Clément Delangue, Théo Matussière, Lysandre Debut, Stas Bekman, Pierric Cistac, Thibault Goehringer, Victor Mustar, François Lagunas, Alexander M. Rush, and Thomas Wolf. Datasets: A community library for natural language processing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing: System Demonstrations, EMNLP*, pages 175–184, 2021.

[41] Lisha Li, Kevin G. Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research, (JMLR)*, 18:185:1–185:52, 2017.

[42] Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, Jing Yi, Weilin Zhao, Xiaozhi Wang, Zhiyuan Liu, Hai-Tao Zheng, Jianfei Chen, Yang Liu, Jie Tang, Juanzi Li, and Maosong Sun. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence*, 5:220–235, 2023.