FlexiBERT: an Exploratory Study of the Transformer Design Space*

Shikhar Tuli[1†] **Bhishma Dedhia**[1] **Xiaorun Wu**[2]
[1]Department of Electrical and Computer Engineering, Princeton University, United States
[2]Department of Mathematics, Princeton University, United States
{stuli,bdedhia,xiaorunw}@princeton.edu

Abstract

Recently, Transformers have gained significant popularity in the industry and academia owing to their tremendous success in natural language processing (NLP) and other machine learning (ML) domains. There is a plethora of architectures in this field, and each model outperforms the other in some task. However, there is often no standard procedure to train such models (e.g. some models are pre-trained on more data than others). A common benchmarking framework is required to compare and contrast the architectural hyperparameter choices of such models to quantitatively show why each model performs the way it does for a certain task. Hence, in this work, we present FlexiBERT—a unified approach for Transformer design space generation, and a tool to benchmark different architectures on popular NLP tasks—exploring and extracting the best set of design choices for each task.

1 Introduction

Self-attention-based Transformer models (Vaswani et al., 2017) have achieved state-of-the-art results in tasks that span across natural language processing (NLP), and recently, even computer vision (Dosovitskiy et al., 2021). Examples of such tasks include machine translation, speech recognition, question answering, sentiment analysis and image recognition. This burgeoning success has been largely driven by large-scale pre-training datasets, increasing computational power and robust training techniques.

Transformer design involves a choice of several hyperparameters, including but not limited to: number of layers l, size of hidden embeddings h, number of attention heads a, size of the hidden layer in the feed-forward network f, and the choice of the similarity metric s. This leads to a design space that scales exponentially with each design decision and thus naturally begs the question that which selection of hyperparameters and what ranges maximally influence performance of a model on a specific task. Investigating this behavior has several challenging aspects.

Evidently, it is computationally infeasible to train every model in the design space. One way to tackle this challenge is by using domain knowledge (*i.e.* defining the constraints for every new model in consideration manually) to craft the architectures one-by-one. But this trial-and-error approach is often too computationally expensive to efficiently explore the design space (Ying et al., 2019).

Intuitively, all we need instead is a performance predictor, that can *reliably* foretell the accuracy for every model in the design space in consideration. Recent works have shown that *directly* obtaining this performance predictor can significantly reduce the training time. Such a surrogate can be trained by querying a few models from the design space and regressing their performance to the remaining space, under some theoretical assumptions (c.f. Section 3.4) of the accuracy profile (Siems et al., 2020; White et al., 2021).

However, learning this surrogate function is not a straightforward task, as each model is a discrete point in the space and the predictor learned may be ill-posed to generalize. Instead, each model can be embedded into a lower dimensional space over which the surrogate learns to predict the performance. A good representation should accurately capture the similarity between the points in the space and should be able to *reliably* generalize the performance predictor. In this work, we represent each Transformer model via a 'computational

*<u>**Confidentiality Notice:**</u> This work has been submitted as part of final project for the course COS 484, Spring 20-21 at Princeton University. Third parties—with the exception of the course instructors and authorized examining persons—may not access this work without the express permission of the authors.

†Corresponding author

graph' using the feed-forward flow of computational 'blocks' representing various operations. Using this representation for every model, we train a `Transformer2vec` embedding, based on the similarity metrics obtained via graph kernels.

Even if a good embedding is obtained, a challenge that remains is in *efficiently* selecting models to train the surrogate function. Intuitively, only those models should be trained that help the most in minimizing the *epistemic* uncertainty in the accuracy prediction. This is exactly an active learning scenario, where we need to choose the next query point that maximizes this uncertainty gain for every new model that is trained (Settles, 2009).

Recent advancements in neural architecture search (NAS) provide various techniques to explore and optimize different models in the deep learning domain, from image recognition to speech recognition and machine translation (Zoph and Le, 2017; Mazzawi et al., 2019). For instance, in the computer-vision domain, there is a plethora of CNN architectures that exploit different design decisions (and some having even new basic operations (Zhang et al., 2018)) to give different performances on different tasks (He et al., 2015; Simonyan and Zisserman, 2015).

Similarly, in the Transformer domain, many architectures have been proposed. Among these, the popular ones include BERT, GPT, XLM, XLNet and many more (Devlin et al., 2018; et al., 2020; Lample and Conneau, 2019; Yang et al., 2019). Just like in CNNs, many variations to these basic architectures have also been proposed to improve performance or efficiency: for example, BART, BORT, ConvBERT, DeBERTa, RoBERTa, etc (Lewis et al., 2019; de Wynter and Perry, 2020; Jiang et al., 2020). However, unlike CNNs (Ying et al., 2019), there isn't any universal framework in NLP to differentiate Transformer architectural hyperparameters. A unified approach is required to characterize and test 'why' each architecture is 'best' in its own way. This calls for a standard benchmarking methodology that unequivocally compares and contrasts different design decisions for each such network, and for every popular NLP task.

In this paper, we present FlexiBERT—a unified approach for Transformer design space generation, and a tool to benchmark different architectures on popular NLP tasks. The rest of the paper is structured as follows. Section 2 presents a brief background on the design space exploration problem along with related works. Section 3 highlights the set of steps and decisions used to formulate the FlexiBERT framework. In Section 4, we present the results from all experiments performed on the design space that was considered. Finally, Section 5 concludes the paper.

2 Background and Related Work

2.1 Transformer Design Space

Vaswani et al. (2017) first proposed the use of Transformers for sequence-to-sequence classification tasks. Transformers compute representations of its input and output without using sequential dependencies like recurrent connections (RNNs) or token-wise convolution operations (CNNs) (Hochreiter and Schmidhuber, 1997; Kim, 2014). Instead, Transformers primarily rely on *self-attention* to up-weight information different distances away in the text.

BERT is one of the most widely used Transformer architectures (Devlin et al., 2018). However, researchers have constantly strived towards improving this model. Liu et al. (2019) proposed a more robust pre-training approach to improve BERT's performance. Other functional improvements have also been proposed, for example the use of auto-regressive methods (Yang et al., 2019) and denoising approaches (Lewis et al., 2019).

In this work, we consider another set of architectural design decisions for BERT—number of encoder layers l, size of hidden embeddings h, number of attention heads a, size of the hidden layer in the feed-forward network f, and the choice of the similarity metric s (accounting functional improvements in the design space forms part of future work). Different BERT pre-trained models (e.g. BERT-Tiny to BERT-Large) use different sets of these hyperparameters. Recent works have tried to explore the most efficient set of design choices in BERT.

Khetan and Karnin (2020) explore such choices, but are only concerned with pruning BERT and do not target optimizing accuracy over different tasks. Further, they only target to get a computationally lighter model with a fixed set of trainable parameters, rather than searching for the highest accuracy model and inspecting its architecture. Moreover, this design is still heavily restricted, using the the same hidden size (h) for all encoder layers. In this work, we propose a novel projection-based approach to make hidden sizes (and thus the di-

mensionality of data-flow) *flexible* across layers—hence the name FlexiBERT.

2.2 Neural Architecture Search

Neural Architecture Search (NAS) is an important machine learning technique which algorithmically searches for new neural-network architectures within a pre-specified design space under a given objective (He et al., 2021).

Although, previously restricted to CNNs (Zoph et al., 2017), NAS has recently seen applications in the Transformer space as well. So et al. (2019) propose the use of standard NAS techniques to search for optimal Transformer architectures. However, their method—Progessive Dynamic Hurdles (PDH) is akin to traditional hyperparameter optimization strategies, like HyperBand (Li et al., 2018), and requires training from scratch for every new model. This work does not use knowledge-transfer, *i.e.* the use of weights from previously trained *neighboring* models to help speed-up the subsequent training process. Knowledge-transfer has been used in recent works, but is only restricted to LSTMs and simple-RNNs (Mazzawi et al., 2019). We propose the use of knowledge-transformer in Transformers for the first time, to the best of our knowledge, by comparing weights with computational graphs of *nearby* models. For more details, please refer to Section 3.5.

Traditionally, NAS has almost always been solved by converting the search problem into a Reinforcement Learning (RL) setting. However, recently, NAS has also seen application of surrogate models for performance prediction in CNNs (Siems et al., 2020). This has resulted in training much lesser number of models to predict the accuracy for the entire design space. These surrogate models could also be used to optimize down-stream tasks like hardware-aware inverse design (selecting the most optimal neural-network, which may not be the one with highest performance, for a given set of hardware constraints) or even co-design (efficiently get an optimal set of hardware and neural-network pairs). Still, such works are mostly restricted to the space of CNNs (Dai et al., 2019). In this paper, we propose a surrogate modeling framework for the Transformer design space.

2.3 Learning Embeddings for Computational Graphs

A popular approach to learning with graph-structured data is to make use of graph kernels—functions which measure the similarity between graphs. Specifically, a graph kernel is a kernel function that computes an inner product on graphs. They allow kernelized learning algorithms such as support vector machines to work directly on graphs, without having to do feature extraction to transform them to fixed-length, real-valued feature vectors.

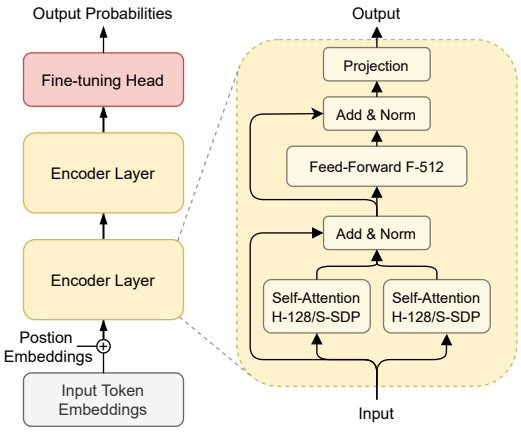

Figure 1: Block-level Computation Graph for BERT-Tiny in FlexiBERT. Projection-layer is same an Identity function since the hidden size of input and output encoder layers is the same.

There are two popularly used graph kernels. First is the Random-Walk (RW) kernel, which conceptually performs random walks on two graphs simultaneously, then counts the number of matching paths that were produced for both graphs (Schölkopf and Warmuth, 2003). However, as the size of the graphs increase, computing this kernel becomes computationally expensive. The second is the Weisfeiler-Lehman Subtree (WL) kernel, which compares tree-like sub-structures of two graphs, which helps distinguish between substructures that the RW kernel may deem identical (Shervashidze et al., 2011). Since the WL kernel has an attractive computational complexity, we use this over the RW kernel. One future direction from this work is to use domain-knowledge to improve the computed graph similarity metric (c.f. Section 3.4 for details).

Figure 1 shows the 'block-level' computational graph for BERT-Tiny. Every operation—including self-attention, the feed-forward neural network, or the 'Add&Norm' layer—can be represented by compute blocks, that are connected in the forward-flow of the network. Using these connection patterns for every possible block permutation, we can generate multiple graphs for the given design space on which we compute the similarity metric.

Once we have similarities computed between every possible graph-pair in the design space, next we learn dense embeddings, the Euclidean distance for which should follow the similarity function. These embeddings would not only be helpful in *effective* visualization of the design space, but also for fast computation of *neighboring* graphs during the active-learning loop. Further, a dense embedding helps us practically train a finite-input surrogate function.

Many works have achieved this using different techniques. Narayanan et al. (2017) train task-specific graph embeddings using a skip-gram model and negative sampling, taking inspiration from `word2vec` (Mikolov et al., 2013). In this work, we take inspiration from `GloVe` instead (Pennington et al., 2014), by applying Multi-Dimension Scaling (MDS) to the similarity matrix (Kruskal, 1964). Hence, using global similarity distances + MDS (+ potentially domain-knowledge), we train the `Transformer2vec` embeddings, that are superior to traditional generalized graph embeddings.

Recently, Cheng et al. (2020) showed that training a WL kernel-guided encoder with a combination of the similarity metric and the reconstruction distance as the loss function, has advantages in scalable and flexible search. In this work, we implement a simpler solution, but with the same goal. Comparing this MDS-based embedding with an encoder-based embedding could easily form an extension to the present work.

2.4 Active Learning

Active learning (AL) is a machine learning framework in which a learning agent can interactively query a simulator to label new data points with the true labels. Evidently, it has the potential to provide an exponential speedup in sample complexity (Hanneke, 2011). Geifman and El-Yaniv (2018) proposed an AL framework to implement NAS efficiently on a CNN design space.

In this work, we implement a simple Gaussian-Process (GP) regression to model the surrogate performance predictor. We employ a *purely-exploratory* uncertainty sampling algorithm to effectively minimize the overall uncertainty of the surrogate function. More concretely, at every step of the AL loop, the next query is chosen based on the Transformer model with the highest *epistemic* uncertainty in accuracy prediction. Modeling *epistemic* with *aleatoric* uncertainty in-tandem, and using mixture-density networks (MDNs) instead of a GP-regression model can be explored as part of future work.

3 Methods

Figure 2 shows a high level overview of the techniques we use—from creating the computational graph for every architecture in the FlexiBERT space, to *actively* training a surrogate function to predict the accuracy for every such model. We next discuss each component of this pipeline in detail.

3.1 Design Space and Model Cards

BERT (Devlin et al., 2018) is a popular architecture based on the encoder described in (Vaswani et al., 2017). It consists of multiple layers that execute multi-headed self-attentions to learn a representation for an input sequence. We consider the key design elements of BERT in our design space. These elements along with their permissible values have been shown in Table 1. Note that we also include a functional design choice by allowing similarity metrics based on either scaled dot-product (SDP) or weighted multiplicative attention (WMA). The motivation for this is that, compared to SDP, WMA has an additional learn-able parameter, which may enhance the expressivity of the model.

Design Element	**Allowed Values**
number of encoder layers (l)	$\{2\ , 4\}$
number of self-attention heads (a^j)	$\{2\ , 4\}$
hidden size (h^j)	$\{128\ , 256\}$
feed-forward dimension (f^j)	$\{512\ , 1024\ , 2048\}$
similarity metric (s^j)	{SDP, WMA}

Table 1: Design space description. Super-script (j) depicts the value for layer-j.

Every model in the design space can therefore be described via a model card which is a dictionary containing the choice of each design-component. For example, BERT-Mini can be denoted by: $\{l$: 4, h: [256, 256, 256, 256], a: [4, 4, 4, 4], f: [1024, 1024, 1024, 1024], s: [SDP, SDP, SDP, SDP]$\}$

Note that the above model has a constant flow of dimensionality throughout the network, *i.e.* data flows as a matrix of dimensions $N \times h$ from one layer to the next, for N tokens in the input. However, we hypothesize that the model would be more expressive should we give it the *flexibility* to decide the dimensionalty of the data at every layer by

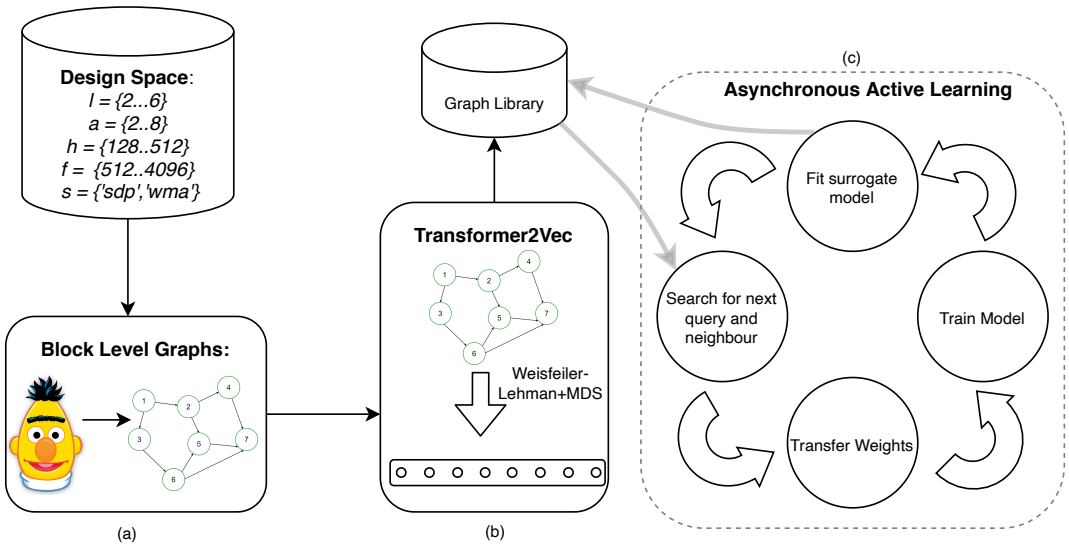

Figure 2: Overview of the FlexiBERT pipeline. (a) Each model in the design space is converted to a graph based on its block-level computational path. (b) Weisfeiler-Lehman kernel is used to compute graph similarity between models in the design space. Multidimensional Scaling (MDS) converts the graph similarity matrix into vector embeddings. (c) Active learning for training a Gaussian-Process based surrogate function. Models are queried based on their performance uncertainty and the weights of their neighbors are used for transfer-learning.

itself. Opposed to vanilla BERT, we allow different configurations across layers in FlexiBERT. We present this next.

3.2 Enhancing Model Flexibility with inter-layer Projections

Traditional Transformers are restricted to a flow of information through a restricted dimensionality thoughout the network. As mentioned before, we allow architectures in the design space to have heterogeneous designs across layers which, in turn, allows different layers to capture information of different dimensions, as it learns more abstract features deeper into the network. To account for this we make the following modifications:

- *Projection layers*: We add an affine projection network between layers having dissimilar hidden sizes to transform the dimensionality of the encoding.
- *Relative positional-encoding*: The original transformer implementation uses an absolute positional encoding at the input and propagates this ahead via residual connections. Since we violate the condition of a constant hidden size across layers, this is not amenable for several models in our design space. Instead we add a relative positional encoding (Shaw et al., 2018) at the beginning of each layer. Such an encoding can entirely replace absolute positional encoding with relative position representations learned using the self-attention mechanism.

It should be noted that this proposed approach would only function when the positional-encodings are trained, instead of being pre-determined (Vaswani et al., 2017). Thanks to *relative* and *trained* positional-encodings, we were able to make the dimensionality of data-flow *flexible* across the network layers. As we highlight in Section 4, these heterogeneous architectures do indeed perform well, and in some cases outperform traditional, fixed and homogeneous models.

3.3 Block-Level Computational Graphs

To learn a lower-dimensional dense manifold of the given design space, characterized by a large number of FlexiBERT models, we convert each model into a computational graph. As described in Section 2.3, we formulate a computational graph based on the forward flow of connections for each compute block. For the design space in consideration, as presented in Table 1, the possible compute blocks include:

- For layer-j, we have 2, 4 or 8 attention heads with: h-128/s-SDP, h-128/s-WMA, h-256/s-SDP, and h-256/s-WMA.
- For layer-j, we have the size of the hidden-layer in the feed-forward network in: 512,

1024, or 2048.

- Other blocks like: Add&Norm, Input and Output.

In this context, we create all possible computational graphs within the design space for every transformer model. Further, we implement recursive hashing between every connection as follows: for every node in this graph, we concatenate the hash of its input, the hash of that node, and its output respectively and take the hash of the result. Doing this for all nodes gives us the hash of the resultant computational graph. This helps us in detecting isomorphic graphs which can be neglected to reduce the design space. For the range of design choices in Table 1 we obtained 2037 unique graphs after removing isomorphic graphs[1]. Here, we limit the design space to only increasing number of attention heads, hidden sizes and feed-forward dimensions (all combinations would results in a design space of size 332,352).

3.4 Transformer2vec

As described in Section 2.3, we compute the Weisfeiler-Lehman Subtree (WL) kernel for all possible computational graph-pairs in the design space. This gives us a matrix of similarity (or dissimilarity defined as 1 - similarity) for all possible graph-pairs, *i.e.* a matrix of size 2037 $\times$ 2037 for the design space in consideration.

For this dissimilarity matrix, we learn dense embeddings that try to map the Euclidean distance (for every graph-pair) with these dissimilarity values. More specifically, we run the SMACOF algorithm by Kruskal (1964) that minimizes the stress function:

$$\sigma(X) = \sum_{i<j\leq n} w_{ij}\big(d_{ij}(X) - \delta_{ij}\big)$$

where $w_{ij} \geq 0$ is the weight of the measurement between a pair of points (i, j), $d_{ij}(X)$ is the Euclidean distance and δ_{ij} is the target distance (given by the dissimilarity matrix). Here, $X \in R^{n\times m}$, where $n = 2037$ is the number of samples and m is the embedding size.

Figure 3 shows the distance prediction error in terms of the Frobenius norm between the dissimilarity matrix and the one generated by calculating the Euclidean distance between every possible pair. We see that at the embedding size of $m = 8$, the prediction error is minimized, and larger sizes for m give insignificant improvements. Thus, for the given design space, $m = 8$ is optimal. As larger design spaces are considered, this optimal m might increase.

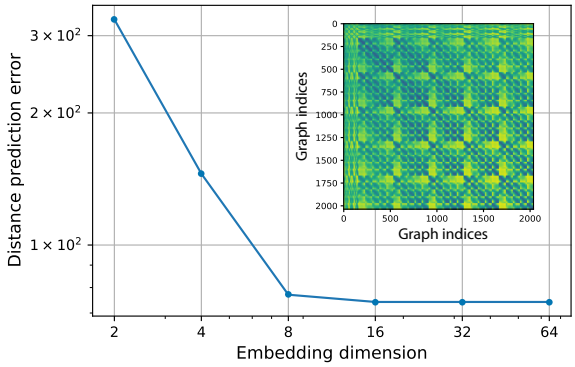

Figure 3: Frobenius norm between the predicted distances between all Computational Graph-pairs from the embeddings vs. those given by the WL kernel for different embedding dimension lengths. Inset: pairwise distance matrix for every graph-pair from the WL kernel.

Figure B.1 shows the annotated t-SNE plot of the trained embeddings of all the models in the design space. As can be seen from the inherent clusters formed, the embeddings indeed capture graph similarity between *neighboring* models. In this work, we hypothesize that *nearby* models will have *similar* accuracies, and thus these 'graph-similarity-based' embeddings along with a Lipschitz-continuous GP surrogate function should be able to reasonably capture the dynamics of the performance frontier across various architectures in the design space.

However, the WL kernel only gives sub-graph similarities by taking the overlap in graph nodes. It does not consider whether two nodes are *inherently* close or not. For example, a computational 'block' (or its respective graph node) for an attention head with $h = 128$ and $s =$ SDP would be closer to another attention block with say, $h = 256$ and $s =$ WMA, but should be farther from a block representing a feed-forward layer (say $f = 2048$). This *domain-knowledge* could be incorporated while calculating distances in the WL kernel to further improve the distance prediction and forms a part of future work.

[1]Details of such tests along with source-code available at: https://github.com/shikhartuli/txf_design-space

3.5 Weight-Transfer between Neighboring Models

Since we train several models on downstream tasks in the active learning process, it is computationally infeasible to pre-train each transformer. Instead, we use the fine-tuned neighbours to initialize the weights of the query model. For each query model, it's $k-$nearest neighbours are decided using the distance between their embeddings (we use $k = 10$). Naturally, we want to transfer weights from the corresponding fine-tuned neighbour that has the largest initial-overlap with the query as the models will intuitively share the same initial internal representations. The shared portion between the models is calculated by adding up the total parameters in initial consecutive layers in the neighbor model that share the same design choices as the query model.

Layers of the neighbor that are included, need to necessarily share the same number of attention heads (a^j) and hidden size (h^j) as the corresponding query layer and the counting stops if this condition is violated. The *overlap fraction* is defined as the ratio between the shared parameters and the total parameters in the query model. If the maximum overlap fraction is above an overlap threshold, the weights of the shared part from the corresponding neighbor are transferred to the query. If this constraint is not met, we look at the next most plausible query.

3.6 Learning the Surrogate Function

Figure 2(c) shows the pipeline implemented to *actively* learn the surrogate function. We start with four pre-trained models in the design space—BERT-Tiny, BERT-Mini, BERT-L-2/H-256 and BERT-L-4/H-128[2]. First, we fine-tune these models on two GLUE tasks, namely SST-2 and QNLI. Further works could include tasks beyond sequence classification. Once these models were fine-tuned, for each task, we initialize a GP surrogate function and fit it with these four points—having mean of the Gaussian set to the respective accuracies and the standard deviation set to zero.

Now, based on this initial model, we predict the accuracies and their respective uncertainties (standard deviation) for every model in the design space using the surrogate function at hand. To *greedily* minimize the overall uncertainty of the model, we query the next model which has the highest uncertainty under the overlap constraint. Once we reach a convergence criterion, such that the overall uncertainty in the model is less than some threshold, we stop the training process. Further details of this process are presented in Section A.1

[2]Pre-trained models available at: https://github.com/google-research/bert

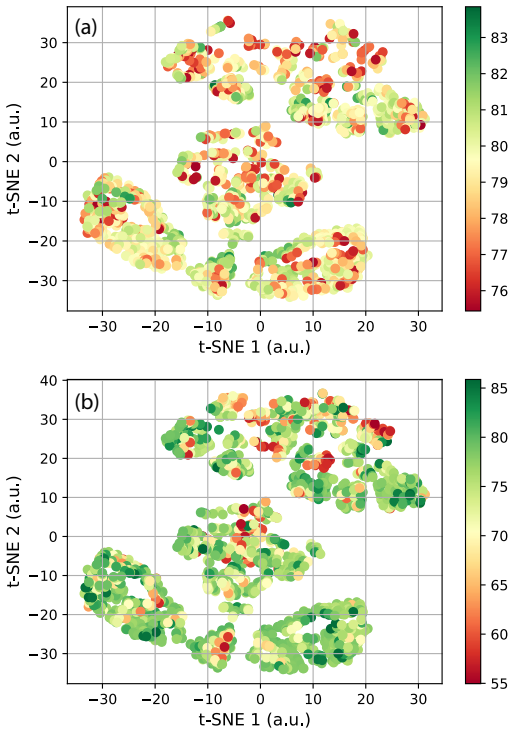

Figure 4: Scatter t-SNE plot of the embeddings, color-coded based on the accuracy of each model on (a) SST-2 and (b) QNLI tasks.

4 Results

In this section we present the results for various experiments that we performed.

4.1 Visualizing Accuracy Profiles

Figure 4(a) present the accuracies predicted by the surrogate function (for the SST-2 task) for all the points in the design space. Initially we hypothesized that we would see certain clusters predominantly with higher accuracies and other clusters with lower accuracies. However, this figure seems to indicate otherwise. Even within certain clusters, there is a diversity of design choices. For example, the cluster on the lower-right has architectures largely with hidden sizes h^js = 128. Still, other design choices have a high impact on the accuracies and they can not just be correlated with the design choice along one dimension. Based on the properties of these points and their respective accuracies, we make the following observations.

First, we observe that networks which are narrow initially (*i.e.* has lower h^j, f^j or a^j for initial layers) and wider in the deeper layers seem to perform better on the test set. This was one of the reasons why we limited the design space to only those architectures with increasing width across the model depth. A simple litmus test of a few models with decreasing width showed deterioration of performance, although a thorough study incorporating all combinations of architectural parameters forms a part of future work. Networks that are wider throughout have lower test accuracies, possibly because of poor generalization due to the over-parameterization in the initial layers.

Second, we observe that networks largely benefit from the WMA similarity in the deeper layers. This could be explained by the increasing level of abstraction of the learned representations as we go deeper into the network, resulting in the demand for a more flexible similarity metric between the key-query pairs. However, this is only true for networks where the hidden size increases too. For networks with WMA similarity in deeper layers, but a lower hidden size, we see that the performance is low. Simply adding WMA similarity onto homogeneous networks hurts performance. For example, for BERT-Mini, adding WMA similarity on all the attention heads leads to a drop in accuracy from 83.5% to 63.9%.

Figure 4(b) shows a substantially different distribution of accuracies for the QNLI task. This difference is expected, since the QNLI task involves a different set of attention-maps to be learned—where SST-2 only concerns positive/negative classification of movie reviews, QNLI requires the model to understand the context of two sentences and predict if the second answers the first (Wang et al., 2018).

4.2 Analysis

Table 2 shows the best and worst three models in the design space for the QNLI task. We see that the best models are heterogeneous and the worst performing models are mostly homogeneous. We also see that only specific combinations of design choices lead to high performance. This can also be seen in Figure 5. This plot is 'peaky' rather than monotonically increasing. This shows that larger models are not-necessarily better performing, although this is not to say that our primary goal is to test model performance with the model size, but rather with heterogeneous sets of design choices. The 'peaky' nature also suggests that only specific combinations of hyperparameters lead to higher accuracies, showing a high correspondence of these design choices among themselves. This merits future investigation by creating 'accuracy-aware' embeddings, forcing points into clusters with high/low accuracies in order to easily determine the correct set of design choices for a specific task.

	Top performing models with accuracies ∼83%
1.	{l: 4, a: [2, 4, 4, 4], h: [128, 128, 128, 256], s: [WMA, WMA, WMA, WMA], f: [1024, 2048, 2048, 2048]}
2	{l: 4, a: [4, 4, 4, 4], h: [128, 128, 256, 256], s: [SDP, WMA, WMA, WMA], f: [512, 512, 1024, 1024]}
3	{l: 4, a: [2, 2, 2, 2], h: [256, 256, 256, 256], s: [SDP, SDP, SDP, WMA], f: [1024, 1024, 1024, 2048]}
	Worst performing models with accuracies ∼57%
1.	{l: 4, a: [4, 4, 4, 4], h: [256, 256, 256, 256], s: [SDP, WMA, WMA, WMA], f: [1024, 1024, 2048, 2048]}
2	{l: 4, a: [2, 2, 4, 4], h: [128, 256, 256, 256], s: [SDP, SDP, SDP, SDP], f: [512, 512, 512, 2048]}
3	{l: 4, a: [4, 4, 4, 4], h: [128, 128, 256, 256], s: [WMA, WMA, WMA, WMA], f: [2048, 2048, 2048, 2048]}

Table 2: Best and worst three performing models in the design space for the QNLI task.

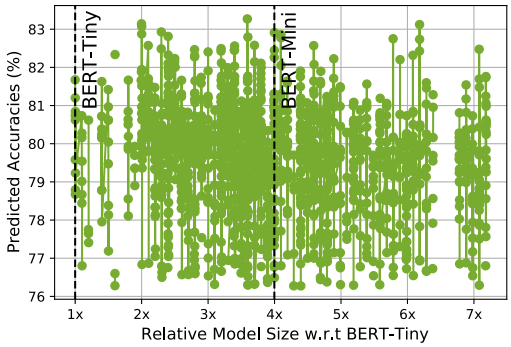

Figure 5: Performance vs. size for the SST-2 task.

5 Conclusion

With the rapid burst in the number of Transformer architectures being proposed in the NLP domain, there is a demand to benchmark these models on a common set of tasks and compare and contrast their performance on different tasks. In this work, we present a proof-of-concept targeting a small subset of this problem—spanning only architectural design choices and their performance on two sentence classification tasks. We observe that the correct set of design choices is different for every task and this dependency among such architectural design choices is highly convoluted, which merits further investigation. We present a few future working directions in Section A.2.

References

Tom B. Brown et al. 2020. Language models are few-shot learners. *CoRR*, abs/2005.14165.

Christopher M. Bishop. 1994. Mixture density networks. Workingpaper, Aston University.

Hsin-Pai Cheng, Tunhou Zhang, Yixing Zhang, Shiyu Li, Feng Liang, Feng Yan, Meng Li, Vikas Chandra, Hai Li, and Yiran Chen. 2020. NASGEM: Neural Architecture Search via Graph Embedding Method.

Xiaoliang Dai, Peizhao Zhang, Bichen Wu, Hongxu Yin, Fei Sun, Yanghan Wang, Marat Dukhan, Yunqing Hu, Yiming Wu, Yangqing Jia, Peter Vajda, Matt Uyttendaele, and Niraj K. Jha. 2019. Chamnet: Towards efficient network design through platform-aware model adaptation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*.

Yonatan Geifman and Ran El-Yaniv. 2018. Deep active learning with a neural architecture search. *CoRR*, abs/1811.07579.

Steve Hanneke. 2011. Rates of convergence in active learning. *The Annals of Statistics*, 39(1).

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition.

Xin He, Kaiyong Zhao, and Xiaowen Chu. 2021. Automl: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212:106622.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Zihang Jiang, Weihao Yu, Daquan Zhou, Yunpeng Chen, Jiashi Feng, and Shuicheng Yan. 2020. Convbert: Improving BERT with span-based dynamic convolution. *CoRR*, abs/2008.02496.

Ashish Khetan and Zohar Karnin. 2020. schuBERT: Optimizing elements of BERT. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2807–2818, Online. Association for Computational Linguistics.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar. Association for Computational Linguistics.

Karl Krauth, Edwin V Bonilla, Kurt Cutajar, and Maurizio Filippone. 2017. Autogp: Exploring the capabilities and limitations of gaussian process models. In *UAI 2017, Conference on Uncertainty in Artificial Intelligence, August 11-15, 2017, Sydney, Australia*, Sydney. © EURECOM. Personal use of this material is permitted. The definitive version of this paper was published in UAI 2017, Conference on Uncertainty in Artificial Intelligence, August 11-15, 2017, Sydney, Australia and is available at :.

J. B. Kruskal. 1964. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27.

Guillaume Lample and Alexis Conneau. 2019. Cross-lingual language model pretraining. *CoRR*, abs/1901.07291.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2019. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *CoRR*, abs/1910.13461.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2018. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52.

Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. 2018. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Hanna Mazzawi, X. Gonzalvo, A. Kraun, P. Sridhar, Niranjan Subrahmanya, I. Lopez-Moreno, H. Park, and Patrick Violette. 2019. Improving keyword spotting and language identification via neural architecture search at scale. In *INTERSPEECH*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, page 3111–3119, Red Hook, NY, USA. Curran Associates Inc.

Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. 2017. graph2vec: Learning distributed representations of graphs. *CoRR*, abs/1707.05005.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Carl Edward Rasmussen. 2004. *Gaussian Processes in Machine Learning*, pages 63–71. Springer Berlin Heidelberg, Berlin, Heidelberg.

Bernhard Schölkopf and Manfred K Warmuth. 2003. *Computational Learning Theory and Kernel Machines: 16th Annual Conference on Computational Learning Theory and 7th Kernel Workshop, COLT/Kernel 2003, Washington, DC, August 24-27, 2003, Proceedings*. Springer Berlin Heidelberg.

Burr Settles. 2009. Active learning literature survey.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-attention with relative position representations.

Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. 2011. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(77):2539–2561.

Julien Siems, Lucas Zimmer, Arber Zela, Jovita Lukasik, Margret Keuper, and Frank Hutter. 2020. Nas-bench-301 and the case for surrogate benchmarks for neural architecture search. *CoRR*, abs/2008.09777.

Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition.

David R. So, Chen Liang, and Quoc V. Le. 2019. The evolved transformer. *CoRR*, abs/1901.11117.

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

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *CoRR*, abs/1804.07461.

Chunyi Wang and Radford M. Neal. 2012. Gaussian process regression with heteroscedastic or non-gaussian residuals.

Colin White, Arber Zela, Binxin Ru, Yang Liu, and Frank Hutter. 2021. How powerful are performance predictors in neural architecture search? *CoRR*, abs/2104.01177.

Adrian de Wynter and Daniel J. Perry. 2020. Optimal subarchitecture extraction for BERT. *CoRR*, abs/2010.10499.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *CoRR*, abs/1906.08237.

Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. 2019. NAS-bench-101: Towards reproducible neural architecture search. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 7105–7114. PMLR.

Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2018. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Barret Zoph and Quoc V. Le. 2017. Neural architecture search with reinforcement learning.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. 2017. Learning transferable architectures for scalable image recognition. *CoRR*, abs/1707.07012.

A Appendices

A.1 Asynchronous Active Learning of the Surrogate Function

Once we have initialized the surrogate function based on the initial set of pre-trained models, we minimize the *overall* uncertainty by selecting the next query to be trained based on its uncertainty under an overlap constraint. The main idea is to select the model with the maximum uncertainty, but should have a minimum overlap of weights with the nearest fine-tuned model so that the training does not reach a local minimum. This is especially important since we are not pre-training any new model that is queried.

The pipeline can be explained as follows. We take the model with the highest uncertainty and test its overlap with its k-nearest neighbors. If the overlap is beyond the overlap constraint (we start this constraint with 90%), the model is chosen for querying into the FlexiBERT simulator and the weights are transferred from the neighbor with the maximum overlap. If this criterion is not reached, we test the next most-uncertain model, in the hope

that as the design space is explored, the model with the higher uncertainty will have a neighbor that is trained and satisfied the overlap constraint.

As more-and-more models are trained, we relax the overlap constraint throughout the training process. If the overlap constraint falls below 50%, we check all the points in the design space for the overlap rather than just the $k-$nearest neighbors. Training the surrogate converges when we can predict the accuracy for every model in the design space with 95% confidence-interval corresponding to $\pm 0.5\%$ accuracy, *i.e.* $1.96 \times \sigma < 0.5\%$.

We call our approach *asynchronous* because, as-soon-as a model is trained, we can instantly re-fit the surrogate function and query the next point, even in a distributed setting. If a queried model is being trained, we pick the next query with the next highest uncertainty. To train the surrogate functions for each task, we used four P-100 GPUs to run four most-uncertain models parallely in our active learning framework. For the SST-2 task, we were able to get to convergence by training just $\sim$30% models in the design space, thanks to the similarity-based embeddings and the GP surrogate function.

Algorithm 1 presents a high-level view of the asynchronous active learning framwork. Implementation details and source-code are available at `https://github.com/shikhartuli/txf_design-space`. The *modular* implementation of FlexiBERT was done by expanding on the `transformers` repository: `https://github.com/huggingface/transformers`.

Algorithm 1: Asynchronous Active Learning of the Surrogate Function

```
Result: surrogate function
Initialize: overlap threshold, confidence interval, surrogate;
while 1.96 × max(std) > confidence interval do
    wait till a worker is free;
    if worker is free then
        fit(surrogate, new acc point);
        acc, std ← predict(surrogate);
        query ← argmax(std);
        for neighbor close to query do
            overlap ← overlap(neighbor, query);
            if overlap > overlap threshold then
                send query to worker;
            end
        end
        if no query to worker then
            relax overlap threshold;
        end
    end
end
```

A.2 Future Work

In this section we present some future working directions. First, we could extend the design space spanning not just BERT-Tiny and BERT-Mini, but also larger models including BERT-Large. Further, we could also incorporate all possible design configurations (*i.e.* also consider models with wider layers in the beginning and shallower layers towards the end).

Second, domain-knowledge could be Incorporated into the graph-kernel for more precise comparison of graph similarity, as discussed in Section 3.4.

Third, MDS-based global embeddings are not easy to scale—each point in the design space requires distance from every other point. Thus, as the design space is enlarged, negative sampling could be used to generate graph-emeddings. Further, encoder-based embeddings could be explored, as explained in Section 2.3. Recently proposed graph-convolutional networks could also be used to generate these embeddings.

Fourth, we have directly fine-tuned models using only four pre-trained models in the design space. Thorough analysis is required to test the effect of pre-training every model and then fine-tuning them to the downstream task. It is also possible that a model far away from any of the pre-trained models has significant effect of the training recipe on its performance. Thus, automatic hyperparameter tuning needs to be incorporated to test this with multiple ablation studies (Liaw et al., 2018).

Fifth, Gaussian-Process Regression (GPR) has several limitations, including lower efficiency in high dimensional spaces (Rasmussen, 2004; Krauth et al., 2017). Further, a GPR assumes the residuals to have the same variance across all observations. This could be tackled with the use of a separate latent variable (Wang and Neal, 2012). Mixture-Density Networks (MDNs) could also be explored for larger design spaces. These have several ad-

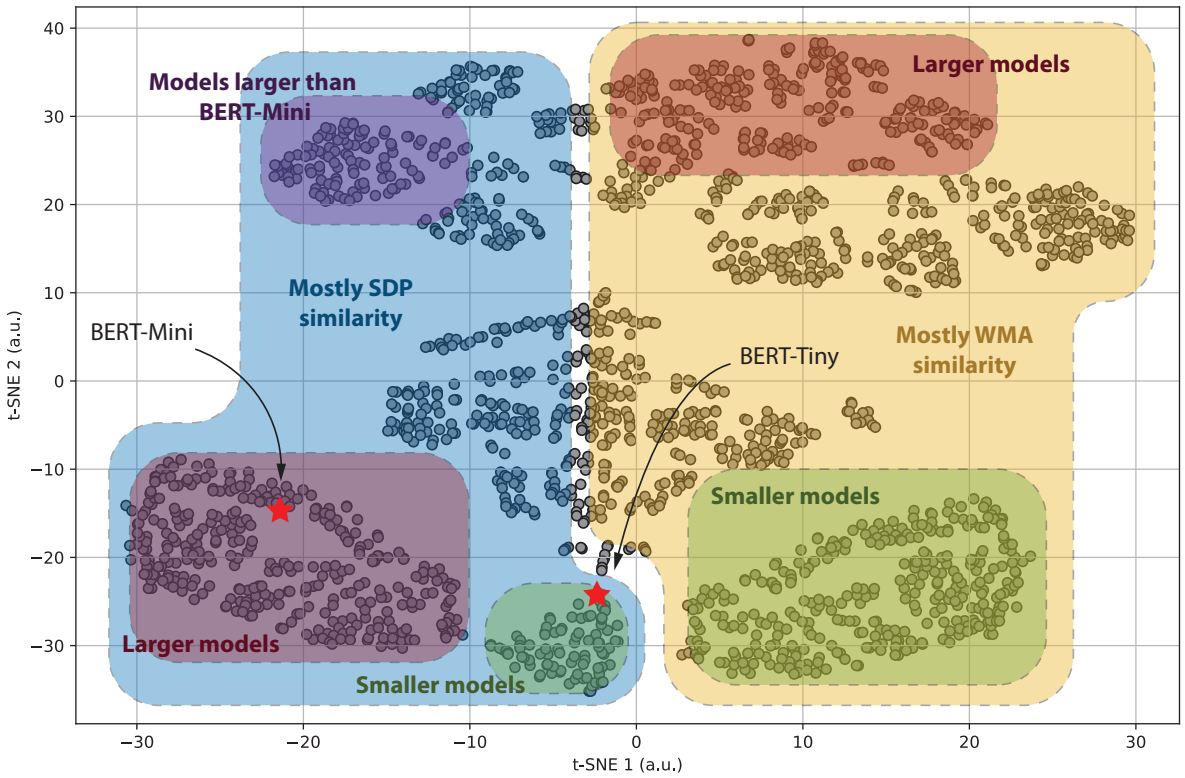

Figure B.1: Annotated t-SNE plot of the Transformer design space using the trained `Transformer2vec` embeddings. The design space incorporates pre-trained models including BERT-Tiny and BERT-Mini (shown as red stars). Transformer models have largely been segregated by the embeddings based on the similarity function—traditional-SDP or WMA, with the larger blue and yellow boxes respectively. Smaller models within each subspace have been shown by green boxes and larger ones by red boxes. The purple box highlights models that use SDP similarity, but are architecturally larger than BERT-Mini, *i.e.* have consistently larger dimension of the hidden layer in the feed-forward network (f^js). Points that are outside these boxes are more heterogeneous, *i.e.* these generally have a mixture of these properties. Axes have been plotted with arbitrary units.

vantages including the application of gradients to the input for faster convergence of the overall uncertainty (Bishop, 1994). Further, MDNs, along with a Nautural Parameter Network (NPN) could be used to model not just the epistemic uncertainty of the surrogate function, but also the aleatoric uncertainty of accuracy in training a specific model (stemming from randomly initialized weights).

Sixth, as described in Section 4.2, accuracy-aware self-supervised embeddings could be learned to automatically extract out good/bad design choices from the data. This would make it easier for academia and industry to get a set of best design choices for the downstream task at-hand, without worrying about intricate details of the inner machinery.

Seventh, this work could easily be extended to a larger space of transformers, even beyond BERT. For example, functional improvements like auto-regressive methods (Yang et al., 2019) and denoising approaches (Lewis et al., 2019) could be added into the design space in order to incorporate comparison of all state-of-the-art models and to be able to argue why one model outperforms the other in a specific task.

Finally, the active-learning approach could be improved to not just *purely-explore* but also *exploit* the design space—training models with lower uncertainty but higher overlap could help speed-up the training process.

B Supplementary Material

In Figure B.1 we present the Transformer design space after projecting the 8-dimensional embeddings onto a 2-D plot for visualization.