1 Introduction

The transformer architecture [12] has revolutionized the AI landscape and enabled the development of large language models (LLMs). However, as these models continue to grow in size. Current trends are forecating running out of high-quality data required for optimal performance [13]. This limitation stimulas the initiative to explore the impact of architectural decisions on smaller models, which this project aims to investigate. By understanding how to optimize smaller models, we can ensure progress of LLMs even with the projected lack of high-quality data.

The choice of activation functions has historically played a crucial role in the advancement of neural networks, such as the shift from Sigmoind activation functions to ReLU (Rectified Linear Unit), which significantly improved training speeds. As the era of large language models (LLMs) unfolded, the development of activation functions continued to evolve, resulting in over 400 documented activation functions [3]. Despite this progress, literature typically compares new activation functions against their immediate predecessors, leading to a gap in the literature: there is no comprehensive comparison of multiple activation functions under consistent conditions. This inconsistency is primarily due to variations in datasets and model sizes used in different studies, highlighting the need for a systematic evaluation. More specifically, with the trend of increasing model sizes, the investiation of impact activation functions have on smaller models was left neglected. This research aims to address this gap by exploring the impact of activation functions on smaller-scale language models with around 10 million parameters.

(Structure of the paper here?)

2 Background and related work

Activation functions are used to introduce non-linearity into neural networks, allowing them to model complex relationships in the data. They are applied to the nodes of the network and are essential for enabling the network to learn and perform a wide range of tasks, beyond what linear models can achieve.

The famous paper "Attention is All You Need" [12] used the state-of-the-art activation function at the time, ReLU. Since then, no significant improvements were made until the introduction of GELU, which quickly became the default activation function in most of the state-of-the-art LLMs [4]. The popularity of GELU stems from its ability to enhance model performance without introducing an efficiency overhead. Despite these advancements, continuous innovation leads to alternatives like GeGLU, noted for its effectiveness [11], also used in last year's winner of the BabyLM challenge [10]. However, a recently published paper suggests a return to ReLU [6], adding further confusion to the search for the optimal activation function. Fortunately, it also provides some clues that guide further exploration and motivate this research.

The paper suggests that suggests that the impact of activation functions diminishes as the model size increases, evident in models with over a billion parameters [6]. This also explains the initial move away from ReLU, since all the research on activation alternatives was done on models with the size of approximately 100 million parameters. Highlighting this finding further motivates the need to investigate activation functions in 10 million parameter models. The impact of activation functions is expected to be more significant in smaller models, and until now, decisions have been made with the trend of increasing model size in mind.

Furthermore, another gap appears in the research on activation functions with trainable parameters (adaptible activaition functions). The possible explanation for this could be the tradeoff between additional trainable parameters and performance. However, this was primarily studied in larger models. The only paper found on adaptive activation functions in LLMs is by Rajanand et al. [8], which found that adaptive activation functions outperform static ones in text-to-text machine translation [NOTE: add the exact numbers from the paper], a task closely related to language models. This suggests that adaptive activation functions could be beneficial for smaller models, but further research is needed to confirm this hypothesis. Given these insights, this research will explore the impact of various activation functions on smaller-scale language models with around 10 million parameters. Hypothesising that at smaller scales, the choice of activation function is crucial and having learnable parameters could be beneficial.

Kolmogorov-Arnold Networks (KAN) represent a recent development in neural network architecture, where activation functions are applied on edges instead of nodes [5]. This approach has been shown to outperform traditional neural networks in some tasks, particularly in scientific applications such as solving partial differential equations. However, at the time of this literature review, it has yet to be tested on language models. The primary benefit of KAN is the optimization of activation on each edge using splines. A spline is a piecewise-defined polynomial function used in interpolation and approximation to create smooth curves through a set of points [ADD REFERENCE]. With a spline on each edge, each edge can have its own custom activation function, trained separately and uniquely shaped. In contrast, adaptive activation functions have the same shape but different gradients. Although this comes with the drawback of an increased number of trainable parameters. This research will experiment with applying KAN to language modeling to assess its efficacy at smaller scales, filling the gap in current literature.

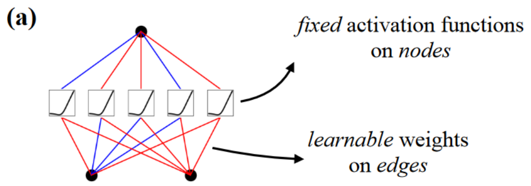
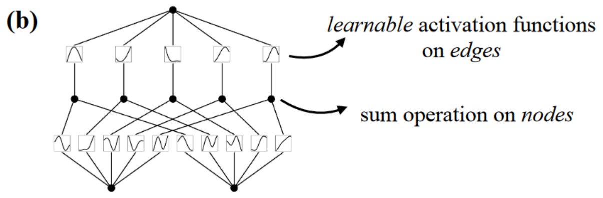
Model (Shallow)	 (a) *fixed* activation functions on *nodes* *learnable* weights on *edges*	 (b) *learnable* activation functions on *edges* sum operation on *nodes*

Figure 1: KAN vs MLP [3]

3 Approach

Transformers are composed of multiple layers, each playing a crucial role in processing input data and generating meaningful representations. Among these layers, the Feed-Forward Neural Network (FFN) layer typically consists of two linear transformations with an activation function in between, effectively forming a simple Multi-Layer Perceptron (MLP). This structure allows the default activation function to be switched out with different activation functions for testing, enabling a direct comparison of their performance while keeping everything else the same. To explore the effectiveness of various activation functions, I will modify existing implementations of prominent models like GPT-NEO and RoBERTa from the Hugging Face library.

3.1 Choosing activation functions

The activation functions to be evaluated are: ReLU, SiLU, Swish, PReLU, GELU, GEGLU, learnable GELU, and learnable GEGLU. Additionally, the KAN network will be compared against all of these options.

GELU

Currently, the most popular activation function in LLMs is also used as the default activation function in the baseline models GPT-NEO and roBERTa. It is a smooth approximation of ReLU, originally defined as $\mathrm{GELU}(x) = x \cdot \Phi(x)$, where $\Phi(x)$ is the Cumulative Distribution Function for the Gaussian Distribution. or optimization purposes, since calculating $\Phi(x)$ is computatinally expensive, it is instead calculated with the tanh approximation as: $\mathrm{GELU}(x) = 0.5x\left(1 + \tanh\left(\sqrt{\frac{2}{\pi}}\left(x + 0.044715x^3\right)\right)\right)$ [2]. This function will be used as a baseline for comparison with all other activations. [add figure of GELU]

ReLU and PReLU

ReLU was considered state-of-the-art at the time of the original transformer paper [12], but has since been surpassed by other activation functions. Baseline models are implemented using the PyTorch library, which prvides an implementation for the ReLU activation function [RELU citation from docs], which will be used in this research.

$$\mathrm{ReLU}(x) = \max(0, x)$$

This implementation will be used as a baseline for comparison with *PReLU* and *GELU*. The comparison with GELU is motivated by the findings of *I. Mirzadeh and Others* [6], which suggest that the use of ReLU is acceptable as the impact of activation functions diminishes with increasing model size. The objective is to determine how much worse (if at all) ReLU is compared to GELU when used with smaller models.

PReLU is a variant of ReLU that incorporates learnable parameters, allowing the activation function to adaptively learn the optimal slope for negative values. The PReLU function is available in the PyTorch library [pytorch citation]. It is mathematically defined as

$$\mathrm{PReLU}(x) = \max(0, x) + a\min(0, x)$$

where a is a learnable parameter. It takes the *num_parameters* parameter which was set to *intermediate_size*. The objective is to evaluate whether adding a learnable parameter to ReLU can enhance performance and to measure the impact on training time. [ADD FIGURE OF RELU]

SiLU and Swish

SiLU was initialy evaluated on LLMs in the original GELU paper [2] but was found to perform worse than the GELU activation fununction. It is defined as

$$\mathrm{silu}(x) = x \cdot \sigma(x), \text{ where } \sigma(x) \text{ is the logistic sigmoid.}$$

PyTorch implementation of SiLU was used [cite pytorch silu]. It will be used only as a baseline comparison for the *Swish* activation function, which is its counterpart with learnable parameter, aiding the objective of exploring the impact of adding learnable parameters to activation functions.

Swish is a self-gated activation function that was proposed by *Ramachandran et al.* [9]. It is not available in the PyTorch library so it was implemented as proposed in the paper

$$\mathrm{swish}(x) = x \cdot F.\mathrm{silu}(\beta \cdot x)$$

where β is a learnable parameter. Using the idea seen in PyTorch PReLU implementation the *num_parameters* parameter, which determines the number of learnable parameters was added and set to *intermediate_size*. The objective is to evaluate whether the Swish activation function can outperform SiLU and to measure the impact of adding learnable parameters to activation functions.

[ADD FIGURE OF SILU]

Learnable GELU

The GELU activation function will be parameterized with learnable parameter. The implementation will be based on the PyTorch GELU implementation with tanh approximation, which, out of the box, does not support learnable parameters. This approach appears to be novel, as no prior research has been found that explores the impact of adding learnable parameters to the GELU activation function. The new GELU activation function adds a learnable parameter as a scaling factor and is defined as follows:

$$0.5 \cdot \beta \cdot \text{input} \cdot \left(1.0 + \tanh\left(\sqrt{\frac{2.0}{\pi}} \cdot \left(\text{input} + 0.044715 \cdot \text{input}^3\right)\right)\right)$$

where β is a learnable parameter.

The *num_parameters* parameter was set to *intermediate_size* to determine the number of learnable parameters. The objective is to evaluate whether the learnable GELU activation function can outperform the standard GELU activation function and to measure the impact of adding learnable parameters to activation functions.

GEGLU and Learnable GEGLU

The *GEGLU* activation function is a variant of the GELU activation function that incorporates a gating mechanism as proposed by *Shazeer et al.* [11]. It promises to improve the performance of the GELU activation function by adding a gating mechanism. The GEGLU activation function was implemented as used by 2023 BabyLM winner [10] [7]:

```
class GeGLU(nn.Module):
 def forward(self, x):
    x, gate = x.chunk(2, dim=-1)
    x = x * F.gelu(gate, approximate='tanh')
    return x
```

Listing 1: Implementation of GeGLU

The *Learnable GEGLU* activation function is an enhanced variant of the GEGLU activation function, incorporating a learnable parameter following the same idea as Learnable GEGLU. This implementation extends the standard GEGLU by introducing a learnable scaling factor, making it a novel approach. The new Learnable GEGLU activation function is defined as follows:

```
def forward(self, input: Tensor) -> Tensor:
    x, gate = input.chunk(2, dim=-1)
    x = x * 0.5 * self.beta * gate * (1.0 +
        torch.tanh(math.sqrt(2.0 / math.pi) * (
        gate + 0.044715 * torch.pow(gate, 3.0)))
        )
    return x
```

Listing 2: Implementation of Learnable GeGLU

The objective is to first evaluate the performance of GeGLU compared to GELU, and then to evaluate the performance of Learnable GeGLU compared to GeGLU. The hypothesis is that Learnable GeGLU will be the best-performing activation function.

KAN-Network

The KAN network is a novel activation function that has shown promising results in the literature. The implementation of the KAN network is available but has shown to be problematic and slow. To address these issues, I will utilize an efficient-KAN implementation [1]. This approach requires careful selection of certain parameters, specifically the number of grid intervals (int) and the order of piecewise polynomials (k). Based on recommendations from the paper, I will set these parameters to 3 and 5, respectively. This configuration aims to balance performance and computational efficiency, ensuring that the KAN implementation is both effective and practical for the experiments. FFN layers from GPT-NEO and roBERTa both use MLPs in their implementation, which can be directly replaced with efficient-KAN implementation. As KAN netowork adds additional trainable parameters, the intermediate size for GPT-NEO was decreased to 256 and for roBERTa to 384 to keep the number of parameters around 10M as for other models.

The objective is to evaluate the performance of the KAN network compared to the other activation functions and to determine whether it is a viable alternative for LLMs.

4 Results

Model	**BLiMP**	**G**
BERT-Learnable-GELU-9.0M-2L-4H-512C-1024I	59.4	
GPT-Learnable-GELU-9.0M-2L-4H-512C-1024	56.782	6
Same baselines		
GPT-9.0M-2L-4H-512C-1024I	55.6	5
BERT-9.0M-2L-4H-512C-1024I	49.1	5
Best baselines GLUE		
GPT-11.0M-3L-4H-512C-1024I-0	(55.17)	
BERT-11.0M-3L-4H-512C-1024I	(49.49)	5
GPT-PReLU-9.0M-2L-4H-512C-1024I	57.0%	
BERT-PReLU-9.0M-2L-4H-512C-1024I	59.2%	
GPT-SiLU-9.0M-2L-4H-512C-1024I	56.3%	
GPT-ReLU-9.0M-2L-4H-512C-1024I	56.6%	
BERT-SiLU-9.0M-2L-4H-512C-1024I	58.3%	
BERT-ReLU-9.0M-2L-4H-512C-1024I	58.5%	
GPT-Swish-9.0M-2L-4H-512C-1024I	53.7%	
BERT-Swish-9.0M-2L-4H-512C-1024I	58.0%	
BERT-kan2-11.0M-2L-4H-512C-384I	56.69%	63
KAN2-GPT-11.0M-2L-4H-512C-256I	63.43%	48
Learnable GELU Seeds		
GPT-Learnable-GELU-seed4-9.0M-2L-4H-512C-1024I	56.0%	
GPT-Learnable-GELU-seed1-9.0M-2L-4H-512C-1024I	56.3%	
GPT-Learnable-GELU-seed2-9.0M-2L-4H-512C-1024I	57.4%	
BERT-Learnable-GELU-seed4-9.0M-2L-4H-512C-1024I	58.2%	
GPT-Learnable-GELU-seed5-9.0M-2L-4H-512C-1024I	58.3%	
GPT-Learnable-GELU-seed3-9.0M-2L-4H-512C-1024I	58.7%	
BERT-Learnable-GELU-seed3-9.0M-2L-4H-512C-1024I	59.0%	
BERT-Learnable-GELU-seed2-9.0M-2L-4H-512C-1024I	59.3%	
BERT-Learnable-GELU-seed5-9.0M-2L-4H-512C-1024I	59.8%	
BERT-Learnable-GELU-seed1-9.0M-2L-4H-512C-1024I	59.8%	

Table 1: Comparison of models across BLiMP and GLUE datasets

References

[1] Blealtan. Efficient-kan, 2024. Accessed: 2024-06-03.

[2] D. Hendrycks and K. Gimpel. Gaussian error linear units (gelus). *arXiv*, arXiv:1606.08415, Jun. 2023.

[3] V. Kunc and J. Kléma. Three decades of activations: A comprehensive survey of 400 activation functions for neural networks. *arXiv*, arXiv:2402.09092, 2024.

[4] M. Lee. Gelu activation function in deep learning: A comprehensive mathematical analysis and performance. *arXiv*, arXiv:2305.12073, Aug 2023.

[5] Z. Liu and Others. Kan: Kolmogorov-arnold networks. *arXiv*, arXiv:2404.19756, May 2024.

[6] I. Mirzadeh and Others. Relu strikes back: Exploiting activation sparsity in large language models. *arXiv*, arXiv:2310.04564, Oct 2023.

[7] LTG Oslo. Ltg bert, 2021. Accessed: 2024-05-15.

[8] A. Rajanand and P. Singh. Erfrelu: Adaptive activation function for deep neural network.

[9] P. Ramachandran, B. Zoph, and Q. V. Le. Searching for activation functions. *arXiv*, arXiv:1710.05941, Oct. 2017.

[10] D. Samuel, A. Kutuzov, L. Øvrelid, and E. Velldal. Trained on 100 million words and still in shape: Bert meets british national corpus. *arXiv*, arXiv:2303.09859, May 2023.

[11] N. Shazeer. Glu variants improve transformer. *arXiv*, arXiv:2002.05202, Feb 2020.

[12] A. Vaswani and Others. Attention is all you need. *arXiv*, arXiv:1706.03762v5, Dec 2017.

[13] P. Villalobos, J. Sevilla, L. Heim, T. Besiroglu, M. Hobbhahn, and A. Ho. Will we run out of data? an analysis of the limits of scaling datasets in machine learning. *arXiv*, arXiv:2211.04325, Oct 2022.