Can Neural Architecture Search Help us Find Faster LLM Architectures? Experiments with GPT-2 based Text Predictor

Anonymous ACL submission

Abstract

Inference with Large Language Models (LLMs) is costly and often dominates the life-cycle cost of LLM-based services. Neural Architecture Search (NAS) can automatically find architectures optimizing the trade-offs between accuracy and inference cost. However, NAS for LLM architectures is computationally prohibitive. We apply the recently proposed LiteTransformerSearch (Javaheripi et al., 2022) algorithm to reduce the inference latency of a GPT-2 based Text Prediction system by 25% without compromising its accuracy. In the process, we discover some new constraints that apply on the optimal neural architectures, and are, therefore, useful in practice to further reduce the computational cost of NAS.

1 Introduction

LLMs have achieved state-of-the-art results in multiple domains and tasks (Zhao et al., 2023), but scalable deployment can be hampered by the high computational costs, large memory footprint and high inference latency (Amodei et al., 2020). Various methods have been proposed to mitigate these issues (Dao et al., 2022; Ling et al., 2023; Liang et al., 2021) that optimize a trained model. Neural Architecture Search (NAS) automates the discovery of optimal architectures for given tasks and hardware. NAS can explore complex architecture spaces considering predefined objectives and leveraging prior knowledge or human expertise, and has been successfully applied to various classes of neural networks (Elsken et al., 2019). Lately, NAS is gaining traction for improving Transformer-based architectures that balances accuracy and efficiency (Chitty-Venkata et al., 2022).

Performance estimation is crucial in NAS, but the conventional method, fully training candidate architectures, is extremely challenging for LLMs due to high computational costs. For example, training and ranking 1200 TransformerXL candidates takes about 19K GPU hours, (Javaheripi et al., 2022). Multiple methods have been proposed to reduce computational needs in performance estimation, including weight sharing and one shot methods (Xie et al., 2023). A recent promising development in this space is the LiteTransformerSearch (*LTS*) algorithm (Javaheripi et al., 2022) that proposes a zero-cost proxy.

In this paper, we present a case study of using LTS to reduce the latency of a real-world commercial web-scale text prediction system. Text prediction enhances typing efficiency by offering real-time, context-dependent word and phrase suggestions while a user is typing (Vashishtha et al., 2023; Chen et al., 2019; Garay-Vitoria and Abascal, 2006). Our system uses GPT-2 style auto-regressive transformer for inference. Using LTS, we reduced the latency by 25% while maintaining prediction quality. In the process, we also discovered a set of constraints on the parameters of the architecture, which helped us further limiting the search space and reducing the computational cost.

2 Related Work

The origins of NAS can be traced back to the 1980s, when genetic algorithm-based methods were in fashion (Schaffer et al., 1992). In the early 2000s, the concept of NEAT (Neuro Evolution of Augmenting Topologies) (Stanley and Miikkulainen, 2002) was proposed, which involves the artificial evolution of neural networks using crossover of different network topologies. However, these methods were not able to achieve the performance of hand-crafted architectures at that time.

Around 2015, NAS architectures started to approach or surpass the performance of human-designed network architectures specifically for CNNs. This triggered industry wide efforts to utilize NAS for discovering better neural architectures and led to the development of several frameworks, notably Microsoft ArchAI (Arc, 2022), Microsoft NNI (Microsoft, 2021), and Keras AutoML(Jin et al., 2023). Multiple benchmarking studies were performed (Ying et al., 2019; Tu et al., 2022; Chitty-Venkata et al., 2023) that evaluated NAS methods on various tasks.

NAS techniques were also utilized for Transformer architecture (Vaswani et al., 2017), Evolved Transformer (So et al., 2019) being one of the first applications. Evolved Transformer achieves the same quality (BLEU score) with half the FLOPs. Liu et al. (2022) proposed Efficient Transformers having mixed attention search space that helped discover architectures and select appropriate attention mechanism to maintain comparable accuracy to the standard Transformer while significantly improving inference latency. See Chitty-Venkata et al. (2022) for a survey of NAS methods applied to transformers.

While effective in discovering better network configurations for transformers, NAS has high computational cost for performance evaluation. To address this issue, many efficient methods have been proposed that can approximate the performance without fully training the architectures in every iteration. See Xie et al. (2023) for a comprehensive survey. Efficient performance evaluation is especially important for transformer architectures, as they have much higher training cost than other neural architectures. LTS (Javaheripi et al., 2022) presents a specialized training-free NAS for efficient language models, using the number of decoder parameters in auto-regressive Transformers as a proxy for task performance. This enables zero-shot performance estimation leading to fast architecture search.

3 Problem Formulation

The overall objective of our work is to employ NAS to find an *architecture* that when trained with data $\mathcal{D}$ and training algorithm $\mathcal{A}$, produces a *model* that has similar *accuracy* (or functional performance) but significantly reduced *latency* (or inference cost) with respect to an existing model that was also trained similarly. This latter architecture/model will be referred to as the *baseline model*.[1]

[1]Since, data $\mathcal{D}$ and training algorithm $\mathcal{A}$ (including training hyperparameters) as well as the inference hardware are assumed to be fixed for a given NAS setup, conceptually, there is a one-to-one mapping between the architectures and the models (subject to minor stochastic variations).

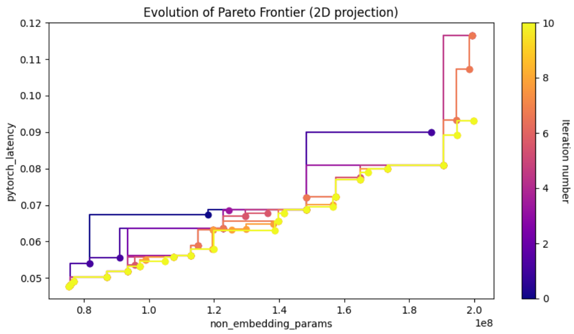

Figure 1: Evolution of the Pareto Frontier with the Search iterations. The last (yellow) line shows the final frontier, denoting the trade-off between the performance and model latency.

3.1 Baseline Model

The baseline model is a 12-layer GPT-2 style transformer (Radford et al., 2019) based text predictor with 204M parameters. Pretraining used 300B tokens from Pile (Gao et al., 2020) with next-word prediction task and evaluation was done on LAMBADA (Paperno et al., 2016). The finetuning and evaluation were respectively done on a custom sentence completion and held-out datasets. To determine if a prediction should be displayed and to limit generation length, a stopping logic based on log probabilities was applied, tuned using various performance metrics (Chen et al., 2019). The final model's functional performance was assessed on a user-generated test set. The model's inference latency was optimized using ONNX Runtime (developers, 2021), a cross-platform machine-learning accelerator for transformer models.

3.2 Objectives and Constraints

Our main optimization criterion is *minimization of inference latency* of the model, subject to the constraint that the *prediction quality* is not compromised. **Inference Latency** is the time taken to generate a prediction. *Prompt Latency*, PL, is the time taken by model to generate the first token and *Token Latency*, TL, is the average time to generate the subsequent tokens.

The *latency per character* is given by $latency\_char(n) = (PL + (n-1) * TL)/n$ for comparison. To remove any outliers, we use the 95^{th} percentile latency, (*P95 latency_char*), the time in which 95% of the inferences are completed.

Prediction Quality: The metrics used are: *Pretraining PPL*: Perplexity on LAMBADA. *Finetuning PPL*: Perplexity on the test set of the finetuning

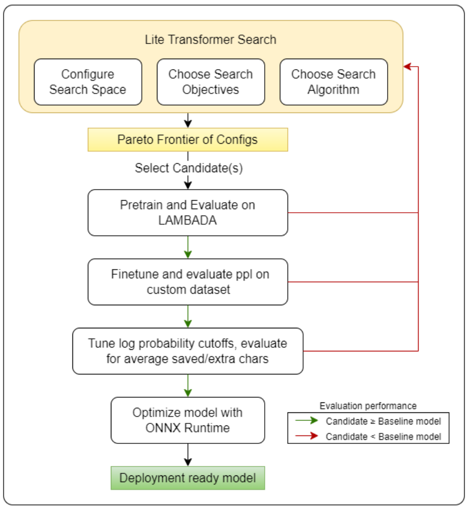

Figure 2: Our Optimization pipeline.

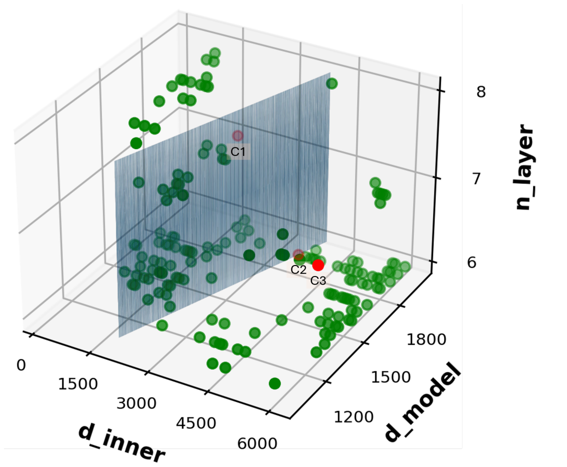

Figure 3: Architectures from the Pareto-frontiers are shown as green points. The red points represent architectures chosen for training and evaluation. The $d_{inner}/d_{model} = 2$ plane is shown in blue. Points in front of the plane are good architectures.

data. *Trigger rate*: The fraction of inputs for which a prediction is generated. *Average Saved Characters (ASC)* is the average number of characters accepted per prediction given. *Average Extra Characters (AEC)* is the average number of characters rejected per prediction. A higher ASC signals time saved by a user and hence a useful prediction while a higher AEC means a higher cognitive load.

4 Approach

We use the approach from LTS to optimize our model, given the constraints and optimization criteria. Fig 2 shows a schematic of the method which is described below.

4.1 Optimization Process

We defined the NAS search space as follows: (we use the notation $\{p_{min}, \ldots, p_{max}|step\_size\}$ to show the ranges used): $n_{layer} \in \{6, \ldots, 18|1\}$, $d_{model} \in \{512, \ldots, 2048|64\}$, $d_{inner} \in \{1024, \ldots, 8192|64\}$ and $n_{head} \in \{4, 8, 16, 32\}$. These parameter ranges contain values of the baseline model (See Table: 1) and allow for variation. As suggested by LTS, we use the number of decoder parameters as the proxy for performance, and set minimization of model latency as the optimization objective. Both metrics can be calculated without expensive model training. Number of decoder parameters varied between 120M and 180M as the baseline had 151M decoder parameters.

LTS performs an *evolutionary search* on candidate architectures to extract better models from the search space over multiple iterations. Since the metrics tend not to be correlated, the search ends up with a Pareto-Frontier (looking like Figure: 1) from which one can choose a specific configuration for training and further optimization. We chose a candidate with slightly more decoder parameters than the baseline as it would ensure better performance. Increasing this parameter count any further would also increase the model latency.

The selected configuration can be pretrained and evaluated similarly to the baseline as described in Sec 3.1. The various evaluations help verify the performance of the candidates as per Fig 2. If no candidate model performs acceptably, we go back to the architecture search and re-run it with additional constraints and heuristics.

5 Experiments and Results

Our initial search yielded a 7 layer, 260M parameter candidate (C1) with 156M decoder parameters and a 35% reduction in per token latency. However, after pretraining, this model performed worse than the original model on LAMBADA. It was observed that ratio of d_{inner} to d_{model} significantly affects quality as the configurations having smaller d_{inner} than d_{model} might hamper learning of important features in the intermediate layers. Therefore, the search was performed again with the constraint that $d_{inner}/d_{model} \geq 2$ (see Fig 3 for illustration).

Fig 1 shows the Pareto-frontier for the run with 10 iterations. The evolution of the Pareto frontier can be seen with the yellow points being the final frontier. The frontier did not change much with further iterations. Two configurations, C2 and C3, were chosen from the frontier and their metrics are presented in Table 1. In addition, we create

Model	d_{inner}	d_{model}	n_{head}	n_{layer}	Prompt latency	Per token latency	Decoder params	Total params
baseline	4096	1024	16	12	47.58	11.78	151M	204M
C1	1664	1984	4	7	$47.15_{\downarrow 0.90\%}$	$7.61_{\downarrow 35.40\%}$	156M	260M
C2	3712	1856	4	6	$46.62_{\downarrow 2.02\%}$	$6.58_{\downarrow 44.14\%}$	165M	262M
C3	6016	1280	4	7	$41.16_{\downarrow 13.16\%}$	$7.28_{\downarrow 40.32\%}$	153M	220M

Table 1: Configurations for the baseline and the candidate models along with their measured latencies (in ms). The candidates were chosen such that their decoder parameters are more than those of the baseline.

Model	PPL	Acc	prompt latency	token latency
baseline	15.77	0.4363	47.58	11.78
C1	21.86	0.3835	47.15	7.61
C2	19.61	0.4052	46.62	6.58
C3-h4	19.27	0.4058	41.16	7.28
C3-h8	17.05	0.4231	41.76	7.31
C3-h16	15.83	0.4438	44.85	7.56

Table 2: Perplexity, Accuracy and Latency (in ms) of the candidates on LAMBADA dataset after pretraining.

Metric	baseline	C3-h16
ASC (to maximize)	11.18	11.44
AEC (to minimize)	2.18	1.94
P95 latency_char	1.62	$1.24_{\downarrow 23.46\%}$

Table 3: Final Evaluation Results. Latency in ms.

two more variants of each configuration by setting the value of n_{head} to 8 (*-h8*) and 16 (*-h16*). This heuristic is based on the observation that increasing the number of attention heads increases the model performance with very little increase in latency.

The pretraining performances of these 6 architectures are shown in Table 2 along with prompt and token latency. The training was done on 64 A100 40GB GPUs and took 26 hours each. C2-h4 and C2-h8 models substantially underperformed, and were not chosen for any further optimization (hence not reported in Table 2). After pretraining, the candidate having lowest LAMBADA ppl, C3-h16, was finetuned on the custom dataset. On another custom dev set, we finetuned certain thresholds to ensure trigger rate and character saving rate equal to that of the baseline model. For the deployment, the model was converted from PyTorch to ONNX format and compute graph optimized by OnnxRuntime to further reduce latency.

Table 3 presents the performance results for C3-h16 on the held out evaluation dataset. As we can see, the configuration discovered by NAS is not only better in terms of prediction quality from the baseline (it has higher ASC and lower AEC rate), but also has 25% lower latency. This model has now been deployed in real-world scenario, where we are observing similar performance and latency profiles as predicted by the offline evaluation.

6 Conclusion and Future Work

Evaluating the performance of large architectures during NAS is computationally expensive, which has been a major bottleneck in applying NAS for LLMs. LTS provides a reliable one-shot proxy for estimating performance. In this paper, we demonstrated that LTS can indeed help us find configurations that can have much lower latency and consequently, lower computational cost, while maintaining the same level of end-task accuracy. In particular, we were able to reduce the P95 latency per character by 23.46% for a large GPT-2 style model with 204 million parameters.

We would also like to highlight two important practical discoveries of our work which is not mentioned in the original LTS algorithm. First, having a d_inner/d_model ratio larger than 2 significantly helps with model's quality. Second, increasing the number of attention heads, n_{head}, in an already discovered configuration also helps with quality improvements with only a slight increase in latency.

There are several open questions that this study prompts, which and can be explored in the future: Does the decoder parameter-end task accuracy link hold for models with 100+ billion parameters? Is there an equivalent for optimizing encoder-only models like BERT and RoBERTa? Can this technique be applied to LLMs trained with instruction fine-tuning and RLHF?

7 Limitations

Our Text Prediction model, based on GPT-2 and using Lite Transformer Search to optimize, has shown promising improvements in our experiments. However, some limitations to its use should be considered.

One limitation is that Lite Transformer Search's training-free proxy for model performance only applies to decoder-only models. This means it cannot be used to optimize encoder-based models like BERT, used widely in the industry at scale.

Another essential point is that this method does not help modify an existing model. Instead, a new model must be trained from scratch. This can be a resource and time-consuming process for large models and may only be feasible for some applications.

Another limitation is that there are very few changeable parameters within the Lite Transformer Search algorithm. This limits the ability to experiment with different activation functions and other hyperparameters, which could improve the model's performance. Currently, it offers no way to compare two models with different activation functions if they were to have the same number of decoder parameters. Further research is needed to determine if there are ways to increase the algorithm's flexibility to incorporate more dimensions into the search space.

Finally, it still needs to be clarified if Lite Transformer Search also works with flash attention. *Flash Attention* (Dao et al., 2022) is a relatively new technique that has shown promise in improving the performance of transformer models. Further experiments are needed to determine if Lite Transformer Search can be effectively combined with flash attention to improve the performance of our Text Prediction model.

Overall, while our Text Prediction model has shown promising results, some limitations to its use should be considered when evaluating its potential for other real-world applications.

References

2022. Archai: Platform for Neural Architecture Search.

Dario Amodei, Danny Hernandez, Girish Sastry, Jack Clark, Greg Brockman, and Ilya Sutskever. 2020. Ai and compute, 2018. *URL https://openai. com/blog/ai-and-compute*, 4.

Mia Xu Chen, Benjamin N Lee, Gagan Bansal, Yuan Cao, Shuyuan Zhang, Justin Lu, Jackie Tsay, Yinan Wang, Andrew M. Dai, Zhifeng Chen, Timothy Sohn, and Yonghui Wu. 2019. Gmail smart compose: Real-time assisted writing.

Krishna Teja Chitty-Venkata, Murali Emani, Venkatram Vishwanath, and Arun K. Somani. 2022. Neural architecture search for transformers: A survey. *IEEE Access*, 10:108374–108412.

Krishna Teja Chitty-Venkata, Murali Emani, Venkatram Vishwanath, and Arun K. Somani. 2023. Neural architecture search benchmarks: Insights and survey. *IEEE Access*, 11:25217–25236.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness.

ONNX Runtime developers. 2021. Onnx runtime. `https://onnxruntime.ai/`. Version: x.y.z.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Neural architecture search: A survey.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. 2020. The Pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*.

Nestor Garay-Vitoria and Julio Abascal. 2006. Text prediction systems: A survey. *Univers. Access Inf. Soc.*, 4(3):188–203.

Mojan Javaheripi, Gustavo H. de Rosa, Subhabrata Mukherjee, Shital Shah, Tomasz L. Religa, Caio C. T. Mendes, Sebastien Bubeck, Farinaz Koushanfar, and Debadeepta Dey. 2022. Litetransformersearch: Training-free neural architecture search for efficient language models.

Haifeng Jin, François Chollet, Qingquan Song, and Xia Hu. 2023. Autokeras: An automl library for deep learning. *Journal of Machine Learning Research*, 24(6):1–6.

Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. 2021. Pruning and quantization for deep neural network acceleration: A survey.

Chen Ling, Xujiang Zhao, Jiaying Lu, Chengyuan Deng, Can Zheng, Junxiang Wang, Tanmoy Chowdhury, Yun Li, Hejie Cui, Xuchao Zhang, Tianjiao Zhao, Amit Panalkar, Wei Cheng, Haoyu Wang, Yanchi Liu, Zhengzhang Chen, Haifeng Chen, Chris White, Quanquan Gu, Jian Pei, Carl Yang, and Liang Zhao. 2023. Domain specialization as the key to make large language models disruptive: A comprehensive survey.

Zexiang Liu, Dong Li, Kaiyue Lu, Zhen Qin, Weixuan Sun, Jiacheng Xu, and Yiran Zhong. 2022. Neural architecture search on efficient transformers and beyond.

Microsoft. 2021. Neural Network Intelligence.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernandez. 2016. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1525–1534, Berlin, Germany. Association for Computational Linguistics.

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

J.D. Schaffer, D. Whitley, and L.J. Eshelman. 1992. Combinations of genetic algorithms and neural networks: a survey of the state of the art. In *[Proceedings] COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 1–37.

David So, Quoc Le, and Chen Liang. 2019. The evolved transformer. In *International conference on machine learning*, pages 5877–5886. PMLR.

K.O. Stanley and R. Miikkulainen. 2002. Efficient evolution of neural network topologies. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, volume 2, pages 1757–1762 vol.2.

Renbo Tu, Nicholas Roberts, Misha Khodak, Junhong Shen, Frederic Sala, and Ameet Talwalkar. 2022. Nas-bench-360: Benchmarking neural architecture search on diverse tasks. In *Advances in Neural Information Processing Systems*, volume 35, pages 12380–12394. Curran Associates, Inc.

Aniket Vashishtha, S Sai Prasad, Payal Bajaj, Vishrav Chaudhary, Kate Cook, Sandipan Dandapat, Sunayana Sitaram, and Monojit Choudhury. 2023. Performance and risk trade-offs for multi-word text prediction at scale. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 2226–2242, Dubrovnik, Croatia. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Xiangning Xie, Xiaotian Song, Zeqiong Lv, Gary G. Yen, Weiping Ding, and Yanan Sun. 2023. Efficient evaluation methods for neural architecture search: A survey.

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, Long Beach, California, USA. PMLR.

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. 2023. A survey of large language models.