LLM Performance Predictors are good initializers for Architecture Search

Anonymous ACL submission

Abstract

In this work, we utilize Large Language Models (LLMs) for a novel use case: constructing Performance Predictors (PP) that estimate the performance of specific deep neural network architectures on downstream tasks. We create *PP prompts* for LLMs, comprising (i) *role* descriptions, (ii) *instructions* for the LLM, (iii) *hyperparameter* definitions, and (iv) *demonstrations* presenting sample architectures with efficiency metrics and 'training from scratch' performance. In machine translation (MT) tasks, GPT-4 with our PP prompts (LLM-PP) achieves a SoTA mean absolute error and a slight degradation in rank correlation coefficient compared to baseline predictors. Additionally, we demonstrate that predictions from LLM-PP can be distilled to a compact regression model (LLM-Distill-PP), which surprisingly retains much of the performance of LLM-PP. This presents a cost-effective alternative for resource-intensive performance estimation. Specifically, for Neural Architecture Search (NAS), we introduce a *Hybrid-Search* algorithm (HS-NAS) employing LLM-Distill-PP for the initial search stages and reverting to the baseline predictor later. HS-NAS performs similarly to SoTA NAS, reducing search hours by approximately 50%, and in some cases, improving latency, GFLOPs, and model size.

1 Introduction

Large language models (LLMs) have diverse applications, encompassing both open-ended tasks (e.g., brainstorming and chat) and closed-ended tasks (e.g., summarization and question answering). This study explores a unique application of LLMs: constructing a performance predictor (LLM-PP) for a deep neural network (DNN) architecture. The predictor takes the DNN architecture description, typically hyperparameters (e.g., #layers, #attention heads), as input and predicts the performance (e.g., BLEU score) for a specific downstream task. The aim is to create a performance predictor with low prediction errors compared to training from scratch. The hypothesis is that LLMs possess a 'general understanding' of DNN architectures, derived from relevant training data like DNN research papers and GitHub repositories. The main objective of this work is to leverage this understanding to design accurate, efficient, and broadly applicable performance predictors, beneficial for tasks like neural architecture search (NAS).

How to design an accurate performance predictor (PP)? To answer this, we create *PP prompts* precisely specifying the task. These prompts include: (i) *role*: high-level description of the assigned LLM role, (ii) *instructions*: detailed task instructions (e.g., downstream task, architecture, performance/efficiency metric) for the LLM to follow, (iii) *hyperparameters*: definitions of architecture-specific hyperparameters, and (iv) *demonstrations*: supervised examples for the PP task with architecture descriptions and performance metrics (e.g., BLEU score). Using GPT-4 (OpenAI, 2023a) as our primary LLM and WMT datasets for machine translation (MT) tasks, we find that GPT-4 with our PP prompts (LLM-PP) predicts architecture performance with a mean absolute error achieving the state-of-the-art (SoTA) and a slightly lower rank correlation coefficient compared to previous SoTA weight-sharing supernet-based performance predictors (Wang et al., 2020; Jawahar et al., 2023b).

Using GPT-4 for LLM-PP entails utilizing the GPT-4 API to score each architecture, rendering LLM-PP prohibitively expensive for various use cases. One example is NAS, where PP evaluates approximately 3,000 candidate architectures for each constraint (e.g., latency $\leq$ 100ms) (Wang et al., 2020). As of August 2023, GPT-4 pricing is 0.03$ per $1K$ tokens [1]. Assuming PP prompts consume about one-third of $1K$ tokens, the estimated cost

[1] https://openai.com/pricing

is approximately ∼30$ for a single constraint on the target hardware. With varying constraint values (e.g., 100ms, 200ms), constraint types (e.g., latency, FLOPs, memory), and target hardware (e.g., Nvidia A100, Raspberry Pi), the cumulative cost can quickly become exorbitant (e.g., 1,800$).

How to design cost-effective PP? To answer this, we distill LLM-PP performance predictions into a tiny MLP model (LLM-Distill-PP) using architecture descriptions (e.g., hyperparameter lists) as input features. Surprisingly, LLM-Distill-PP can significantly maintain the performance of LLM-PP. Assuming LLM-Distill-PP needs only 3,000 examples, the estimated cost is approximately ∼30$ for a single downstream task, amortized across various constraint values, types, and target hardware.

Can LLM-Distill-PP speed up architecture search while preserving the efficiency and the quality of SoTA NAS? To answer this, we apply using LLM-Distill-PP as the PP to design efficient MT architectures via SoTA NAS methods like HAT (Wang et al., 2020). We introduce the *Hybrid-Search* algorithm (HS-NAS), where LLM-Distill-PP serves as the PP for the first 15 search iterations, and a weight-sharing supernet (SoTA performance predictor) takes over for the remaining 15 iterations. HS-NAS achieves roughly 50% faster search than SoTA NAS, maintaining or improving on the performance of architectures designed by SoTA NAS. In some cases, it also yields reduced latency (∼2%), FLOPs (∼1%), and model size (∼2%).

Main contributions: (1) We propose LLM-PP, leveraging few-shot prompting of LLM for accurate performance predictors, achieving SoTA mean absolute error. (2) We introduce LLM-Distill-PP, with a better amortized cost than LLM-PP, suitable for PP-heavy use cases. (3) HS-NAS, a search algorithm, reduces NAS search time by half compared to SoTA, identifying more efficient architectures by leveraging LLM-Distill-PP and SoTA performance estimators. (4) We provide prompts, training and evaluation data for LLM-Distill-PP models, and code with detailed reproducibility instructions.

2 Related Work

Performance Predictors. In NLP, a common approach to construct performance predictors is training a weight-sharing supernet model, jointly training various architectures by sharing weights with the largest model in the search space (Wang et al., 2020; Xu et al., 2022a; Jawahar et al., 2023a,b). During each training step, an architecture is randomly selected from the search space, and its corresponding weights are extracted from the largest model's weight matrices. These weights are then trained for the target task. Post-training, architecture performance is predicted by extracting the relevant weights and evaluating on the validation set. Key challenges in supernet training include weight co-adaptation (Bender et al., 2018; Zhao et al., 2021), capacity bottleneck (Jawahar et al., 2023b), and gradient conflict (Gong et al., 2021).

NAS for NLP. NAS is a general framework for designing efficient NLP architectures meeting user-defined constraints across various dimensions: (i) *architecture family* (encoder-only (Yin et al., 2021; Xu et al., 2022a, 2021, 2022b), decoder-only (Javaheripi et al., 2022), encoder-decoder (Wang et al., 2020; Jawahar et al., 2023a,b) without limiting to Transformers), (ii) *constraint types* (latency, FLOPs, model size), and (iii) *tasks* (task-agnostic pretraining (Xu et al., 2022a; Javaheripi et al., 2022; Jawahar et al., 2023b), task-specific training (Wang et al., 2020; Jawahar et al., 2023a)). The evolutionary search-based algorithm employs a performance predictor to identify high-quality architectures, utilizing real or predicted efficiency metrics to discard those not meeting specified constraints.

LLMs for NAS. GENIUS (Zheng et al., 2023), a recent search algorithm for image classification, uses LLMs to generate convolution-based architectures. However, it trains these candidates from scratch, incurring high practical costs. Contrasting with our approach, (i) GENIUS uses LLMs to generate architectures, while we use LLMs to predict their performance, (ii) the search cost for our work is upper bounded by SoTA NAS for MT (~ 5 NVIDIA V100 hours), much more efficient than GENIUS (~ 960 NVIDIA V100 hours), and (iii) we focus on Transformer-based encoder-decoder architectures for machine translation. For more on the synergy between LLMs and AutoML, see Tornede et al. (2023). Additional background on related topics such as LLMs and distillation can be found in A.1.

3 Performance Prediction Problem

Informally, the performance prediction problem entails providing a DNN architecture description (usually hyperparameters like #layers, #attention heads) to the predictor, which then outputs the performance (e.g., BLEU score) for a specified downstream task. An ideal predictor

should minimize prediction errors compared to the performance achieved through training from scratch. Formally, let T represent a downstream task, $\mathcal{A}_T$ its search space of architectures, and $\mathcal{Y}_T \subset \mathcal{R}$ the real space of performance scores. Define $\mathcal{D}_T$ as the data distribution over $\mathcal{A}_T \times \mathcal{Y}_T$. The performance predictor is denoted by $f_T : \mathcal{A}_T \rightarrow \mathcal{Y}_T$. The labeled test set $\mathcal{L}_T^{test} = \{(\mathbf{a}_i, p_i)\}_{i=1}^m \sim (\mathcal{D})_T^m$ comprises architecture, performance pairs drawn i.i.d. from $\mathcal{D}_T$. p_i is the performance obtained by training the architecture $\mathbf{a}_i$ from scratch to convergence on task T (known as 'training from scratch' (TFS) performance). The performance predictor's quality is assessed using two metrics: Mean Absolute Error (MAE) calculates the mean absolute difference between predictions and their corresponding TFS performances, formalized as $\sum_{(\mathbf{a}_i,p_i)\sim(\mathcal{D})_T} \frac{|f_T(\mathbf{a}_i)-p_i|}{|(\mathcal{D})_T|}$. Kendall rank correlation coefficient is another metric that computes the ranking correlation between a set of predictions and their corresponding TFS performances, formalized as Kendall-Tau$([f_T(\mathbf{a}_1), \ldots, f_T(\mathbf{a}_m)], [p_1, \ldots, p_m])$. Examples for these metrics are discussed in Section A.2. Recently, Jawahar et al. (2023b) emphasized the importance of both MAE and Kendall-Tau metrics in evaluating performance predictor quality. For instance, a predictor with a 38% better MAE and a 12% worse Kendall-Tau, compared to a base predictor, led NAS to find an architecture with a 4% BLEU improvement. Conversely, a predictor with a 5% worse MAE and a 6% higher Kendall-Tau resulted in a NAS architecture with a 0.1% BLEU improvement. Hence, better MAE and better Kendall-Tau are positively correlated with higher-quality architecture.

4 Baseline Performance Predictors

In NAS for NLP literature, the SoTA method for constructing performance predictors (f_T) involves training a weight-sharing supernet model on task T. Simply put, a weight-sharing supernet model is the largest model in the search space, capable of parameterizing all architectures via weight sharing. The parameters for a specific architecture are obtained by extracting the relevant rows and columns from the supernet model's weight matrix. Typically, the supernet is trained by iteratively sampling an architecture from the search space and training the extracted weights for that architecture. Formalizing the supernet's training objective: Denote the training data distribution as $\mathcal{X}_{train}$. Represent the training sample and label as x and y, where $x, y \sim \mathcal{X}_{train}$. a_{rand} is a uniformly sampled architecture from the search space $\mathcal{A}_T$. a_{large} and a_{small} denote the largest and smallest architectures in $\mathcal{A}_T$. The subnet with architecture a is denoted by s_a, parameterized by the supernet model weights W. The training objective of the supernet using sandwich sampling (Yu et al., 2020) is given by

$$\min_W \mathbb{E}_{x,y\sim\mathcal{X}_{train}}[\mathbb{E}_{a_{rand}\sim\mathcal{A}}[\mathcal{L}(s_{a_{rand}}(x;W), y)] + \mathcal{L}(s_{a_{large}}(x;W), y) + \mathcal{L}(s_{a_{small}}(x;W), y)].$$

Hardware-aware Transformers (Wang et al., 2020) employs single-path one-shot (SPOS) optimization (Guo et al., 2020), focusing on optimizing only a_{rand} at each training step. Mixture-of-Supernets (Jawahar et al., 2023b) (MoS) utilizes mixture-of-experts (MoE) (Fedus et al., 2022) to enhance the supernet's capacity, with the router specializing weights for each architecture. MoS comes in two variants: layer-wise MoS and neuron-wise MoS, differing in the degree of freedom for weight generation. Both variants of MoS employ sandwich sampling for supernet training.

5 LLM Performance Predictor (LLM-PP)

LLM demonstrates a "general understanding" of DNN architectures, likely acquired through training on relevant data sources like research papers and GitHub repositories. Testing these architecture understanding capabilities involves prompting LLM to generate hyperparameter definitions and design principles for architecture search (Zheng et al., 2023). These LLM capabilities contribute to effective performance prediction by aiding the mapping of DNN architectures to their performances.

To this end, we propose the LLM-based Performance Predictor (LLM-PP), which involves prompting an LLM to generate performance predictions for DNN architectures. The prompts, referred to as *PP prompts*, must be meticulously designed to precisely convey the performance prediction task to the LLM. Illustrated in Figure 1, PP prompts break down the task into four main components: *role*, *instructions*, *hyperparameters*, and *demonstrations*, followed by the test architecture. The *role* specifies the LLM's role, describing the downstream task (e.g., machine translation) and the performance metric (e.g., BLEU). The *instructions* provide five detailed instructions covering the downstream task,

Role

You are a performance estimator for machine translation task, where you will estimate the BLEU score for the test architecture.

Instruction

You should follow these instructions:
1. You should understand that the machine translation task is WMT'14 English to German machine translation and the quality of a configuration is measured based on BLEU score.
2. Some examples for WMT'14 English to German machine translation are as follows:
Example 1:
Input: Resumption of the session
Output: Wiederaufnahme der Sitzungsperiode
...
Example n_{task}:
Input: Please rise, then, for this minute' s silence.
Output: Ich bitte Sie, sich zu einer Schweigeminute zu erheben.
3. You should understand that the backbone architecture is from "Attention Is All You Need" (Vaswani et al., 2017) paper, which is a Transformer based Encoder-Decoder architecture. We use the same hyperparameters and optimization algorithms.
4. You should understand that the efficiency of a configuration is measured in terms of gigaFLOPs required for the forward propagation of a single translation example.
5. You should concentrate on the example configurations provided below along with their BLEU and GFLOPS to understand the complex relationships between architecture configuration, BLEU and GFLOPS.

Hyperparameters

Hyperparameter definition:
'encoder-embed-dim-subtransformer' corresponds to encoder embedding dimension
'encoder-layer-num-subtransformer' corresponds to number of encoder layers
'encoder-ffn-embed-dim-all-subtransformer' correspond to embedding dimension of each FFN layer in encoder
...

Demonstrations

Example 1:
encoder-embed-dim-subtransformer: 512
encoder-layer-num-subtransformer: 6
encoder-ffn-embed-dim-all-subtransformer: [1024, 1024, 2048, 2048, 2048, 1024]
...
BLEU: 24.30
GFLOPS: 2.7
...
Example n_{arch}:
...

Test

Test Architecture:
encoder-embed-dim-subtransformer: 640
encoder-layer-num-subtransformer: 6
encoder-ffn-embed-dim-all-subtransformer: [2048, 1024, 1024, 1024, 2048, 1024]
...
BLEU:

Figure 1: Prompt template to prompt LLM to generate performance predictions for WMT'14 EN-DE task. The expanded version of the prompt template can be seen in Appendix A.3.

DNN architecture, and model efficiency metrics. The first two focus on the task specifics, specifying the task type (e.g., machine translation), dataset (e.g., WMT'14 En-De), performance metric (e.g., BLEU), and inputs/outputs (e.g., source/target language) for n_{task} examples from the dataset. The third instruction details the DNN architecture, including backbone (e.g., Transformer), type (e.g., encoder-decoder), and a reference to the original DNN paper. The fourth instruction outlines efficiency metrics details (e.g., GFLOPs), included in the demonstrations. The final instruction directs the LLM to consider complex relationships between architecture configuration, performance, and efficiency metric. The third component, *hyperparameters*, defines architecture-specific hyperparameters. *Demonstrations* is the final component containing n_{arch} supervised examples, each representing an architecture from the search space with hyperparameter values, efficiency score, and TFS performance score. The design process of the LLM-PP prompt is discussed in A.4.

5.1 Evaluation Setup

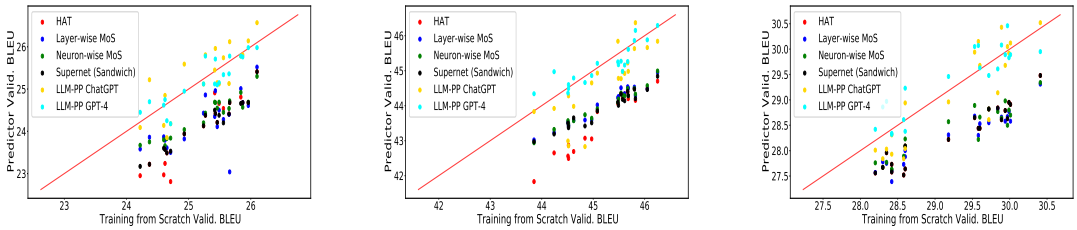
Downstream tasks. We utilize established research (Wang et al., 2020; Jawahar et al., 2023a,b) and opt for popular machine translation (MT) benchmarks: WMT'14 En-De, WMT'14 En-Fr, and WMT'19 En-De. Detailed statistics of these benchmarks are available in A.6.1. Our chosen performance metric is BLEU (Papineni et al., 2002).

DNN architecture. We adopt the Transformer-based Encoder-Decoder architecture (Vaswani et al., 2017). The implementation, training settings, and search space ($\mathcal{A}$) mirror Wang et al. (2020), detailed in A.6.2. Our evaluation dataset (TFS-Eval) is sourced from Jawahar et al. (2023b), featuring 30 architectures with their TFS performance scores for each WMT dataset. FLOPs, latency, and model size computations for architectures are done using the implementation from Wang et al. (2020).

Performance predictors. Baseline performance predictors include: (i) HAT (Wang et al., 2020), (ii) Supernet (Sandwich) (Jawahar et al., 2023b) (HAT, with sandwich sampling instead of SPOS), (iii) Layer-wise MoS (Jawahar et al., 2023b), and (iv) Neuron-wise MoS (Jawahar et al., 2023b). We build three LLM-PP variants, utilizing Mistral (Jiang et al., 2023) (Mistral-7B-Instruct-v0.1), ChatGPT (OpenAI, 2023b) (GPT-3.5-turbo, June version), and GPT-4 (OpenAI, 2023a) (June version). For PP prompts, we randomly sample: (i) 5 examples ($n_{task} = 5$) from the downstream task for the second instruction and (ii) 10 examples ($n_{task} = 10$) from TFS-eval for the demonstrations component. The remaining 20 examples from TFS-eval will be used for reporting the predictor quality. For all predictors, we repeat the experiments with three different seeds and report the average MAE and Kendall-Tau between the predictor performance and the TFS performance.

5.2 Results

LLM-PP predictions closely align with TFS performance scores compared to the baselines. Figure 2 illustrates the TFS versus performance predictor validation BLEU for different WMT benchmarks. The diagonal line (red line) represents the perfect predictor, where the predicted performance exactly matches the TFS score. The predictions from the supernet-based predictors (i.e., all non-LLM-based ones) are consistently underestimates of the TFS performance for all architectures across three benchmarks. In contrast, LLM-PP predictions



(a) WMT'14 En-De (b) WMT'14 En-Fr (c) WMT'19 En-De

Figure 2: Training from scratch validation BLEU vs. performance predictor validation BLEU for WMT benchmarks. Performance scores from the optimal predictor should lie on the diagonal (red line).

Dataset	WMT'14 En-De		WMT'14 En-Fr		WMT'19 En-De		Average	
Performance Predictor	**MAE**	**Kendall**	**MAE**	**Kendall**	**MAE**	**Kendall**	**MAE (↓)**	**Kendall (↑)**
Baseline								
HAT	1.14	0.71	1.59	0.79	0.91	0.72	1.21	0.74
Supernet (Sandwich)	1.05	**0.81**	1.27	0.78	0.91	0.72	1.08	0.77
Layer-wise MoS	0.97	0.56	1.16	0.79	0.96	**0.74**	1.03	0.70
Neuron-wise MoS	0.87	0.79	1.18	**0.87**	0.87	0.67	0.97	**0.78**
LLM-PP								
Mistral	0.73	0.22	0.60	0.34	0.92	0.18	0.75	0.25
ChatGPT	0.42	0.52	0.82	0.61	0.72	0.56	0.65	0.56
GPT-4	0.28	0.65	**0.28**	0.75	0.32	0.65	**0.29**	0.68
LLM-PP GPT-4 Ablation								
Demonstraions only	0.31	0.52	0.30	0.66	0.34	0.61	0.32	0.60
+ Role + Hyp.	0.27	0.53	0.32	0.71	0.32	0.67	0.30	0.64
+ First instruction	0.26	0.60	0.34	0.68	0.34	0.58	0.31	0.62
+ Second instruction	0.27	0.60	0.31	0.72	0.35	0.66	0.31	0.66
+ Third instruction	0.31	0.50	0.33	0.73	**0.29**	0.67	0.31	0.63
+ Fourth instruction	0.25	0.63	0.32	0.65	0.33	0.71	0.30	0.66
+ Fifth instruction	0.28	0.65	**0.28**	0.75	0.32	0.65	**0.29**	0.68
LLM-Distill-PP								
ChatGPT	0.32	0.6	1.01	0.79	0.95	0.65	0.76	0.68
GPT-4	**0.22**	0.64	0.34	0.76	0.38	0.68	0.31	0.69

Table 1: Average MAE and Kendall-Tau between the performance predictor performance and the TFS performance, across three different seeds.

are largely closer to the diagonal line, showcasing the high accuracy of LLM-PP.

LLM-PP achieves SoTA MAE, slightly trailing baselines in Kendall-Tau. Table 1 displays the MAE and Kendall-Tau of baseline and LLM-PP predictors. Neuron-wise MoS stands out as the best baseline on average across datasets, boasting the lowest MAE and highest Kendall-Tau score. LLM-PP Mistral outperforms supernet-based baselines in MAE for WMT'14 En-De and WMT'14 En-Fr tasks. LLM-PP ChatGPT and LLM-PP GPT-4 surpass Neuron-wise MoS in MAE, with LLM-PP GPT-4 achieving the SoTA MAE score. However, LLM-PP slightly lags behind baselines in Kendall-Tau. In A.5, we examine the histogram of distances between items in discordant pairs in the gold ranking for Neuron-wise MoS and LLM GPT-4. Discordant pairs of LLM-PP mostly cluster around the low gold ranking distances region, similar to Neuron-wise MoS, which shouldn't significantly impact PP use cases (as observed in Section 7.1). The resulting CDF of gold ranking distances for discordant pairs for LLM-PP GPT-4 and Neuron-wise MoS are very similar. These results indicate that PP prompts can effectively design accurate performance predictors. Within LLM-PP, GPT-4 outperforms ChatGPT on both metrics across datasets.

LLM-PP benefits from all the components of PP prompts. The last major row in Table 1 displays the performance when ablating different components of PP prompts. LLM-PP's overall superior performance is attributed to having all PP prompt components together. Surprisingly, LLM-PP outperforms baselines in MAE even without any instructions (Demonstration only), showcasing the LLM's remarkable ability to grasp the performance prediction task based solely on demonstrations. While the MAE performance of different ablation variants is largely similar, there are differences in Kendall-Tau performance across variants. The second instruction (introducing downstream task-specific examples) and the fourth instruction (describing the efficiency metric) play crucial roles in achieving high Kendall-Tau for LLM-PP.

LLM-PP exceeds non-supernet baselines (White et al., 2022), with LLM-PP GPT-4 achieving a high

Kendall Tau, as discussed in A.8.1. LLM-PP attains SoTA MAE and SoTA Kendall-Tau scores for low-resource/indigenous languages (Ebrahimi et al., 2023) (see A.8.2) and uncommon evaluation metric (COMET (Rei et al., 2022), see A.8.3). LLM-PP provides fairly robust performance predictions (see A.8.4). While LLM-PP excels in performance prediction quality, its cost scales linearly with the number of predictions. This cost can become prohibitive, especially for PP-heavy applications like NAS, where the number of predictions can reach several thousand.

6 Distillation of LLM-PP

To illustrate the cost, let's consider the example of NAS run by HAT (Wang et al., 2020) for a latency constraint on a given hardware, involving the evaluation of approximately 3,000 candidate architectures. As of August 2023, the pricing for GPT-4 is 0.03$ per 1K tokens. Assuming PP prompts consume about one-third of 1K tokens, the estimated cost per constraint on a given hardware would be around 30$ ($\frac{0.03*3000}{3}$). The total cost depends on the number of constraint types (e.g., latency, memory, FLOPs), values (e.g., 100ms, 200ms), and hardware options (e.g., Nvidia A100, Raspberry Pi). For instance, with three constraint types, five values for each constraint, and four target hardwares, the estimated cost could soar to approximately $1,800$\$ ($\frac{0.03*3000*3*5*4}{3}$) per downstream task. To address this cost challenge, we propose LLM-Distill-PP, a cost-effective alternative trained on distilled outputs of LLM-PP. LLM-Distill-PP, a MLP based regressor, is trained using a distillation dataset for the PP task. This dataset is created by sampling architectures from the search space and recording the downstream task performance predicted by LLM-PP. LLM-Distill-PP is trained using architecture-specific hyperparameters as features and the distilled output as labels. Once trained, LLM-Distill-PP can predict the performance of unseen architectures for the given downstream task. If the number of distillation examples is small (e.g., $3,000$), the estimated cost to query LLM-PP will be approximately 30$ ($\frac{0.03*3000}{3}$). This one-time cost of LLM-Distill-PP is amortized across different constraint types, values, and hardwares (e.g., 60 search runs), leading to a substantial 98.3% reduction in cost (from $1,800$\$ to 30$). LLM-Distill-PP achieves a superior efficiency-accuracy trade-off, offering comparable accuracy to LLM-PP but with significantly faster prediction times (0.01s vs. 11.9s), as detailed in A.10.1.

Setup. LLM-Distill-PP's architecture encoding details can be found in A.6.3. The hyperparameters of its regression model, borrowed from HAT's latency predictor, include 3 hidden layers, 400 as the hidden dimension, 128 as the batch size, 1e-5 as the learning rate, and 5000 as the number of training steps. Distillation from LLM-PP uses only 3000 architecture examples for each downstream task.

Results. LLM-Distill-PP's results are summarized in the third major row of Table 1. Despite its simple model design, LLM-Distill-PP performs similarly or better than LLM-PP for both ChatGPT and GPT-4. In the case of ChatGPT, LLM-Distill-PP exhibits an average improvement of roughly 17% in both MAE and Kendall-Tau over LLM-PP. For GPT-4, LLM-Distill-PP has a 7% lower average MAE compared to LLM-PP while maintaining similar Kendall-Tau. Notably, LLM-Distill-PP achieves the SoTA MAE for the WMT'14 En-De task, outperforming LLM-PP by 20%. Two main factors contribute to this result. First, the smaller size of LLM-Distill-PP (a linear regression model with only 486K parameters) reduces the likelihood of overfitting compared to LLM-PP (an LLM with several billion parameters), resulting in better performance. Second, LLM-Distill-PP is a specialist model with parameters trained specifically for the performance prediction task using a few thousand examples. In contrast, LLM-PP is a generalist model that performs in-context learning with PP prompts and 10 demonstrations.

7 LLM-Distill-PP for Architecture Search

Given LLM-Distill-PP's ability to achieve high-performance prediction quality in a cost-effective manner, we explore its application in a real-world task: NAS. In NAS, performance predictors typically rank candidate architectures to identify high-performing ones. As discussed in Section 2, existing NAS research in NLP primarily uses weight-sharing supernets as performance predictors. Therefore, we address the research question: *Can LLM-Distill-PP accelerate architecture search while maintaining the efficiency and quality of SoTA NAS?* To answer this question, we introduce the Hybrid-Search algorithm for NAS (HS-NAS). The core idea of HS-NAS is to employ LLM-Distill-PP for a subset of search iterations, utilizing the supernet for the remaining iterations.

Search Algorithm	BLEU (↑)	Latency (ms) (↓)	GFLOPs (↓)	Model Size (M) (↓)	Search Hours (↓)
WMT'14 En-De					
HAT	27.9	102.0	3.0	64.4	1.09
Layer-wise MoS	27.8	100.4	3.08	64.4	1.45
Neuron-wise MoS	**28.0**	**99.0**	3.26	72.2	1.39
HS-NAS (GPT-4, HAT, 1, 15)	27.9	99.7	**2.96**	**63.1**	**0.56**
WMT'14 En-Fr					
HAT	40.8	**96.4**	2.61	**63.8**	6.33
Layer-wise MoS	40.5	99.4	2.96	70.5	6.81
Neuron-wise MoS	**40.9**	97.6	3.13	70.5	7.03
HS-NAS (GPT-4, HAT, 1, 15)	40.7	98.2	**2.54**	**63.8**	**3.15**
WMT'19 En-De					
HAT	44.7	100.8	3	73.06	1.11
Layer-wise MoS	**44.9**	96.8	3.26	82.95	1.13
Neuron-wise MoS	**44.9**	122.4	3.34	82.95	1.21
HS-NAS (GPT-4, HAT, 1, 15)	44.4	**70.0**	**2.51**	**66.36**	**0.46**

Table 2: HS-NAS versus SoTA NAS on three MT benchmarks for latency constraint of $100ms$ - Test BLEU, latency in milliseconds, GFLOPs, model size in millions, and search hours.

This approach is applied to the evolutionary search algorithm proposed in HAT.

Algorithm 1 Hybrid-Search algorithm for Neural Architecture Search (HS-NAS). Changes to HAT's search algorithm are in red color. The expanded algorithm can be found in A.9.

Input: LLM-Distill-PP model: `llm-distill-pp`, Weight-sharing supernet: `supernet`, Latency predictor: `latency-predictor`, #Search iterations: `num-iterations`, Population size: `population-size`, Latency constraint: `latency-constraint`, LLM-Distill-PP Start Iteration: `llm-start-iteration`, LLM-Distill-PP End Iteration: `llm-end-iteration`, ...
Output: `best-architecture`

```
popu ← population-size rand. samples from search space // create init. population
for iter ← 1 to num-iterations do
    // gen. parents by picking top cand. arch.
    if llm-start-iteration < iter < llm-end-iteration then
        parents ← top 'num-parents' arch. from popu by llm-distill-pp
    else
        parents ← top 'num-parents' arch. from popu by supernet
    mut-popu = HAT's mutation logic
    cross-popu = HAT's crossover logic
    popu = parents ∪ mut-pop ∪ cross-pop
return top arch. from popu
```

LLM-Distill-PP will be used as performance predictor for all the search iterations in between `llm-start-iteration` and `llm-end-iteration`. In rest of the iterations, supernet will be used as performance predictor. When `llm-start-iteration=1` and `llm-end-iteration=num-iterations`, HS-NAS uses LLM-Distill-PP as performance predictor for all the search iterations. HS-NAS comes with four arguments: (`llm-distill-pp`, `supernet`, `llm-start-iteration`, `llm-end-iteration`). For all our search experiments, we use LLM-Distill-PP GPT-4 as `llm-distill-pp` due to its superior performance over the ChatGPT counterpart (see the third major row in Table 1). We use the `latency-predictor` and `supernet` from HAT. Other details of the setup (e.g., efficiency metric for search (search hours), and architecture (latency, GFLOPs, model size)) can be seen in A.7.

7.1 Results

Varying benchmarks. HS-NAS shows comparable performance to the SoTA across benchmarks, achieving approximately a 50% reduction in search hours. In some cases, it even enhances latency, GFLOPs, and model size, as illustrated in Table 2. This pattern highlights the effectiveness of using LLMs as good initializers for architecture search.

Varying latency constraints. The trend observed in HS-NAS remains consistent across different latency constraints. Table 3 presents a comparison of the HS-NAS configuration (GPT-4, HAT, 1, 15) against the SoTA NAS for different latency constraints: 100ms, 150ms, and 200ms. Alongside a 50% reduction in search hours, HS-NAS attains comparable or improved GFLOPs and maintains the same model size compared to SoTA NAS.

Varying start and end iteration pairs. Among different start and end iteration pairs, HS-NAS utilizing LLM-Distill-PP (GPT-4) for the initial 50%

Search Algorithm	BLEU (↑)	Latency (ms) (↓)	GFLOPs (↓)	Model Size (M) (↓)	Search Hours (↓)
100ms					
HAT	40.8	**96.4**	2.61	**63.8**	6.33
Layer-wise MoS	40.5	99.4	2.96	70.5	6.81
Neuron-wise MoS	**40.9**	97.6	3.13	70.5	7.03
HS-NAS (GPT-4, HAT, 1, 15)	40.7	98.2	**2.54**	**63.8**	**3.15**
150ms					
HAT	41.3	176.4	**3.31**	**74.3**	7.33
Layer-wise MoS	**41.4**	**158.7**	4.3	92.8	8.39
Neuron-wise MoS	**41.4**	200.2	4.26	92.8	8.35
HS-NAS (GPT-4, HAT, 1, 15)	**41.4**	172.6	**3.31**	**74.3**	**3.69**
200ms					
HAT	41.5	187.5	**3.7**	**79.5**	7.8
Layer-wise MoS	41.4	205.6	4.49	99.4	8.63
Neuron-wise MoS	41.6	**184.1**	4.53	99.4	8.77
HS-NAS (GPT-4, HAT, 1, 15)	**42.0**	187.8	**3.7**	**79.5**	**3.88**

Table 3: HS-NAS versus SoTA NAS on WMT'14 En-Fr for different latency constraints - Test BLEU, latency in milliseconds, GFLOPs, model size in millions, and search hours.

Search Algorithm	BLEU (↑)	Latency (ms) (↓)	GFLOPs (↓)	Model Size (M) (↓)	Search Hours (↓)
HAT	27.9	102.0	3.0	64.4	1.09
HS-NAS (GPT-4, HAT, 1, 30)	27.5	99.3	3.34	72.2	0.04
HS-NAS (GPT-4, HAT, 1, 5)	27.4	100.4	2.96	63.1	0.97
HS-NAS (GPT-4, HAT, 25, 30)	28.0	119.1	3.18	70.9	0.95
HS-NAS (GPT-4, HAT, 1, 15)	**27.9**	**99.7**	**2.96**	**63.1**	**0.56**
HS-NAS (GPT-4, HAT, 16, 30)	27.6	101.7	3.34	72.2	0.75
HS-NAS (GPT-4, HAT, 1, 25)	27.7	98.9	3.01	63.1	0.23

Table 4: HS-NAS versus HAT on WMT'14 En-De for latency constraint: $100ms$ - Test BLEU, latency in milliseconds, GFLOPs, model size in millions, and search hours.

of iterations and HAT supernet for the remainder performs comparably or outperforms HAT across all metrics. Table 4 presents the results of HS-NAS for various start and end iteration pairs. Utilizing LLM-Distill-PP for the entire search yields lower performance, indicating that a marginal degradation in Kendall-Tau hinders LLM-Distill-PP's effectiveness in handling the complete search. These trends underscore the utility of a predictor with SoTA MAE scores for the initial search, while a predictor with SoTA Kendall-Tau is valuable for the later stages of the search.

Varying initialization seeds, FLOPs constraints, underlying supernet. HS-NAS exhibits resilience to initialization effects stemming from different seeds, yielding largely consistent results across metrics. Further details are provided in A.10.2. HS-NAS performs comparably to HAT under varying FLOPs constraints, showcasing a minimum 16% reduction in search hours, a 1.2% improvement in latency, consistent GFLOPs, and identical model sizes. These trends persist consistently across benchmarks, as outlined in A.10.3. The superiority of HS-NAS remains robust across different underlying supernets, as elucidated in A.10.4.

Trivially constructed efficient adaptations of SoTA. Search hours can be trivially reduced in several ways: halving the total number of search iterations and/or using distilled SoTA predictor instead of using supernet predictor directly. While these adaptations lead to a big drop in BLEU performance (1.8% for HAT (`num-iter.`=15)) or a big increase in latency and GFLOPs (9.7% and 32% respectively for Distilled HAT (`num-iter.`=15)), HS-NAS dominates these adaptions in search hour reductions, while maintaining SoTA performance and not degrading on any footprint metric, as detailed in A.10.5. Putting all the observed trends of HS-NAS together, we find that the generality of HS-NAS extends to constraint types (latency, FLOPs), constraint values (different latencies, different FLOPs), different tasks (MT benchmarks), and underlying supernet (HAT, Neuron-wise MoS), while being robust to initialization effects.

8 Conclusion

This work shows that LLMs can be employed to create accurate and cost-effective performance predictors, providing insights into enhancing NAS. This contribution adds to the expanding field of LLMs in NAS, suggesting future research directions in adapting LLMs for both candidate architecture generation and joint performance prediction.

9 Limitations

- **Expanding task domains.** Our evaluation setup, centered on machine translation benchmarks, aligns with existing NAS for NLP literature (Wang et al., 2020; Jawahar et al., 2023a,b), primarily focusing on machine translation tasks. Investigating the applicability of the LLM-PP framework to diverse NLP tasks (e.g., summarization, language modeling) and non-NLP domains (e.g., speech recognition, computer vision) stands as a crucial avenue for future exploration.

- **Exploring diverse architectures.** This work focused on classic Transformer architectures as outlined by Vaswani et al., aligning with NAS for NLP literature. While our primary investigation remained focused on these architectures, examining other architecture types (e.g., convolution embedding based (Salesky et al., 2023)) stands as a pertinent future direction.

10 Acknowledgments

We used ChatGPT for rephrasing and grammar checking of the paper.

References

Rohan Anil, Andrew M. Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, Eric Chu, Jonathan H. Clark, Laurent El Shafey, Yanping Huang, Kathy Meier-Hellstern, Gaurav Mishra, Erica Moreira, Mark Omernick, Kevin Robinson, Sebastian Ruder, Yi Tay, Kefan Xiao, Yuanzhong Xu, Yujing Zhang, Gustavo Hernandez Abrego, Junwhan Ahn, Jacob Austin, Paul Barham, Jan Botha, James Bradbury, Siddhartha Brahma, Kevin Brooks, Michele Catasta, Yong Cheng, Colin Cherry, Christopher A. Choquette-Choo, Aakanksha Chowdhery, Clément Crepy, Shachi Dave, Mostafa Dehghani, Sunipa Dev, Jacob Devlin, Mark Díaz, Nan Du, Ethan Dyer, Vlad Feinberg, Fangxiaoyu Feng, Vlad Fienber, Markus Freitag, Xavier Garcia, Sebastian Gehrmann, Lucas Gonzalez, Guy Gur-Ari, Steven Hand, Hadi Hashemi, Le Hou, Joshua Howland, Andrea Hu, Jeffrey Hui, Jeremy Hurwitz, Michael Isard, Abe Ittycheriah, Matthew Jagielski, Wenhao Jia, Kathleen Kenealy, Maxim Krikun, Sneha Kudugunta, Chang Lan, Katherine Lee, Benjamin Lee, Eric Li, Music Li, Wei Li, YaGuang Li, Jian Li, Hyeontaek Lim, Hanzhao Lin, Zhongtao Liu, Frederick Liu, Marcello Maggioni, Aroma Mahendru, Joshua Maynez, Vedant Misra, Maysam Moussalem, Zachary Nado, John Nham, Eric Ni, Andrew Nystrom, Alicia Parrish, Marie Pellat, Martin Polacek, Alex Polozov, Reiner Pope, Siyuan Qiao, Emily Reif, Bryan Richter, Parker Riley, Alex Castro Ros, Aurko Roy, Brennan Saeta, Rajkumar Samuel, Renee Shelby, Ambrose Slone, Daniel Smilkov, David R. So, Daniel Sohn, Simon Tokumine, Dasha Valter, Vijay Vasudevan, Kiran Vodrahalli, Xuezhi Wang, Pidong Wang, Zirui Wang, Tao Wang, John Wieting, Yuhuai Wu, Kelvin Xu, Yunhan Xu, Linting Xue, Pengcheng Yin, Jiahui Yu, Qiao Zhang, Steven Zheng, Ce Zheng, Weikang Zhou, Denny Zhou, Slav Petrov, and Yonghui Wu. 2023. Palm 2 technical report.

Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. 2018. Understanding and simplifying one-shot architecture search. In *International conference on machine learning*, pages 550–559. PMLR.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality.

Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, Barret Zoph, Liam Fedus, Maarten P Bosma, Zongwei Zhou, Tao Wang, Emma Wang, Kellie Webster, Marie Pellat, Kevin Robinson, Kathleen Meier-Hellstern, Toju Duke, Lucas Dixon, Kun Zhang, Quoc Le, Yonghui Wu, Zhifeng Chen, and Claire Cui. 2022. GLaM: Efficient scaling of language models with mixture-of-experts. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 5547–5569. PMLR.

Abteen Ebrahimi, Manuel Mager, Shruti Rijhwani, Enora Rice, Arturo Oncevay, Claudia Baltazar, María Cortés, Cynthia Montaño, John E. Ortega, Rolando Coto-solano, Hilaria Cruz, Alexis Palmer, and Katharina Kann. 2023. Findings of the AmericasNLP 2023 shared task on machine translation into indigenous languages. In *Proceedings of the Workshop on Natural Language Processing for Indigenous Languages of the Americas (AmericasNLP)*, pages 206–

219, Toronto, Canada. Association for Computational Linguistics.

William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39.

Chengyue Gong, Dilin Wang, Meng Li, Xinlei Chen, Zhicheng Yan, Yuandong Tian, Vikas Chandra, et al. 2021. Nasvit: Neural architecture search for efficient vision transformers with gradient conflict aware supernet training. In *International Conference on Learning Representations*.

Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. 2020. Single path one-shot neural architecture search with uniform sampling. In *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVI*, page 544–560, Berlin, Heidelberg. Springer-Verlag.

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

Ganesh Jawahar, Subhabrata Mukherjee, Xiaodong Liu, Young Jin Kim, Muhammad Abdul-Mageed, Laks Lakshmanan, V.S., Ahmed Hassan Awadallah, Sebastien Bubeck, and Jianfeng Gao. 2023a. AutoMoE: Heterogeneous mixture-of-experts with adaptive computation for efficient neural machine translation. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 9116–9132, Toronto, Canada. Association for Computational Linguistics.

Ganesh Jawahar, Haichuan Yang, Yunyang Xiong, Zechun Liu, Dilin Wang, Fei Sun, Meng Li, Aasish Pappu, Barlas Oguz, Muhammad Abdul-Mageed, Laks V. S. Lakshmanan, Raghuraman Krishnamoorthi, and Vikas Chandra. 2023b. Mixture-of-supernets: Improving weight-sharing supernet training with architecture-routed mixture-of-experts.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b.

Yoon Kim and Alexander M. Rush. 2016. Sequence-level knowledge distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1317–1327, Austin, Texas. Association for Computational Linguistics.

Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. 2023. Orca: Progressive learning from complex explanation traces of gpt-4.

OpenAI. 2023a. Gpt-4 technical report.

OpenAI. 2023b. Introducing chatgpt.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Gray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Ricardo Rei, José G. C. de Souza, Duarte Alves, Chrysoula Zerva, Ana C Farinha, Taisiya Glushkova, Alon Lavie, Luisa Coheur, and André F. T. Martins. 2022. COMET-22: Unbabel-IST 2022 submission for the metrics shared task. In *Proceedings of the Seventh Conference on Machine Translation (WMT)*, pages 578–585, Abu Dhabi, United Arab Emirates (Hybrid). Association for Computational Linguistics.

Elizabeth Salesky, Neha Verma, Philipp Koehn, and Matt Post. 2023. Multilingual pixel representations for translation and effective cross-lingual transfer. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13845–13861, Singapore. Association for Computational Linguistics.

Karan Singhal, Shekoofeh Azizi, Tao Tu, S Sara Mahdavi, Jason Wei, Hyung Won Chung, Nathan Scales, Ajay Tanwani, Heather Cole-Lewis, Stephen Pfohl, et al. 2022. Large language models encode clinical knowledge. *arXiv preprint arXiv:2212.13138*.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca.

Alexander Tornede, Difan Deng, Theresa Eimer, Joseph Giovanelli, Aditya Mohan, Tim Ruhkopf, Sarah Segel, Daphne Theodorakopoulos, Tanja Tornede, Henning Wachsmuth, and Marius Lindauer. 2023. Automl in the age of large language models: Current challenges, future opportunities and risks.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023a. Llama: Open and efficient foundation language models.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023b. Llama 2: Open foundation and fine-tuned chat models.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. 2020. HAT: Hardware-aware transformers for efficient natural language processing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7675–7688, Online. Association for Computational Linguistics.

Colin White, Mikhail Khodak, Renbo Tu, Shital Shah, Sébastien Bubeck, and Debadeepta Dey. 2022. A deeper look at zero-cost proxies for lightweight nas. In *ICLR Blog Track*. Https://iclr-blog-track.github.io/2022/03/25/zero-cost-proxies/.

Minghao Wu, Abdul Waheed, Chiyu Zhang, Muhammad Abdul-Mageed, and Alham Fikri Aji. 2023. Lamini-lm: A diverse herd of distilled models from large-scale instructions. *CoRR*, abs/2304.14402.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2022. Smoothquant: Accurate and efficient post-training quantization for large language models. *arXiv*.

Dongkuan Xu, Subhabrata Mukherjee, Xiaodong Liu, Debadeepta Dey, Wenhui Wang, Xiang Zhang, Ahmed Hassan Awadallah, and Jianfeng Gao. 2022a. Few-shot task-agnostic neural architecture search for distilling large language models. In *Advances in Neural Information Processing Systems*.

Jin Xu, Xu Tan, Renqian Luo, Kaitao Song, Jian Li, Tao Qin, and Tie-Yan Liu. 2021. Nas-bert: Task-agnostic and adaptive-size bert compression with neural architecture search. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, page 1933–1943, New York, NY, USA. Association for Computing Machinery.

Jin Xu, Xu Tan, Kaitao Song, Renqian Luo, Yichong Leng, Tao Qin, Tie-Yan Liu, and Jian Li. 2022b. Analyzing and mitigating interference in neural architecture search. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 24646–24662. PMLR.

Yichun Yin, Cheng Chen, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. 2021. AutoTinyBERT: Automatic hyper-parameter optimization for efficient pre-trained language models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 5146–5157, Online. Association for Computational Linguistics.

Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas Huang, Xiaodan Song, Ruoming Pang, and Quoc Le. 2020. Bignas: Scaling up neural architecture search with big single-stage models. In *Computer Vision – ECCV 2020*, pages 702–717, Cham. Springer International Publishing.

Yiyang Zhao, Linnan Wang, Yuandong Tian, Rodrigo Fonseca, and Tian Guo. 2021. Few-shot neural architecture search. In *International Conference on Machine Learning*, pages 12707–12718. PMLR.

Mingkai Zheng, Xiu Su, Shan You, Fei Wang, Chen Qian, Chang Xu, and Samuel Albanie. 2023. Can gpt-4 perform neural architecture search?

A Appendix

A.1 Related Work - Extended

LLMs. LLMs can be classified into two categories based on their training methods: foundation and instruction-tuned LLMs. Foundation LLMs, which includes GPT-3 (Brown et al., 2020), GLaM (Du et al., 2022), LLaMA-1 (Touvron et al., 2023a), undergo language model training on unannotated corpus from the web. These LLMs typically encode a lot of useful knowledge in their parameters and can be used for a downstream task by either fine-tuning or zero/few-shot prompting. Instruction-tuned LLMs are usually foundation LLMs that undergo instruction-tuning, where LLMs are explicitly fine-tuned to follow user defined instructions well. Such LLMs include InstructGPT (Ouyang et al., 2022), ChatGPT (OpenAI, 2023b), GPT-4 (OpenAI, 2023a), LLaMA-2 (Touvron et al., 2023b), and PaLM-2 (Anil et al., 2023). In practice, instruction-tuned LLMs can follow a wide range of

user's instructions, even those that are outside the instruction tuning data distribution (Ouyang et al., 2022). However, depending on the task, instruction-tuned LLMs are prone to generating content that are factually incorrect, hallucinated, ignores user's instruction, toxic, and so on (Ouyang et al., 2022). These challenges make the current SoTA LLMs unreliable for critical applications such as medical diagnosis (Singhal et al., 2022).

Distilling LLMs. Distilling the generations from LLMs to smaller student models has become commonplace in NLP these days (Taori et al., 2023; Chiang et al., 2023; Wu et al., 2023; Mukherjee et al., 2023). The key motivations for such efforts include: (i) *cost reduction*: most LLMs are either behind a paywall or require high-end GPUs (e.g., NVIDIA A100) with high GPU memory (e.g., 80GB) to use, (ii) *latency reduction*: most LLMs are too slow even on high-end hardware (e.g., OPT-175B takes 4s for decoding 16 sequences of length 1024 on 8 NVIDIA A100 80GB GPUs (Xiao et al., 2022)), and (iii) *customization*: most LLMs are general purpose and are difficult to finetune. The commonly used distillation technique is sequence level knowledge distillation (Kim and Rush, 2016), where the student models are finetuned on responses from teacher LLMs via a standard language modeling objective.

A.2 Examples for Metrics

A.2.1 Mean Absolute Error

If predictions and TFS performances match perfectly, MAE will be zero, e.g., predictions are [23.4, 25.9, 28.1] and TFS performances are [23.4, 25.9, 28.1]. If predictions and TFS performances are mostly similar, MAE will be low, e.g., predictions are [23.4, 25.9, 28.1] and TFS performances are [23.3, 25.8, 28.2], MAE is 0.1. If predictions and TFS performances are extremely different, MAE will be high, e.g., predictions are [21.2, 24.0, 22.1] and TFS performances are [23.3, 25.8, 28.2], MAE is 3.33.

A.2.2 Kendall-Tau

If predictions and TFS performances match perfectly, Kendall-Tau will be 100, e.g., predictions are [23.4, 25.9, 28.1] and TFS performances are [23.4, 25.9, 28.1]. If predictions and TFS performances are different but their architecture rankings are similar, Kendall-Tau will be 100, e.g., predictions are [23.4, 25.9, 28.1] and TFS performances are [22.2, 23.4, 25.1]. If predictions and TFS performances are different and their architecture rankings are dissimilar, Kendall-Tau will be negative, e.g., predictions are [23.4, 25.9, 28.1] and TFS performances are [23.4, 25.1, 22.2], Kendall-Tau is -0.33.

A.3 Prompt Template - Expanded version

The expanded version of the prompt template can be seen in Figure 3.

A.4 Prompt Template - Design Process

The design process began by examining crucial elements of the machine translation task, commonly used model architectures, and relevant efficiency metrics. Initially, we presented only *demonstrations*, borrowing hyperparameter wording from HAT's configuration file. Subsequently, we added the *role* and definition of each *hyperparameters*, using wording from HAT's helper description. Moving forward, our aim was to craft instructions enabling the LLM to grasp essential tasks, architecture, and metric details. Most instructions are prefixed with 'You should' to encourage strict adherence. Five instructions were incorporated. The first specifies the dataset, translation direction, and quality metric. The second provides examples randomly sampled from the training set, presented with generic prefixes ('Input:' for source sentence, 'Output:' for target sentence). The third outlines the architecture, citing the 'Attention Is All You Need' (Vaswani et al., 2017) paper, assuming the LLM is familiar with this popular work. Standard settings and optimization algorithms are noted for training the architectures. The fourth identifies the efficiency metric in the demonstrations. The final instruction aims to summarize the relationships the LLM should learn to solve the task effectively.

A.5 Kendall-Tau - Fine-grained analysis

We perform a fine-grained analysis of Kendall-Tau performance for Neuron-wise MoS and LLM-PP GPT-4. In figure 4, we plot the histogram of distance between the items in the discordant pairs in the gold ranking for Neuron-wise MoS and LLM GPT-4 across three MT benchmarks. The discordant pairs of LLM-PP lie mostly around low gold ranking distances region (like Neuron-wise MoS), which should not ideally have a big negative impact for the NAS task. In figure 5, we plot the corresponding cummulative distribution function (CDF). The CDF of gold ranking distances for discordant

Role

You are a performance estimator for machine translation task, where you will estimate the BLEU score for the test architecture.

Instruction

You should follow these instructions:
1. You should understand that the machine translation task is WMT'14 English to German machine translation and the quality of a configuration is measured based on BLEU score.
2. Some examples for WMT'14 English to German machine translation are as follows:
Example 1:
Input: Resumption of the session
Output: Wiederaufnahme der Sitzungsperiode
...
Example n_{task}:
Input: Please rise, then, for this minute' s silence.
Output: Ich bitte Sie, sich zu einer Schweigeminute zu erheben.
3. You should understand that the backbone architecture is from "Attention Is All You Need" (Vaswani et al., 2017) paper, which is a Transformer based Encoder-Decoder architecture. We use the same hyperparameters and optimization algorithms.
4. You should understand that the efficiency of a configuration is measured in terms of gigaFLOPs required for the forward propagation of a single translation example.
5. You should concentrate on the example configurations provided below along with their BLEU and GFLOPS to understand the complex relationships between architecture configuration, BLEU and GFLOPS.

Hyperparameters

Hyperparameter definition:
'encoder-embed-dim-subtransformer' corresponds to encoder embedding dimension
'encoder-layer-num-subtransformer' corresponds to number of encoder layers
'encoder-ffn-embed-dim-all-subtransformer' correspond to embedding dimension of each FFN layer in encoder
'encoder-self-attention-heads-all-subtransformer' correspond to number of self attention heads in each encoder layer
'decoder-embed-dim-subtransformer' corresponds to decoder embedding dimension
'decoder-layer-num-subtransformer' corresponds to number of decoder layers
'decoder-ffn-embed-dim-all-subtransformer' correspond to embedding dimension of each FFN layer in decoder
'decoder-self-attention-heads-all-subtransformer' correspond to number of self attention heads in each decoder layer
'decoder-ende-attention-heads-all-subtransformer' correspond to number of cross attention heads in each decoder layer
'decoder-arbitrary-ende-attn-all-subtransformer' correspond to number of encoder layers attended by cross-attention heads in each decoder layer (-1 means only attend to the last layer; 1 means attend to last two layers, 2 means attend to last three layers)

Demonstrations

Example 1:
encoder-embed-dim-subtransformer: 512
encoder-layer-num-subtransformer: 6
encoder-ffn-embed-dim-all-subtransformer: [1024, 1024, 2048, 2048, 2048, 1024]
encoder-self-attention-heads-all-subtransformer: [4, 8, 8, 8, 4, 4]
decoder-embed-dim-subtransformer: 512
decoder-layer-num-subtransformer: 4
decoder-ffn-embed-dim-all-subtransformer: [2048, 1024, 1024, 1024]
decoder-self-attention-heads-all-subtransformer: [4, 4, 8, 4]
decoder-ende-attention-heads-all-subtransformer: [4, 8, 8, 8]
decoder-arbitrary-ende-attn-all-subtransformer: [-1, -1, 1, -1]
BLEU: 24.30
GFLOPS: 2.7
...
Example n_{arch}:
...

Test

Test Architecture:
encoder-embed-dim-subtransformer: 640
encoder-layer-num-subtransformer: 6
encoder-ffn-embed-dim-all-subtransformer: [2048, 1024, 1024, 1024, 2048, 1024]
encoder-self-attention-heads-all-subtransformer: [4, 8, 8, 4, 4, 4]
decoder-embed-dim-subtransformer: 512
decoder-layer-num-subtransformer: 3
decoder-ffn-embed-dim-all-subtransformer: [1024, 2048, 2048]
decoder-self-attention-heads-all-subtransformer: [8, 8, 8]
decoder-ende-attention-heads-all-subtransformer: [8, 4, 4]
decoder-arbitrary-ende-attn-all-subtransformer: [-1, 1, 1]
BLEU:

Figure 3: Prompt template to prompt LLM to generate performance predictions for WMT'14 EN-DE task.

pairs for LLM-PP GPT-4 and Neuron-wise MoS are very similar.

A.6 Machine Translation Details

A.6.1 Machine Translation - Dataset Statistics

The statistics of the MT benchmarks is shown in Table 5.

A.6.2 Machine Translation - Training Details and Search Space

Settings for training machine translation model include: $40K$ training steps, a cosine learning rate scheduler, Adam optimizer, and a warmup of learning rate from 10^{-7} to 10^{-3} with cosine annealing. The validation loss is used for model selection. The beam size is four with length penalty of 0.6. The search space ($\mathcal{A}$) is borrowed from HAT (Wang et al., 2020), which is also shown in Table 6.

A.6.3 Architecture Encoding

Each machine translation architecture is encoded using a list of following 10 values:

1. *Encoder embedding dimension* corresponds to embedding dimension of the encoder.

2. *Encoder #layers* corresponds to number of encoder layers.

3. *Average encoder FFN. intermediate dimension* corresponds to average of FFN intermediate dimension across encoder layers.

4. *Average encoder self attention heads* corresponds to average of number of self attention heads across encoder layers.

5. *Decoder embedding dimension* corresponds to embedding dimension of the decoder.

6. *Decoder #Layers* corresponds to number of decoder layers.

7. *Average Decoder FFN. Intermediate Dimension* corresponds to average of FFN intermediate dimension across decoder layers.

8. *Average decoder self attention heads* corresponds to average of number of self attention heads across decoder layers.

9. *Average decoder cross attention heads* corresponds to average of number of cross attention heads across decoder layers.

10. *Average arbitrary encoder decoder attention* corresponds to average number of encoder layers attended by cross-attention heads in each decoder layer (-1 means only attend to the last layer, 1 means attend to the last two layers, 2 means attend to the last three layers).

A.7 Search and Evaluation Setup - Details

The hyperparameters of HS-NAS's search algorithm are taken from HAT: `num-iterations=30`, `population-size=125`, `num-parents=25`, `num-mutations=50`, `num-crossover=50`, and `mutate-prob=0.3`. We experiment with three `latency-constraints`: $100ms$, $150ms$, and $200ms$. Once the search returns the best architecture, the final weights for this architecture is obtained by training the architecture from scratch to convergence using HAT's training settings (see A.6.2). The target hardware for search is NVIDIA V100 GPU with 32GB GPU RAM. The efficiency metric for search is search hours, which accounts for the time taken to complete all the search iterations. We focus on the following architecture-specific efficiency metrics: (i) *latency* - time taken in milliseconds to encode a sentence in source language and generate the translation in target language, (ii) *GFLOPs* - gigaFLOPs taken for the feedforward propagation, and (iii) *model size* - number of architecture-specific parameters in millions. Scripts to compute these metrics are taken from HAT's codebase [2] and we refer readers to the HAT paper for more details about how these metrics are computed.

A.8 LLM-PP - Extended Results

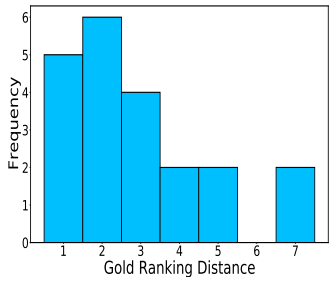
A.8.1 LLM-PP vs. non-supernet baselines.

LLM-PP beats non-supernet baselines as well. We add comparison to five non-supernet baselines: #Params, #FLOPs, grad-norm, snip, and snyflow (see White et al. for details). From Table 7, it is clear that LLM-PP GPT-4 achieves a high Kendall Tau, outperforming all the non-supernet baselines. These results along with Table 1 showcases the superior performance of LLM-PP across a wide range of baselines.

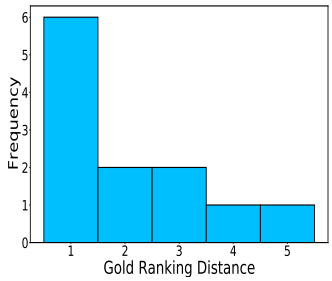
A.8.2 LLM-PP on recent datasets and low-resource/indigenous languages.

LLM-PP works well for recent datasets and low-resource/indigenous languages. Compared to SoTA

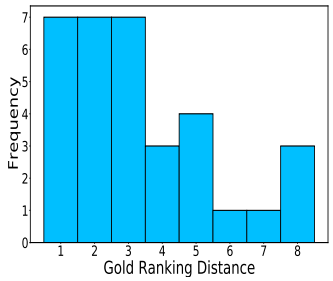
[2] https://github.com/mit-han-lab/hardware-aware-transformers



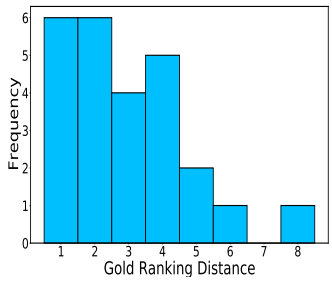
(a) Neur. MoS - WMT'14 En-De



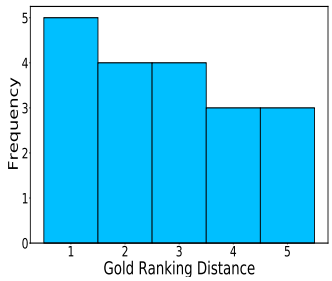
(b) Neur. MoS - WMT'14 En-Fr



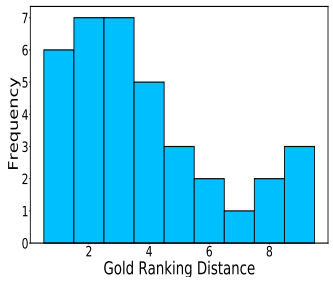
(c) Neur. MoS - WMT'19 En-De



(d) LLM GPT-4 - WMT'14 En-De

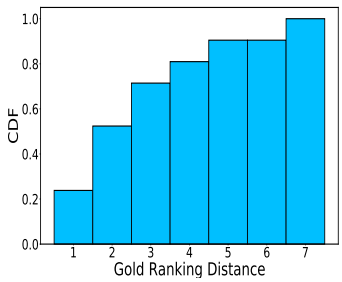


(e) LLM GPT-4 - WMT'14 En-Fr

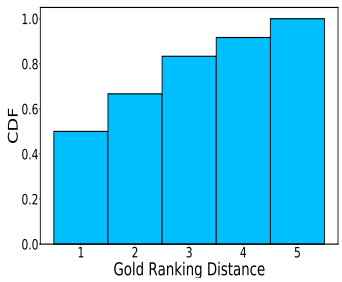


(f) LLM GPT-4 - WMT'19 En-De

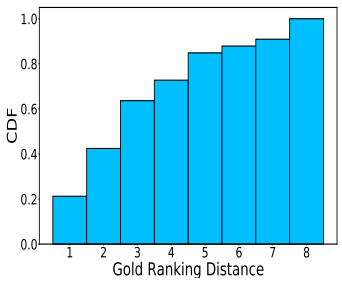
Figure 4: Histogram of distance between the items in the discordant pairs in the gold ranking for Neuron-wise MoS and LLM GPT-4 across three MT benchmarks. The discordant pairs of LLM-PP lie mostly around low gold ranking distances region (like Neuron-wise MoS), which should not ideally have a big negative impact for the NAS task.



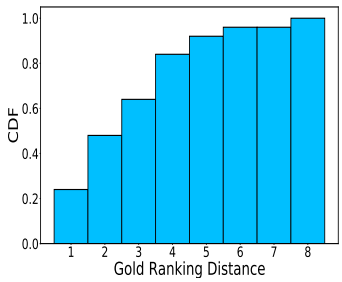
(a) Neur. MoS - WMT'14 En-De



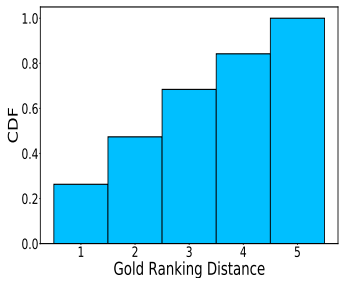
(b) Neur. MoS - WMT'14 En-Fr



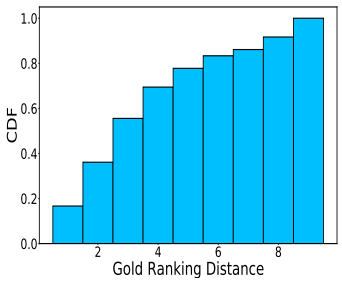
(c) Neur. MoS - WMT'19 En-De



(d) LLM GPT-4 - WMT'14 En-De



(e) LLM GPT-4 - WMT'14 En-Fr



(f) LLM GPT-4 - WMT'19 En-De

Figure 5: Cummulative distribution function of distance between the items in the discordant pairs in the gold ranking for Neuron-wise MoS and LLM GPT-4 across three MT benchmarks. The cummulative distribution function of gold ranking distances for discordant pairs for LLM-PP GPT-4 and Neuron-wise MoS are very similar.

performance predictors, LLM-PP GPT-4 works well for recent datasets (e.g., 2023 benchmark), low-resource/indigenous languages (e.g., Bribri, Chatino). From the recent shared task: "AmericasNLP 2023 Shared Task on Machine Translation into Indigenous Languages" (Ebrahimi et al., 2023), we take three machine translation benchmarks: Bribri to Spanish, Chatino to Spanish, and

Dataset	Year	Source Lang	Target Lang	#Train	#Valid	#Test
WMT	2014	English (en)	German (de)	4.5M	3000	3000
WMT	2019	English (en)	German (de)	43M	2900	2900
WMT	2014	English (en)	French (fr)	35M	26000	26000

Table 5: Statistics - Machine translation benchmark.

Hyperparameter Attribute	Value choices
Encoder-Embedding-Dim	{512, 640}
Decoder-Embedding-Dim	{512, 640}
#Encoder-Layers	{6}
#Decoder-Layers	{1, 2, 3, 4, 5, 6}
Encoder-QKV-Dim	{512}
Decoder-QKV-Dim	{512}
#Encoder-Self-Attention-Heads (PL)	{4, 8}
#Decoder-Self-Attention-Heads (PL)	{4, 8}
#Decoder-Cross-Attention-Heads (PL)	{4, 8}
#Decoder-Arbitrary-Attention (PL)	{-1, 1, 2}
Encoder-FFN-Intermediate-Embed-Dim (PL)	{1024, 2048, 3072}
Decoder-FFN-Intermediate-Embed-Dim (PL)	{1024, 2048, 3072}

Table 6: Search space ($\mathcal{A}$), borrowed from HAT (Wang et al., 2020). 'PL' refers to hyperparameters that vary per layer.

Kendall-Tau	WMT'14 En-De	WMT'14 En-Fr	WMT'19 En-De
# Params	0.42	0.51	0.54
# FLOPs	0.43	0.53	0.54
grad-norm	-0.42	-0.42	-0.52
snip	-0.42	-0.27	-0.3
synflow	-0.31	-0.47	-0.49
LLM-PP GPT-4	**0.65**	**0.75**	**0.65**

Table 7: Kendall-Tau of LLM-PP GPT-4 vs. non-supernet baselines. LLM-PP beats non-supernet baselines as well.

Spanish to Bribri. Compared to WMT 2014, WMT 2019 benchmarks, these three benchmarks are very recent (2023 year) and one of the languages in each translation direction is an low-resource/indigenous language (Bribri, Chatino). As shown in Table 8, we compare LLM-PP GPT-4 against SoTA performance (BLEU) predictors on these benchmarks in terms of quality (MAE, Kendall-Tau). It is clear that LLM-PP achieves the SoTA MAE score across these benchmarks, which is consistent with the trends in WMT 2014, WMT 2019 benchmarks (as shown in Table 1). Impressively, on two of these benchmarks, LLM-PP also achieves the SoTA Kendall-Tau score. Put together, these results clearly showcase that LLM-PP generalizes well to recent datasets and low-resource languages.

A.8.3 LLM-PP for COMET metric.

LLM-PP generalizes well to uncommon evaluation metrics. We build performance predictors that predict the Crosslingual Optimized Metric for Evaluation of Translation (COMET) (Rei et al., 2022) (Unbabel/wmt22-comet-da), which is relatively newer than the BLEU metric. Consider the Table 9 (performance averaged across two seeds), on the Bribri to Spanish task and the Chatino to Spanish task, LLM-PP achieves the SoTA MAE and SoTA Kendall Tau performance compared to SoTA performance predictors. These results show that LLM-PP generalizes well to uncommon evaluation metrics like COMET. Note that we exclude Spanish to BriBri task, since COMET does not support Bribri.

A.8.4 LLM-PP for robust predictions.

LLM-PP provides fairly robust performance predictions. We compute the predictions for 8500 randomly sampled architectures using LLM-PP GPT-4 three times and compute the standard deviation of the three predictions for each architecture. The mean of the standard deviation for 8500 architectures is very low: 0.21, 0.27, 0.27 BLEU for WMT'14 En-De, WMT'14 En-Fr, and WMT'19 En-De respectively. Thus, LLM-PP provides fairly robust performance predictions. For all our search experiments, we use a single estimate from LLM-PP.

A.9 HS-NAS - Expanded Algorithm

The expanded algorithm for HS-NAS can be found in Algorithm 2.

A.10 LLM-Distill-PP - Extended Results

A.10.1 Performance predictor quality vs. prediction time.

Table 10 shows the efficiency (time taken to predict performance for 10 architectures) and accuracy (MAE, Kendall) for supernet-based PP (HAT, Layer-wise MoS, Neuron-wise MoS), LLM-PP (GPT-4), and LLM-Distill-PP (GPT-4). LLM-Distill-PP provides the best efficiency-accuracy tradeoff with on par accuracy as LLM-PP but significantly faster prediction time (0.01s vs. 11.9s).

A.10.2 Varying initialization seeds.

HS-NAS seems robust to initialization effects caused by different seeds, achieving largely similar

Algorithm 2 Hybrid-Search algorithm for Neural Architecture Search (HS-NAS). Changes to HAT (Wang et al., 2020)'s search algorithm are in red color.

Input:
LLM-Distill-PP model: `llm-distill-pp`,
Weight-sharing supernet: `supernet`,
Latency predictor: `latency-predictor`,
#Search iterations: `num-iterations`,
Population size: `population-size`,
#Parents: `num-parents`,
#Mutations: `num-mutations`,
#Crossovers: `num-crossover`,
Mutate probability: `mutate-prob`,
Latency constraint: `latency-constraint`,
LLM-Distill-PP Start Iteration: `llm-start-iteration`,
LLM-Distill-PP End Iteration: `llm-end-iteration`
Output: `best-architecture`

```
popu ← population-size rand. samples from search space // create init. population
for iter ← 1 to num-iterations do
    // gen. parents by picking top cand. arch.
    if llm-start-iteration < iter < llm-end-iteration then
        parents ← top 'num-parents' arch. from popu by llm-distill-pp
    else
        parents ← top 'num-parents' arch. from popu by supernet
    // gen. cand. via mutation
    mutate-popu = {}
    for mi ← 1 to num-mutations do
        gene ← mutate a random eg from popu with mutate-prob
        if gene satisfies latency-constraint via latency-predictor then
            mutate-popu = mutate-popu ∪ gene
    // gen. cand. via cross-over
    crossover-popu = {}
    for ci ← 1 to num-crossover do
        gene ← crossover two random eg from popu
        if gene satisfies latency-constraint via latency-predictor then
            crossover-popu = crossover-popu ∪ gene
    // upd. population
    popu = parents ∪ mutate-popu ∪ crossover-popu
return top arch. from popu
```

Dataset Performance Predictor	Bribri to Spanish MAE	 Kendall	Chatino to Spanish MAE	 Kendall	Spanish to Bribri MAE	 Kendall
HAT	0.28	0.15	1.55	**0.16**	0.72	0.02
Layer-wise MoS	0.33	-0.13	2.42	-0.17	0.63	-0.14
Neuron-wise MoS	0.29	-0.35	2.94	-0.06	0.43	0.09
LLM-PP GPT-4	**0.16**	**0.29**	**1.21**	0.08	**0.32**	**0.20**

Table 8: MAE and Kendall-Tau between the performance predictor performance and the TFS performance, across two different seeds. LLM-PP works well for recent datasets and low-resource/indigenous languages.

Dataset Performance Predictor	Bribri to Spanish MAE	 Kendall	Chatino to Spanish MAE	 Kendall
HAT	0.03	0.24	0.02	-0.15
Layer-wise MoS	0.02	-0.05	0.02	0.26
Neuron-wise MoS	0.02	0.32	0.01	0.34
LLM-PP GPT-4	**0.01**	**0.32**	**0.01**	**0.54**

Table 9: MAE and Kendall-Tau between the performance predictor performance and the TFS performance for COMET metric, across two different seeds. LLM-PP generalizes well to uncommon evaluation metrics like COMET.

Performance Predictor	MAE	Kendall-Tau	Prediction Time (s)
HAT	1.14	0.71	10.5
Layer-wise MoS	1.05	**0.81**	13.9
Neuron-wise MoS	0.97	0.56	13.3
LLM-PP GPT-4	0.28	0.65	11.9
LLM-Distill-PP GPT-4	**0.22**	0.64	**0.01**

Table 10: Performance predictor quality vs. prediction time.

numbers on all metrics. This result is shown in Table 11, where latency numbers change slightly while numbers for other metrics are almost the same.

A.10.3 Varying FLOPs constraints.

HS-NAS performs similarly to HAT for different FLOPs constraints, with at least 16% reduction in search hours, 1.2% improvement in latency, same GFLOPs and same model size. Table 12 contains these superior results of HS-NAS across 2.5 and 3.0 GFLOPs constraints. These trends largely hold true across benchmarks as well, as shown in Table 13.

A.10.4 Varying underlying supernet.

The dominance of HS-NAS seems consistent across the underlying supernet. In the results so far, HAT is the supernet used by HS-NAS. In Table 14, we replace HAT with Neuron-wise MoS and show that HS-NAS performs similarly to Neuron-wise MoS, with at least 50% reduction in search hours, better or similar model size and GFLOPs.

A.10.5 Trivially constructed efficient adaptations of SoTA

Search hours can be trivially reduced in several ways: halving the total number of search iterations and/or using distilled SoTA predictor instead of using supernet predictor directly. As shown in Table 15, the former approach suffers from a big drop in BLEU performance (1.8% for HAT (`num-iter.`=15)), while the latter approach suffers from a big increase in latency and GFLOPs (9.7% and 32% respectively for Distilled HAT (`num-iter.`=15)). On the other hand, HS-NAS dominates these adaptions in search hour reductions, while maintaining the performance of SoTA and not degrading on any footprint metric.

Seed	BLEU (↑)	Latency (ms) (↓)	GFLOPs (↓)	Model Size (M) (↓)	Search Hours (↓)
100ms					
1	40.7	104.1	2.54	63.8	3.14
2	40.7	98.2	2.54	63.8	3.15
3	40.7	101.2	2.58	63.8	3.16
150ms					
1	41.5	160.4	3.35	74.3	3.89
2	41.4	172.6	3.31	74.3	3.69
3	41.5	158.5	3.35	74.3	3.84

Table 11: Initialization effects of HS-NAS (GPT-4, HAT, 1, 15) on WMT'14 En-Fr for different latency constraints - Test BLEU, latency in milliseconds, GFLOPs, model size in millions, and search hours. HS-NAS seems robust to initialization effects, achieving similar numbers on all metrics of interest.

Search	BLEU (↑)	Latency (ms) (↓)	GFLOPs (↓)	Model Size (M) (↓)	Search Hours (↓)
2.5 GFLOPs					
HAT	**26.9**	69.5	**2.47**	**41.0**	2.54
HS-NAS	26.7	**68.6**	**2.47**	**41.0**	**2.13**
3.0 GFLOPs					
HAT	27.5	125.4	**2.98**	**49.4**	2.08
HS-NAS	**27.6**	**123.9**	**2.98**	**49.4**	**1.51**

Table 12: HS-NAS (GPT-4, HAT, 1, 15) vs. HAT on WMT'14 En-De for different FLOPs constraints - Test BLEU, latency in milliseconds, GFLOPs, model size in millions, and search hours. HS-NAS (GPT-4, HAT, 1, 15) performs similarly to HAT, with at least 16% reduction in search hours, 1.2% improvement in latency, same GFLOPs and same model size.

Search	BLEU (↑)	Latency (ms) (↓)	GFLOPs (↓)	Model Size (M) (↓)	Search Hours (↓)
WMT'14 En-De					
HAT	27.5	125.4	**2.98**	**49.4**	2.08
HS-NAS	**27.6 (+0.4%)**	**123.9 (-1.2%)**	**2.98**	**49.4**	**1.51 (-27.4%)**
WMT'14 En-Fr					
HAT	39.4	**69.6**	**2.99**	**49.1**	6.69
HS-NAS	**39.8 (+1%)**	96.8 (+39.1%)	**3**	**49.1**	**4.2 (-37.2%)**
WMT'19 En-De					
HAT	42.9	85.5	**2.99**	**49.6**	2.35
HS-NAS	**43.1 (+0.5%)**	**71.9 (+15.9%)**	**2.99**	**49.6**	**2.03 (-13.6%)**

Table 13: HS-NAS (GPT-4, HAT, 1, 15) vs. HAT across benchmarks for 3.0 GFLOPs constraint - Test BLEU, latency in milliseconds, GFLOPs, model size in millions, and search hours. HS-NAS (GPT-4, HAT, 1, 15) performs similarly or better than HAT, with at least 13% reduction in search hours, at least 1.2% improvement in latency (in most cases), same GFLOPs, and same model size.

Search	BLEU (↑)	Latency (ms) (↓)	GFLOPs (↓)	Model Size (M) (↓)	Search Hours (↓)
100ms					
Neuron-wise MoS	**40.9**	**97.6**	**3.13**	**70.5**	7.03
HS-NAS (GPT-4, Neur., 1, 15)	**40.9**	126.9 (+30%)	**3.13**	**70.5**	**3.36 (-52.2%)**
150ms					
Neuron-wise MoS	**41.4**	200.2	4.26	92.8	8.35
HS-NAS (GPT-4, Neur., 1, 15)	41.3 (-0.2%)	**162.2 (19.0%)**	**4.22 (-0.9%)**	**91.5 (1.4%)**	**4.14 (-50.4%)**
200ms					
Neuron-wise MoS	41.6	**184.1**	**4.53**	**99.4**	8.77
HS-NAS (GPT-4, Neur., 1, 15)	**41.7 (+0.2%)**	191.2 (+3.9%)	**4.53**	**99.4**	**4.22 (-51.8%)**

Table 14: HS-NAS (GPT-4, Neuron-wise MoS, 1, 15) versus SoTA NAS on WMT'14 En-Fr for different latency constraints - Test BLEU, latency in milliseconds, GFLOPs, model size in millions, and search hours. HS-NAS is accompanied by four arguments: (`llm-distill-pp`, `supernet`, `llm-start-iteration`, `llm-end-iteration`). Across latency constraints, HS-NAS performs similarly or improves upon SoTA NAS, with at least 50% reduction in search hours, better or similar model size and GFLOPs.

Search	BLEU (↑)	Latency (ms) (↓)	GFLOPs (↓)	Model Size (M) (↓)	Search Hours (↓)
HAT (num-iter.=30)	**27.9**	102.0	3.0	64.4	1.09
HAT (num-iter.=15)	27.4 (-1.8%)	107.6 (+5.5%)	**2.96 (-1.3%)**	**63.1 (-2%)**	0.65 (-40.4%)
Distilled HAT (num-iter.=15)	27.8 (-0.4%)	111.9 (+9.7%)	3.97 (+32%)	**63.1 (-2%)**	0.58 (-46.8%)
HS-NAS (GPT-4, HAT, 1, 15)	**27.9**	**99.7 (-2.3%)**	**2.96 (-1.3%)**	**63.1 (-2%)**	**0.56 (-48.6%)**

Table 15: HS-NAS versus trivial efficient adaptations of SoTA with half of the original search iterations (original num-iterations $= 30$): *original SoTA*, *distilled SoTA* on WMT'14 En-De for 100ms latency constraint - Test BLEU, latency in milliseconds, GFLOPs, model size in millions, and search hours. HS-NAS is accompanied by four arguments: (llm-distill-pp, supernet, llm-start-iteration, llm-end-iteration). Efficient adaptations of SoTA reduce search hours by at least 40%, at the expense of either a big drop in BLEU performance (1.8% for HAT (num-iter.=15)) or big increase in latency and GFLOPs (9.7% and 32% respectively for Distilled HAT (num-iter.=15)). On the other hand, HS-NAS dominates these adaptions in search hour reductions, while maintaining the performance of SoTA and not degrading on any footprint metric.