MeRino: Entropy-Driven Design for Generative Language Models on IoT Devices

Youpeng Zhao[1], Ming Lin[2], Huadong Tang[3], Qiang Wu[3], Jun Wang[1]
[1]Department of Computer Science, University of Central Florida
[2]Independent Researcher
[3]School of Electrical and Data Engineering, University of Technology Sydney
youpeng.zhao@ucf.edu, linming04@gmail.com, huadong.tang@student.uts.edu.au,
qiang.wu@uts.edu.au, jun.wang@ucf.edu

Abstract

Generative Large Language Models (LLMs) stand as a revolutionary advancement in the modern era of artificial intelligence (AI). However, scaling down LLMs for resource-constrained hardware, such as Internet-of-Things (IoT) devices requires non-trivial efforts and domain knowledge. In this paper, we propose a novel information-entropy framework for designing mobile-friendly generative language models. The whole design procedure involves solving a mathematical programming (MP) problem, which can be done on the CPU within minutes, making it nearly zero-cost. We evaluate our designed models, termed MeRino, across fourteen NLP downstream tasks, showing their competitive performance against the state-of-the-art autoregressive transformer models under the mobile setting. Notably, MeRino achieves similar or better performance on both language modeling and zero-shot learning tasks, compared to the 350M parameter OPT while being 4.9× faster on NVIDIA Jetson Nano with 5.5× reduction in model size.

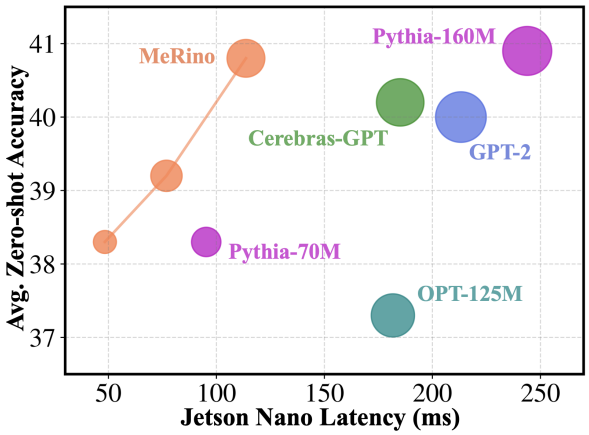

Figure 1: Average zero-shot accuracy and inference latency on NVIDIA Jetson Nano for mobile-level LLMs. Results were evaluated using lm-evaluation-harness (Gao et al. 2021) on open-sourced pre-trained models. The diameter of each circle denotes the corresponding model FLOPs.

Introduction

The Transformer architecture, originally introduced in (Vaswani et al. 2017), has revolutionized the field of natural language processing (NLP). It has become the de-facto building block in many pre-trained generative large language models (LLMs) (Radford et al. 2019; Brown et al. 2020). Thanks to their ability to scale up to billion-level parameters, LLMs have exhibited exceptional abilities in solving complex tasks through text generation, with prominent applications such as ChatGPT (OpenAI 2022) and Claude (Anthropic 2023). Nevertheless, running LLMs in cloud computing platforms with specialized hardware accelerators can be very expensive. It is estimated that running a single ChatGPT query might consume around 0.3 kWh, which is roughly 1000× more than a simple Google search, highlighting the significant financial and environmental impact of recurring LLM usage (Soham 2024). Therefore, it is imperative to downsize the footprints of LLMs to meet the sustainability and efficiency requirements of ethical and responsible AI solutions (Wu et al. 2022).

With the ever-increasing popularity of edge computing, deploying LLMs on resource-constrained hardware, e.g., mobile phones and internet-of-things (IoT) devices becomes a much more appealing solution. Such on-device AI both speeds up response latency and enhances data privacy and security, making LLMs more accessible, efficient, and practical in a wide range of daily applications (Apple 2024; Microsoft 2024).

However, the practical deployment of LLMs for edge devices has met with its unique obstacles. First, integrating existing pre-trained LLMs can be prohibitively expensive, as the weights for most models can easily exceed the main memory capacity. For instance, for the OPT-125M model, running a single query of length 128 would consume over 1,200 MB of DRAM or flash memory, with close to 200 ms latency on NVIDIA Jetson Nano. For mobile applications, the ideal memory usage should be less than 10% of the total capacity, to avoid potential system overloading and out-of-memory (OOM) failure. Several works (Liu et al. 2024; Hu et al. 2024) have considered designing optimized LLMs in the regime of 1B parameter, which yields better task performances but suffers from the same latency overhead as previous methods. This motivates us to deploy even smaller LLMs (<100 M) with even lower latency

(<100 ms). Second, the hardware configuration for each edge computing platform varies from device to device. Thus, designing a one-fits-all model that satisfies all requirements is a non-trivial task. Neural architecture search (NAS) (Xu et al. 2021), has recently emerged as a promising solution for efficient architecture designs that offer better accuracy and computation trade-offs against manually designed models. However, NAS methods are usually computationally expensive, due to the costly model evaluations and supernet-training processes, which often consume thousands of GPU hours. To mitigate the high costs and promote the search efficiency of NAS, a series of proxies (Javaheripi et al. 2022; Zhou et al. 2022) have been proposed to evaluate the accuracy of neural network architectures in a low-cost manner. However, these methods generally focus on vision models, and directly applying them to language models could produce sub-optimal solutions. Moreover, these methods usually require forward and backward passes over the architecture, which could be compute-intensive for low-end hardware.

In this work, we present an entropy-driven framework to design lightweight variants of generative language models tailored for resource-constrained devices **under 100 M parameters**. Our key idea is to maximize the entropy for autoregressive transformers, which positively correlates with the model performance, under given computational constraints, such as parameter size, FLOPs, and latency. The optimal model configuration is generated by solving a mathematical programming (MP) problem utilizing an Evolutionary Algorithm (EA). Our entropy-driven algorithm can automatically design an optimal transformer model running on the target IoT hardware within minutes, significantly reducing the design costs of existing NAS-based methods.

Our designed models termed **MeRino**, achieve competitive performance compared to OPT and GPT models, with much better parameter and computation efficiency across numerous NLP downstream tasks. Notably, MeRino obtains comparable accuracy performance against OPT-350M, with $5.5\times$ reduction in model size, $4.5\times$ reduction in FLOPs, and $4.9\times$ faster latency on NVIDIA Jetson Nano. MeRino also significantly outperforms previous NAS-based methods by a clear margin, in terms of both model accuracy and search efficiency.

The key contributions of this work are summarized as follows:

- We propose an entropy-driven framework to address the challenge of designing efficient generative language models for resource-constrained devices at nearly zero cost.
- Our design paradigm leverages the Maximum Entropy Principle and constrained mathematical programming (MP) to optimize transformer architectures given computation budgets.
- Experimental results show that our designed models, termed MeRino, achieve much better accuracy/computation tradeoffs against both manually designed and NAS-designed models, with improved parameter, computation, and latency speedup on mobile devices.

Related Work

Large Language Models (LLMs). Generative large language models (LLMs) have emerged as the standard solution to a wide range of NLP tasks. GPT-3 (Brown et al. 2020), in particular, pushed the boundaries of casual language models by scaling up the model size to 175 billion parameters. In the pursuit of democratizing and fostering reproducible research in LLMs, a series of open-sourced family models are subsequently released, most notably OPT (Zhang et al. 2022), LLaMA (Touvron et al. 2023), Pythia (Biderman et al. 2023), and Cerebras-GPT (Dey et al. 2023). MiniCPM (Hu et al. 2024) and MobileLLM (Liu et al. 2024) further unveil the potential of LLMs under mobile settings through various training and design strategies. However, **the above-mentioned models are all above 100 M parameter size**; in this work, we aim to design transformer-based language models in sub-100 M regime that achieves competitive performance for IoT devices with limited memory space and compute power.

Model Compression. Two of the most widely studied techniques in designing compressed versions of language models are quantization and sparsification. Quantization (Frantar et al. 2023) reduces the memory footprint by quantizing the model weights from high precision (FP16/32) to low precision (INT4/8) at the cost of negligible accuracy performance drop. Sparsity-driven methods aim to remove unnecessary computation by identifying the most crucial components in either model weights (Frantar and Alistarh 2023; Ashkboos et al. 2024) or Key-Value (KV) cache (Zhang et al. 2023; Zhao, Wu, and Wang 2024), to accelerate the LLM inference process. In this work, orthogonal to the above-mentioned methods, our proposed approach emphasizes the architecture design of lightweight transformer-based LLMs.

Neural Architecture Search (NAS). Due to its success in computer vision (CV), neural architecture search (NAS) has recently gained attention in the NLP community. The general approach is to train a giant supernet (Xu et al. 2021) to efficiently search for compressed language model versions, however, it usually incurs heavy computation costs and requires extensive engineering efforts. Training-free proxies have been proposed to reduce the evaluation costs of existing NAS methods. TE-NAS (Chen, Gong, and Wang 2021) calculates the neural tangent kernel (NTK) score to estimate the architecture accuracy and TF-TAS (Zhou et al. 2022) proposes a gradient-based DSS-Score to rank the architectures in Vision Transformers (ViTs). LTS (Javaheripi et al. 2022) utilizes the decoder parameter count as a proxy for perplexity ranking in generative language models. However, these methods are generally data-dependent, requiring storing network parameters and weights in memory, which could be compute-intensive on IoT devices with limited power. In this work, we propose an entropy-based framework that can design better model architecture and run directly on target hardware with higher search efficiency.

Information Theory in Deep Learning. Information theory recently has emerged as a powerful tool for studying deep neural networks. Several previous studies (Chan et al.

2021; Saxe et al. 2018) have attempted to establish a connection between the information entropy and the neural network architectures. For instance, (Chan et al. 2021) tries to interpret the learning ability of deep neural networks using subspace entropy reduction. (Saxe et al. 2018) investigates the information bottleneck in deep architectures and explores the entropy distribution and information flow in deep neural networks. Additionally, (Sun et al. 2021) focuses on designing high-performance vision models via maximizing multi-level entropy. In this work, we focus on using information entropy to design efficient generative transformer language models.

Methodology

In this section, we begin by presenting some preliminary details on transformer models. Next, we introduce our novel definition of network entropy for transformer models. Moreover, we demonstrate that the subspace entropy positively correlates with the model performance after training. Finally, we present our entropy-driven design procedure, which solves a constrained mathematical programming problem using the Evolutionary Algorithm (EA).

Preliminaries

Multi-Head Self-Attention (MHSA). Multi-head attention (MHSA) is a crucial component within the transformer architecture that enables the model to selectively attend to different segments of the input sequence. This mechanism involves projecting the input sequence into multiple attention heads, each of which calculates an independent attention distribution. In MHSA computation, there are specifically four main matrices involved: attention matrices $W^Q, W^K, W^V \in \mathbb{R}^{d_{in} \times d_{in}/h}$ and the final output project matrix $W^O \in \mathbb{R}^{d_{in} \times d_{out}}$. Given the output of previous layers $X \in \mathbb{R}^{n \times d_{in}}$ as input, the attention function is formulated as:

$$Q, K, V = XW^Q, XW^K, XW^V \tag{1}$$

$$\text{Attn}(Q, K, V) = \text{SoftMax}(\frac{QK^T}{\sqrt{d_{in}/h}})(V) \tag{2}$$

where Q, K, and V represent queries, keys, and values, respectively. MHSA is defined by concatenating h attention heads and producing outputs as follows:

$$\text{MHSA}(X) = \text{Concat}(\text{Attn}_i, ..., \text{Attn}_h)W^O \tag{3}$$

In addition, the transformer layer adopts residual connection and layer normalization on top of MHSA to compute the final outputs.

$$X^{\text{MHSA}} = \text{LayerNorm}(X + \text{MHSA}(X)) \tag{4}$$

Position-wise Feed-forward Network (FFN). In addition to the MHSA, each transformer layer includes a feed-forward network (FFN). The FFN applies two point-wise fully connected layers followed by a non-linear activation function, such as ReLU. Operations within FFN can be formulated as follows:

$$X^{\text{FFN}} = \text{ReLU}(X^{\text{MHSA}}W^{\text{FFN}_1} + b_1)W^{\text{FFN}_2} + b_2 \tag{5}$$

Similarly, the FFN also incorporates residual connections and layer normalization to compute the final outputs:

$$X^{\text{FFN}} = \text{LayerNorm}(X^{\text{MHSA}} + X^{\text{FFN}}) \tag{6}$$

Entropy of Neural Network

From the perspective of information theory (Jaynes 1957), deep neural networks (DNNs) can be regarded as information systems, and their performance is closely related to the expressive power of such networks. The notion of entropy is often used to measure such expressiveness in DNNs. The fundamental operation in DNNs is matrix multiplication, which computes the weighted sum of inputs and weights in each layer of a neural network. According to previous works (Chan et al. 2021), for a simple one-layer neural network with weight matrix $W \in \mathbb{R}^{c_1 \times c_2}$, the entropy can be defined as

$$\widehat{H}(W) \triangleq \mathbb{E}\{\sum_{j=1}^{r_i} \log(1 + \frac{s_j^2}{\epsilon^2})\} \tag{7}$$

where $r_i = \min(c_1, c_2)$, s_j is the j-th largest singular value of W and ϵ is a small constant.

For an L-layer network $f(\cdot)$, we can define the network entropy by accumulating the entropy of each layer as:

$$\widehat{H}_f = \widehat{H}(W_1, W_2, ..., W_L) = \sum_{i=1}^{L} \widehat{H}(W_i) \tag{8}$$

The entropy measures the *expressiveness* of the deep neural network, which is positively correlated with the network performance (Sun et al. 2021). However, directly maximizing the above-defined entropy leads to the creation of over-deep networks, since according to Eq. (8), the expressivity (entropy) grows exponentially faster in depth (number of layers L), than in width (dimension of W_i). For an over-deep network, a small perturbation in low-level layers of the network will lead to an exponentially large perturbation in the high-level output of the network (Roberts, Yaida, and Hanin 2021). During the back-propagation process, the gradient flow often cannot effectively propagate through the entire network.

Effectiveness of Neural Network

To verify the negative impact when the network is over-deep, in Table **??**, we conduct experiments of training two transformer architectures with a similar parameter size of 40 M. One model, referred to as the 'Wide' model, consists of only one layer and an embedding dimension of 256. The other model, referred to as the 'Deep' model, consists of 24 layers but only with an embedding dimension of 64. Both models are trained under the same setting until convergence. We observe that even though the 'Deep' network has much higher entropy, it obtains worse perplexity performance after training than the 'Wide' network. This observation aligns with

Model	L	E	Params	Entropy	Effective γ	Entropy w/ γ	Perplexity
'Wide'	1	256	40 M	2784	**0.008**	**2243**	**53.7**
'Deep'	24	64	40 M	**4680**	0.25	2042	71.9

Table 1: Perplexity comparison of two different structures of autoregressive transformer models on the LM1B dataset. Lower perplexity is better.

the common belief that over-deep networks hinder effective information propagation and are difficult to train and optimize (Roberts, Yaida, and Hanin 2021).

To address the potential trainability issues, we propose adding additional constraints to control the depth-width ratio of networks. Specifically, we adopt the term *effectiveness* γ from the work (Roberts, Yaida, and Hanin 2021) and define it as follows:

$$\gamma = \beta L / \hat{w} \tag{9}$$

Here, $\hat{w}$ is the effective width of a L-layer network and β is a scaling factor to control γ within the range of 0 and 1. To enforce the above constraint, we revise Eq. (9) as follows:

$$\widehat{H}_f = (1-\gamma)\sum_{i=1}^{L} H(W_i) \tag{10}$$

Compared to the previous subspace entropy definition, Eq. (10) penalizes networks with larger depth-to-width ratios (higher γ). This constraint helps alleviate potential trainability issues by promoting a more balanced depth-width ratio in the network architecture. By considering both *expressiveness* (entropy) and *effectiveness* (the depth-width ratio), we aim to design more capable and trainable models.

Entropy of Transformers

Consider a L-layer transformer model with embedding dimension E and FFN dimension F, according to Theorem 1 in (Levine et al. 2020), the depth-width sufficiency behavior satisfied a logarithmic condition in transformer models. Subsequently, we propose to define the effective width of MHSA and FFN and their corresponding entropy as:

$$\hat{w}_{\text{MHSA}} = \log E, \qquad \hat{w}_{\text{FFN}} = \log F \tag{11}$$

$$\widehat{H}_{\text{MHSA}} = (1 - \frac{\beta L}{\hat{w}_{\text{MHSA}}}) \sum_{i=1}^{L} \widehat{H}(W_i^Q, W_i^K, W_i^V, W_i^O) \tag{12}$$

$$\widehat{H}_{\text{FFN}} = (1 - \frac{\beta L}{\hat{w}_{\text{FFN}}}) \sum_{i=1}^{L} \widehat{H}(W_i^{\text{FFN}_1}, W_i^{\text{FFN}_2}) \tag{13}$$

Therefore, we define the total entropy of the transformer model as linear combinations of the MHSA and FFN entropy:

$$\widehat{H} = \alpha_1 \widehat{H}_{\text{MHSA}} + \alpha_2 \widehat{H}_{\text{FFN}} \tag{14}$$

where $\alpha = (\alpha_1, \alpha_2)$ are tunable hyperparameters.

Fast Entropy Approximation. Given the above definitions, we can easily calculate entropy for any transformer model.

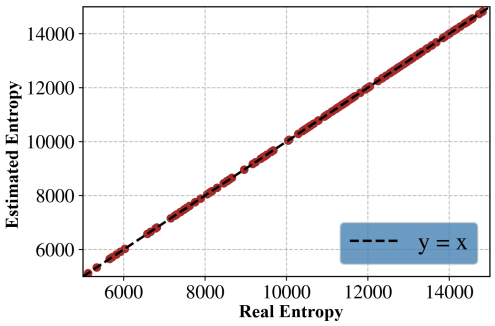

Figure 2: Our entropy estimation based on table lookup is very accurate, with an average error rate of 0.03%.

However, performing singular value decomposition (SVD) is a costly operation. For large models, it sometimes requires minutes to run SVD, which inhibits an efficient design. To accelerate the entropy computation, we build an entropy lookup table to approximate the total entropy of a given transformer model. The lookup table is built through a pre-computation process that considers all possible combinations of expected entropy values for different dimensions. This step incurs only a one-time cost and the resulting lookup table can be shared across multiple experiments. With the lookup table in place, we can efficiently calculate the entropy of transformer models and enable a more efficient design process for transformer models. Figure 2 shows that our table lookup method can provide very accurate entropy estimation at almost zero cost, thus greatly speeding up the search process.

Evaluating Transformer without Training. Recent studies (Jaynes 1957; Sun et al. 2021) have demonstrated that entropy, which captures the information capacity of neural network architecture, can be a reliable indicator for final model accuracy. In this part, we provide experimental results that empirically establish a strong correlation between our proposed entropy of transformers and their perplexity results on the One Billion Word (LM1B) (Chelba et al. 2013) dataset after training. To this end, we uniformly sample 81 unique transformer architectures and each model is fully trained from *scratch*. The performance is measured using perplexity and we calculate the correlation between each transformer model's entropy and perplexity score on the validation set.

Figure 3 illustrates the correlation between the sampled architectures' performance (negative perplexity) and their entropy. The results indicate strong correlations, as evidenced by Spearman's Rank Correlation (ρ) and Kendall Rank Correlation (τ) scores exceeding 0.8 and 0.6, respectively. Furthermore, we compare our entropy to three com-

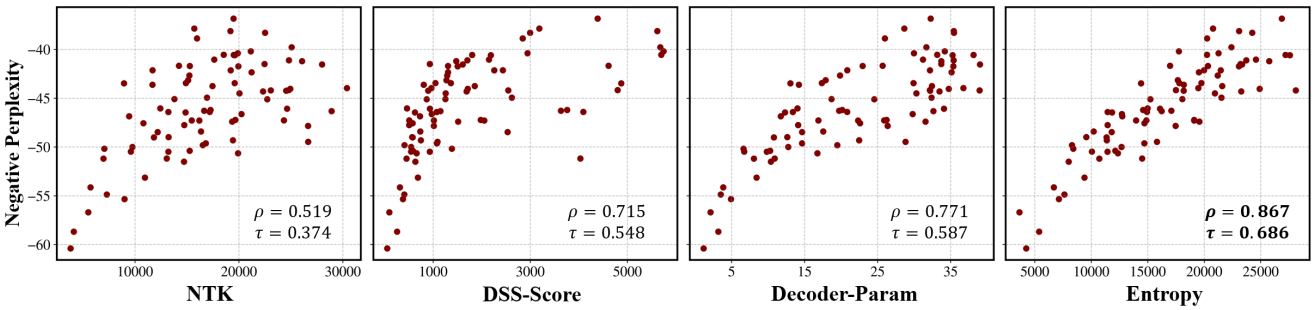

Figure 3: Correlation comparison of different training-free predictors, e.g., NTK (Chen, Gong, and Wang 2021), DSS-Score (Zhou et al. 2022), and Decoder-Param (Javaheripi et al. 2022), and transformer performance (negative perplexity, higher is better). ρ is Spearman's Rank and τ is Kendall Tau. Larger values mean higher correlation.

monly used low-cost evaluation proxies, namely Decoder Params in LTS (Javaheripi et al. 2022), DSS-Score in TF-TAS (Zhou et al. 2022), and NTK in TE-NAS (Chen, Gong, and Wang 2021). As depicted in Figure 3, our proposed entropy outperforms the other three accuracy predictors. Particularly, our method is more capable of identifying high-performance transformer architecture than previous methods.

Note that there are two principal distinctions between our entropy-driven approach and previous zero-shot NAS methods (Javaheripi et al. 2022; Zhou et al. 2022; Chen, Gong, and Wang 2021). First, zero-shot NAS methods are predominantly *data-driven*. Our method, on the other hand, is mathematically driven with clear motivation from the perspective of information theory (Jaynes 1957). Second, zero-shot NAS methods usually require forward/backward passes over the architecture, where the model parameters and feature maps have to be stored in GPU memory. In contrast, our methodology is purely analytical and the expensive entropy calculation process is substituted by a table lookup procedure, therefore highly efficient. **Our method requires zero GPU memory in the design stage**. In summary, our method is a much better approach to designing efficient language models for edge devices than zero-shot NAS.

Designing Mobile Language Models

Search Space. In the design of MeRino, we introduce an adaptive block-wise search space to construct the backbone architecture, as shown in Figure 4. Each transformer block consists of numerous transformer layers of the same number of attention heads, FFN dimensions, and embedding dimensions. Moreover, we incorporate parameter sharing technique (Lan et al. 2019) within each transformer block. This means that all MHSA and FFN layers share the same weights, resulting in transformer models with reduced memory footprint.

Within each transformer block, in MHSA layers, we fix the head dimension to 64 and make the attention head number elastic so that each attention module can decide its necessary number of heads. We also set the *Q-K-V* dimensions the same as embedding dimensions. To prevent information bottlenecks, we also ensure that as the network goes deeper, the embedding dimension of each transformer block should be non-decreasing.

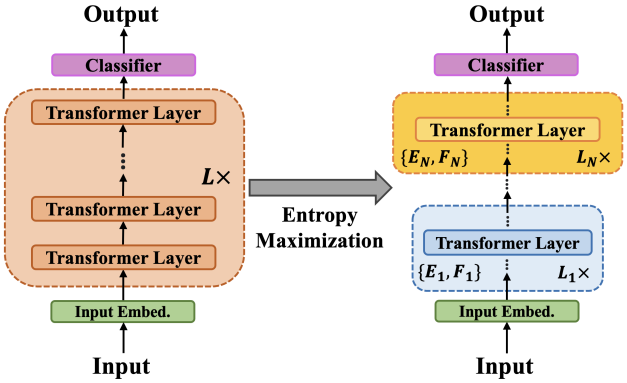

Figure 4: Our proposed adaptive block-wise transformer design. Left is the standard autoregressive transformer design, which consists of L homogeneous layers, and right is the optimal architecture design after entropy maximization, where there are N number of transformer blocks and each transformer block has adaptive width (E_i, R_i) and depth (L_i).

Search Process. To design a transformer model $f(\cdot)$ with N transformer blocks under a given computation budget C, we propose to optimize the parameters $\{E_j, F_j, L_j\}_{j=1,\dots,N}$ by solving a mathematical programming (MP) problem. E_j, F_j, and L_j denote the embedding dimension, FFN dimension, and number of layers in the j-th transformer block, respectively. For simplicity, we use ComputeCost as a general term to represent computation constraints such as parameter size, FLOPs, and latency.

The objective of the MP problem is to maximize the total transformer entropy, representing the expressiveness and effectiveness of the model while considering constraints on the computational cost. The MP problem is formulated as follows:

$$\max_{\{E_i,F_i,L_i\}} \sum_{j=1}^{N} L_j[(1-\frac{\beta L_j}{\log E_j})\widehat{H}_{\text{MHSA}} + (1-\frac{\beta L_j}{\log F_j})\widehat{H}_{\text{FFN}}]$$
$$\text{s.t.} \quad \text{ComputeCost}[f(\cdot)] \leq \mathcal{C}, \quad E_1 \leq \cdots \leq E_N \tag{15}$$

To solve this optimization problem, we employ an Evolu-

	Params	HellaSwag	WinoGrande	ARC-E	ARC-C	OpenbookQA	BoolQ	WIC	CB	WSC	RTE	PubmedQA	LogiQA	Avg.
Pythia-70M	70 M	0.269	**0.529**	0.335	0.214	0.272	0.589	0.486	0.339	0.365	0.523	0.409	0.266	0.383
Pythia-162M	162 M	0.292	0.492	0.373	0.231	0.264	0.571	0.500	0.446	0.365	**0.563**	0.544	0.269	**0.409**
Cerebras-111M	111 M	0.267	0.490	0.336	0.207	0.256	0.621	0.500	0.411	0.365	0.549	**0.552**	0.266	0.402
GPT-2-124M	124 M	**0.300**	0.516	0.382	0.230	0.272	0.554	0.492	0.410	**0.433**	0.531	0.430	0.245	0.400
OPT-125M	125 M	0.267	0.503	0.386	0.233	0.226	0.554	0.498	0.357	0.365	0.444	0.372	**0.286**	0.373
OPT-350M	331 M	0.283	0.523	**0.389**	0.233	**0.286**	0.618	0.500	**0.464**	0.365	0.542	0.414	0.280	0.408
MeRino-52M	52 M	0.267	0.507	0.327	0.212	0.242	0.541	**0.525**	**0.411**	0.413	0.502	0.377	0.276	0.383
MeRino-61M	61 M	0.273	0.510	0.336	0.209	0.248	0.610	0.502	0.375	0.365	0.534	0.484	0.255	0.392
MeRino-64M	64 M	0.274	0.528	0.341	**0.234**	0.267	**0.621**	0.505	0.393	0.375	0.545	0.540	0.278	0.408

Table 2: Detailed zero-shot learning results for MeRino and publicly available pre-trained LLMs.

	FLOPs (↓)	Latency (↓)	**WikiText-2**	**PTB**
Pythia-70M	100 G	95 ms	40.95	60.28
Pythia-162M	270 G	243 ms	23.52	36.02
Cerebras-111M	260 G	185 ms	36.93	51.89
GPT-2-124M	290 G	213 ms	25.19	33.95
OPT-125M	210 G	182 ms	23.62	29.02
OPT-350M	720 G	559 ms	**18.51**	**23.08**
MeRino-52M	60 G	48 ms	39.05	52.18
MeRino-61M	110 G	77 ms	34.24	34.11
MeRino-64M	160 G	114 ms	22.47	27.06

Table 3: Performance comparison on WikiText-2 (Merity et al. 2016) and Penn TreeBank (PTB) (Marcus, Santorini, and Marcinkiewicz 1993) for language modeling tasks.

tionary Algorithm (Reeves 2007). Note that Eq. (15) can be solved by any non-linear programming solver in principle. We choose EA due to its simplicity. Due to the page limit, detailed descriptions of the EA and mutation algorithm are omitted.

Experiments

In this section, we first describe detailed settings for search, training, and evaluation. Next, we report the main results of MeRino on various NLP tasks. Finally, we conduct ablation studies of each key component in our design methodology.

Experimental Settings

Datasets. For pre-training, we use the publicly available Pile dataset (Gao et al. 2020), which is pre-processed by removing duplication and tokenized using byte-level encoding. For evaluation, we evaluate our models across fourteen different downstream NLP tasks, namely WikiText-2 (Merity et al. 2016), Penn Treebank (PTB) (Marcus, Santorini, and Marcinkiewicz 1993), HellaSwag (Zellers et al. 2019), WinoGrande (Sakaguchi et al. 2019), OpenBookQA (Mihaylov et al. 2018), ARC (Clark et al. 2018), PubmedQA (Jin et al. 2019), LogiQA (Liu et al. 2020), and SuperGLUE (Wang et al. 2019) benchmark BoolQ, CB, WIC, WSC and RTE. We report the model performance using the perplexity and average accuracy for language modeling tasks, and zero-shot learning tasks, respectively.

Implementation Details. We follow the settings in (Zhang et al. 2022) and train each model from scratch for 600k steps with an effective batch size of 1024 and sequence length of 1024 on 8 NVIDIA H100 80 GB GPUs. For the learning rate schedule, we follow (Biderman et al. 2023) and adopt AdamW (Loshchilov and Hutter 2017) optimizer, with a starting learning rate of 6e-4, warm-up steps of 1000, and linear learning rate decay. For evaluation, we adopt the codebase of lm-evaluation-harness (Gao et al. 2021) for a fair comparison. We conduct latency experiments on NVIDIA Jetson Nano GPU 8GB and the inference latency is calculated with a batch size of 1 and sequence length of 128 averaged over 16 measurements.

Main Results

Comparison with SOTA LLMs. As our scope is mobile-friendly language models, we mainly compare pre-trained LLMs under 1B parameters. At the time of paper submission, the weights and evaluation for MobileLLM (Liu et al. 2024) are not released, thus we do not directly compare our method with it.

Table 2 and **??** report the detailed accuracy and latency results of our MeRino models and baseline models, such as GPT-2 (Radford et al. 2019), OPT (Zhang et al. 2022), Pythia (Biderman et al. 2023) and Cerebras-GPT (Dey et al. 2023). Compared to the OPT family, MeRino achieves superior accuracy with much less parameter size and FLOPs. Specifically, MeRino-64M obtains similar average accuracy as OPT-350M but with 82% and 78% reduction in model size and computation respectively. Above all, MeRino achieves an average inference speedup of 4.6× against OPT family models, respectively. Our smallest model, MeRino-52M achieves similar performance as Pythia-70M but with 2.0× faster runtime.

Comparison with Zero-shot NAS. We compare our methods against three zero-shot NAS approaches, namely TE-NAS (Chen, Gong, and Wang 2021), DSS-Score (Zhou et al. 2022) and LTS (Javaheripi et al. 2022). We conduct searches using the same FLOPs constraints (60/110/160 G), and report the downstream NLP performance of searched architectures. We also compare MeRino with naive scaling of the layer or embedding dimension for transformer-based LLMs. In Figure 5, we can see that under the same computation constraint, our entropy-driven design can produce much more capable language models than both the naive scaling method and NAS-based approaches.

We also compare the search efficiency of our method with

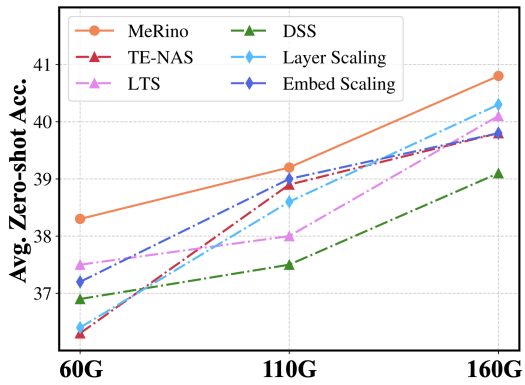

Figure 5: Performance comparison of MeRino, NAS-based methods, and naive scaling methods.

Method	Search Device	Search Time (h)	Energy Costs (Wh)	Average Acc.
TE-NAS	GPU*	1.2	300	0.389
Ours	CPU†	0.05	0.75	**0.408**

Table 4: Searching cost comparison of TE-NAS and our method. *: NVIDIA GTX 1080Ti GPU; †: NVIDIA Jetson Nano.

previous zero-cost NAS. Since the entropy-based score is based on pure analytical formulation, our method can directly run on target low-end hardware instead of GPU or TPU. In Table 4, we can see that it only takes about 0.05 hours to run on NVIDIA Jetson Nano, while TE-NAS consumes around 1.2 hours with a single NVIDIA GTX 1080Ti GPU. This further validates the high efficiency and scalability of our design methodology for diverse IoT devices.

Combining Quantization. We further apply the popular weight quantization method, GPTQ (Frantar et al. 2023) to our MeRino models. Figure 6 shows that MeRino can maintain almost identical performance when applying INT8 quantization, and yields only an accuracy drop of 1% in lower INT4 format.

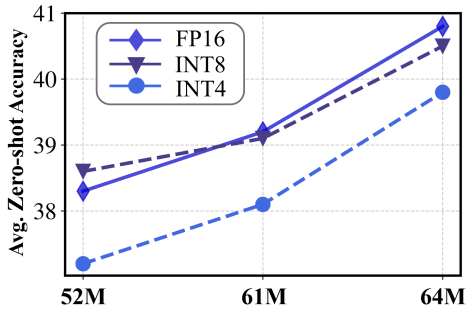

Figure 6: Comparison of our MeRino models in FP16, INT8 and INT4 precision.

Parameter Sharing	Params	FLOPs	Average Acc.
	59 M	60 G	0.381
✓	52 M		**0.383**
	79 M	110 G	**0.395**
✓	61 M		0.392
	100 M	160 G	0.403
✓	64 M		**0.408**

Table 5: Performance comparison of parameter sharing technique under three different FLOPs target.

Effectiveness Constraint γ	Weighted Entropy α	Params	FLOPs	Average Acc.
		62 M		0.360
✓		59 M	110 G	0.384
✓	✓	61 M		**0.392**

Table 6: Accuracy performance comparison of effectiveness constraint and weighted entropy.

Ablation Study

Impact of Parameter Sharing. We further study the impact of the parameter sharing technique for MeRino, and present the results in Table **??**. We can see that sharing parameters within the same transformer block helps improve both parameter efficiency and downstream zero-shot average accuracy. For instance, under the FLOPs budget of 160 G, block-wise parameter sharing reduces parameter size by 36% and improves downstream performances by 0.2%. On average, with the parameter sharing technique, MeRino obtains 23% reduction in parameter size and 0.5% accuracy gain.

Impact of γ and α. As shown in Table **??**, effectiveness constraint γ plays a key role in helping our entropy-driven framework design more capable and trainable models. When using effectiveness constraint γ, the final searched language model obtains +2.4% average accuracy gain. Similarly, we can see that using weighted entropy helps improve the average zero-shot accuracy by 0.8%.

Conclusion

In this paper, we propose a novel design framework aiming to generate efficient autoregressive language models for mobile devices under 100 M parameters at nearly zero cost. Leveraging the Maximum Entropy Principle, we formulate a constrained mathematical programming problem and optimize the network architecture by maximizing the entropy of transformer decoders under given computational budgets. Our designed model, MeRino can achieve comparable performance against both state-of-the-art pre-trained LLMs and NAS-designed models with significant improvement in model size reduction and inference runtime speedup on NVIDIA Jetson Nano.

Acknowledgments

This work was sponsored in part by the U.S. National Science Foundation (NSF) under Grants 1907765 and 2400014. The authors would like to thank the anonymous AAAI reviewers for their constructive feedback to improve this work.

References

Anthropic. 2023. Introducting Claude. https://www.anthropic.com/news/introducing-claude.

Apple. 2024. Apple Intelligence. https://www.apple.com/apple-intelligence/.

Ashkboos, S.; Croci, M. L.; do Nascimento, M. G.; Hoefler, T.; and Hensman, J. 2024. SliceGPT: Compress Large Language Models by Deleting Rows and Columns. *ArXiv*, abs/2401.15024.

Biderman, S. R.; Schoelkopf, H.; Anthony, Q. G.; Bradley, H.; O'Brien, K.; Hallahan, E.; Khan, M. A.; Purohit, S.; Prashanth, U. S.; Raff, E.; Skowron, A.; Sutawika, L.; and van der Wal, O. 2023. Pythia: A Suite for Analyzing Large Language Models Across Training and Scaling.

Brown, T. B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; Agarwal, S.; Herbert-Voss, A.; Krueger, G.; Henighan, T. J.; Child, R.; Ramesh, A.; Ziegler, D. M.; Wu, J.; Winter, C.; Hesse, C.; Chen, M.; Sigler, E.; Litwin, M.; Gray, S.; Chess, B.; Clark, J.; Berner, C.; McCandlish, S.; Radford, A.; Sutskever, I.; and Amodei, D. 2020. Language Models are Few-Shot Learners. *ArXiv*, abs/2005.14165.

Chan, K. H. R.; Yu, Y.; You, C.; Qi, H.; Wright, J.; and Ma, Y. 2021. ReduNet: A White-box Deep Network from the Principle of Maximizing Rate Reduction. *ArXiv*, abs/2105.10446.

Chelba, C.; Mikolov, T.; Schuster, M.; Ge, Q.; Brants, T.; Koehn, P. T.; and Robinson, T. 2013. One billion word benchmark for measuring progress in statistical language modeling. In *Interspeech*.

Chen, W.; Gong, X.; and Wang, Z. 2021. Neural Architecture Search on ImageNet in Four GPU Hours: A Theoretically Inspired Perspective. *ArXiv*, abs/2102.11535.

Clark, P.; Cowhey, I.; Etzioni, O.; Khot, T.; Sabharwal, A.; Schoenick, C.; and Tafjord, O. 2018. Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. *ArXiv*, abs/1803.05457.

Dey, N.; Gosal, G. S.; Chen, Z.; Khachane, H.; Marshall, W.; Pathria, R.; Tom, M.; and Hestness, J. 2023. Cerebras-GPT: Open Compute-Optimal Language Models Trained on the Cerebras Wafer-Scale Cluster.

Frantar, E.; and Alistarh, D. 2023. SparseGPT: Massive Language Models Can Be Accurately Pruned in One-Shot. *ArXiv*, abs/2301.00774.

Frantar, E.; Ashkboos, S.; Hoefler, T.; and Alistarh, D. 2023. GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers. *International Conference on Learning Representations (ICLR)*.

Gao, L.; Biderman, S. R.; Black, S.; Golding, L.; Hoppe, T.; Foster, C.; Phang, J.; He, H.; Thite, A.; Nabeshima, N.; Presser, S.; and Leahy, C. 2020. The Pile: An 800GB Dataset of Diverse Text for Language Modeling. *ArXiv*, abs/2101.00027.

Gao, L.; Tow, J.; Biderman, S.; Black, S.; DiPofi, A.; Foster, C.; Golding, L.; Hsu, J.; McDonell, K.; Muennighoff, N.; Phang, J.; Reynolds, L.; Tang, E.; Thite, A.; Wang, B.; Wang, K.; and Zou, A. 2021. A framework for few-shot language model evaluation.

Hu, S.; Tu, Y.; Han, X.; He, C.; Cui, G.; Long, X.; Zheng, Z.; Fang, Y.; Huang, Y.; Zhao, W.; Zhang, X.; Thai, Z. L.; Zhang, K.; Wang, C.; Yao, Y.; Zhao, C.; Zhou, J.; Cai, J.; Zhai, Z.; Ding, N.; Jia, C.; Zeng, G.; Li, D.; Liu, Z.; and Sun, M. 2024. MiniCPM: Unveiling the Potential of Small Language Models with Scalable Training Strategies. *ArXiv*, abs/2404.06395.

Javaheripi, M.; Shah, S.; Mukherjee, S.; Religa, T. L.; Mendes, C. C. T.; de Rosa, G.; Bubeck, S.; Koushanfar, F.; and Dey, D. 2022. LiteTransformerSearch: Training-free On-device Search for Efficient Autoregressive Language Models. *ArXiv*, abs/2203.02094.

Jaynes, E. T. 1957. Information Theory and Statistical Mechanics. *Physical Review*, 106: 620–630.

Jin, Q.; Dhingra, B.; Liu, Z.; Cohen, W. W.; and Lu, X. 2019. PubMedQA: A Dataset for Biomedical Research Question Answering. In *Conference on Empirical Methods in Natural Language Processing*.

Lan, Z.; Chen, M.; Goodman, S.; Gimpel, K.; Sharma, P.; and Soricut, R. 2019. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. *ArXiv*, abs/1909.11942.

Levine, Y.; Wies, N.; Sharir, O.; Bata, H.; and Shashua, A. 2020. The Depth-to-Width Interplay in Self-Attention. *arXiv: Learning*.

Liu, J.; Cui, L.; Liu, H.; Huang, D.; Wang, Y.; and Zhang, Y. 2020. LogiQA: A Challenge Dataset for Machine Reading Comprehension with Logical Reasoning. In *International Joint Conference on Artificial Intelligence*.

Liu, Z.; Zhao, C.; Iandola, F. N.; Lai, C.; Tian, Y.; Fedorov, I.; Xiong, Y.; Chang, E.; Shi, Y.; Krishnamoorthi, R.; Lai, L.; and Chandra, V. 2024. MobileLLM: Optimizing Sub-billion Parameter Language Models for On-Device Use Cases. *ArXiv*, abs/2402.14905.

Loshchilov, I.; and Hutter, F. 2017. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*.

Marcus, M. P.; Santorini, B.; and Marcinkiewicz, M. A. 1993. Building a Large Annotated Corpus of English: The Penn Treebank. *Comput. Linguistics*, 19: 313–330.

Merity, S.; Xiong, C.; Bradbury, J.; and Socher, R. 2016. Pointer Sentinel Mixture Models. *ArXiv*, abs/1609.07843.

Microsoft. 2024. Introducing Copilot+ PCs. https://blogs.microsoft.com/blog/2024/05/20/introducing-copilot-pcs/.

Mihaylov, T.; Clark, P.; Khot, T.; and Sabharwal, A. 2018. Can a Suit of Armor Conduct Electricity? A New Dataset

for Open Book Question Answering. In *Conference on Empirical Methods in Natural Language Processing*.

OpenAI. 2022. Introducting ChatGPT. https://openai.com/blog/chatgpt.

Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; and Sutskever, I. 2019. Language Models are Unsupervised Multitask Learners.

Reeves, C. R. 2007. Evolutionary computation: a unified approach. *Genetic Programming and Evolvable Machines*, 8: 293–295.

Roberts, D. A.; Yaida, S.; and Hanin, B. 2021. The Principles of Deep Learning Theory. *ArXiv*, abs/2106.10165.

Sakaguchi, K.; Bras, R. L.; Bhagavatula, C.; and Choi, Y. 2019. WINOGRANDE: An Adversarial Winograd Schema Challenge at Scale. *Commun. ACM*, 64: 99–106.

Saxe, A. M.; Bansal, Y.; Dapello, J.; Advani, M. S.; Kolchinsky, A.; Tracey, B. D.; and Cox, D. D. 2018. On the information bottleneck theory of deep learning. *Journal of Statistical Mechanics: Theory and Experiment*, 2019.

Soham. 2024. The Cost of Inference: Running the Models. https://tinyml.substack.com/p/the-cost-of-inference-running-the.

Sun, Z.; Lin, M.; Sun, X.; Tan, Z.; Li, H.; and Jin, R. 2021. MAE-DET: Revisiting Maximum Entropy Principle in Zero-Shot NAS for Efficient Object Detection. In *International Conference on Machine Learning*.

Touvron, H.; Lavril, T.; Izacard, G.; Martinet, X.; Lachaux, M.-A.; Lacroix, T.; Rozière, B.; Goyal, N.; Hambro, E.; Azhar, F.; Rodriguez, A.; Joulin, A.; Grave, E.; and Lample, G. 2023. LLaMA: Open and Efficient Foundation Language Models. *ArXiv*, abs/2302.13971.

Vaswani, A.; Shazeer, N. M.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention is All you Need. *ArXiv*, abs/1706.03762.

Wang, A.; Pruksachatkun, Y.; Nangia, N.; Singh, A.; Michael, J.; Hill, F.; Levy, O.; and Bowman, S. R. 2019. SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems. *ArXiv*, abs/1905.00537.

Wu, C.-J.; Raghavendra, R.; Gupta, U.; Acun, B.; Ardalani, N.; Maeng, K.; Chang, G.; Aga, F.; Huang, J.; Bai, C.; et al. 2022. Sustainable ai: Environmental implications, challenges and opportunities. *Proceedings of Machine Learning and Systems*, 4: 795–813.

Xu, J.; Tan, X.; Luo, R.; Song, K.; Li, J.; Qin, T.; and Liu, T.-Y. 2021. NAS-BERT: Task-Agnostic and Adaptive-Size BERT Compression with Neural Architecture Search. *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*.

Zellers, R.; Holtzman, A.; Bisk, Y.; Farhadi, A.; and Choi, Y. 2019. HellaSwag: Can a Machine Really Finish Your Sentence? In *Annual Meeting of the Association for Computational Linguistics*.

Zhang, S.; Roller, S.; Goyal, N.; Artetxe, M.; Chen, M.; Chen, S.; Dewan, C.; Diab, M.; Li, X.; Lin, X. V.; Mihaylov, T.; Ott, M.; Shleifer, S.; Shuster, K.; Simig, D.; Koura, P. S.; Sridhar, A.; Wang, T.; and Zettlemoyer, L. 2022. OPT: Open Pre-trained Transformer Language Models. *ArXiv*, abs/2205.01068.

Zhang, Z. A.; Sheng, Y.; Zhou, T.; Chen, T.; Zheng, L.; Cai, R.; Song, Z.; Tian, Y.; Ré, C.; Barrett, C. W.; Wang, Z.; and Chen, B. 2023. H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models. *ArXiv*, abs/2306.14048.

Zhao, Y.; Wu, D.; and Wang, J. 2024. ALISA: Accelerating Large Language Model Inference via Sparsity-Aware KV Caching. *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 1005–1017.

Zhou, Q.; Sheng, K.; Zheng, X.; Li, K.; Sun, X.; Tian, Y.; Chen, J.; and Ji, R. 2022. Training-free Transformer Architecture Search. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 10884–10893.