Instruction-Following Pruning for Large Language Models

Bairu Hou [1 2 *] **Qibin Chen** [1] **Jianyu Wang** [1] **Guoli Yin** [1] **Chong Wang** [1] **Nan Du** [1]
Ruoming Pang [1] **Shiyu Chang** [2] **Tao Lei** [1]

Abstract

With the rapid scaling of large language models (LLMs), structured pruning has become a widely used technique to learn efficient, smaller models from larger ones, delivering superior performance compared to training similarly sized models from scratch. In this paper, we move beyond the traditional *static pruning* approach of determining a fixed pruning mask for a model, and propose a *dynamic approach* to structured pruning. In our method, the pruning mask is input-dependent and adapts dynamically based on the information described in a user instruction. Our approach, termed "instruction-following pruning", introduces a sparse mask predictor that takes the user instruction as input and dynamically selects the most relevant model parameters for the given task. To identify and activate effective parameters, we jointly optimize the sparse mask predictor and the LLM, leveraging both instruction-following data and the pre-training corpus. Experimental results demonstrate the effectiveness of our approach on a wide range of evaluation benchmarks. For example, our 3B activated model improves over the 3B dense model by 5-8 points of absolute margin on domains such as math and coding, and rivals the performance of a 9B model.

1. Introduction

Structured pruning techniques have become a widely adopted method for reducing the inference cost of large language models (Wang et al., 2020; Sreenivas et al., 2024; Muralidharan et al., 2024; Meta AI, 2024). These methods typically optimize a binary mask over language model parameters to minimize either language modeling or task-specific loss (Xia et al., 2024; Sreenivas et al., 2024; Meta AI, 2024). Once the mask is optimized, the resulting mask is fixed, allowing deployment of a smaller, pruned model. However, the fixed nature of the pruned model poses challenges in real-world inference scenarios, where tasks can vary significantly, for instance, coding, mathematics, or domain-specific requirements, each demanding distinct skills and knowledge from the original language model. A static pruned model may struggle to balance inference efficiency with high performance across diverse tasks.

Given this, we explore a paradigm shift from *static* pruning masks to *dynamic* ones, addressing the central question:

Can LLMs learn to select the most suited parameters based on the task description?

We aim to automatically generate input-specific pruning masks tailored to the tasks described in user prompts. This dynamic, context-aware pruning mechanism enables the language model to perform inference using only the parameters necessary for the task, offering a compelling balance between efficiency and expressivity compared to using a static dense model. Moreover, because the parameters are selected and fixed, our method avoids reloading new parameters during the decoding process. This design choice contrasts with other dynamic methods such as contextual sparsity (Liu et al., 2023; Zhou et al., 2024) and mixture-of-experts (Lepikhin et al., 2020; Fedus et al., 2022; Dai et al., 2024), which load different parameters at each decoding step, leading to significant weight loading costs. Our design is particularly suited for on-device models (*e.g.* smartphones and laptops), where the inference typically samples a few responses given the same user query (or the same task). In this case, the same activated parameters are selected and cached as a dense model, therefore achieving the same speedup as the static pruning.

To address the central question, we present Instruction-Following Pruning (IFPRUNING), a method that integrates a sparsity predictor with the language model to dynamically generate input-dependent pruning masks, as illustrated in Figure 1. Specifically, we focus on structured pruning of the feed-forward neural network layers, where entire rows or columns of the weight matrices are pruned (Xia et al., 2024; Gunter et al., 2024; Sreenivas et al., 2024). The user

*Work done while interning at Apple. [1]Apple AI/ML [2]UC Santa Barbara. Correspondence to: Bairu Hou <bairu@ucsb.edu>, Tao Lei <tao_lei2@apple.com>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025.

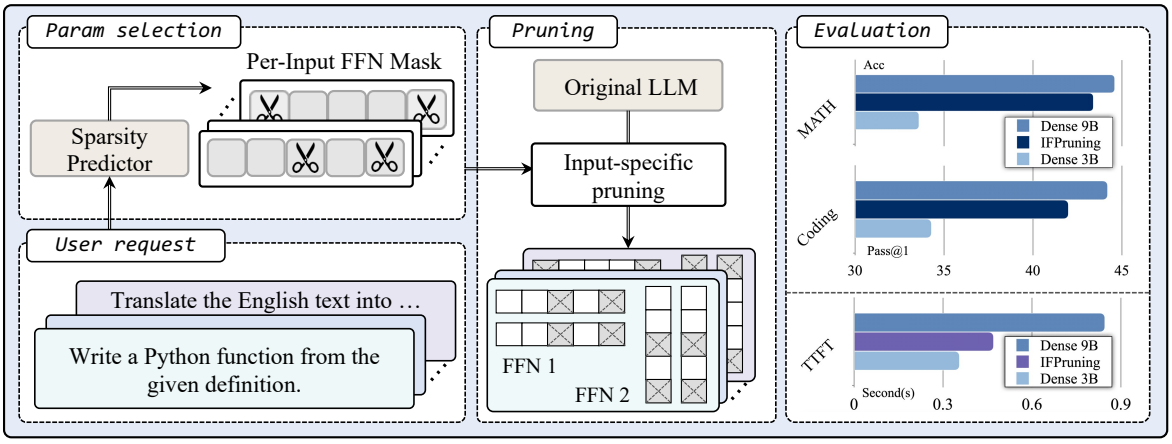

Figure 1. Overview of IFPRUNING. (Left) For each given prompt, the sparsity predictor (much smaller than the LLM) determines which rows and columns of the FFN matrices should be activated. (Middle) The LLM is then pruned accordingly and uses the selected parameters to perform inference for that specific prompt. (Right) By pruning a 9B LLM to 3B for each input, IFPRUNING significantly outperforms the dense 3B model and achieves performance levels close to the dense 9B model. It also achieves nearly the same inference latency as the dense 3B model, as measured by time-to-first-token (TTFT).

prompt is first passed into the sparsity predictor, which assigns importance scores to the rows and columns of each feed-forward network layer. These scores are then transformed into differentiable masks using the SOFTTOPK operator (Ainslie et al., 2023), to achieve a predefined number of sparsity (*e.g.*, reducing a 9B language model to 3B active parameters). The resulting masks are applied to the language model, in which the feed-forward layers are pruned using the masks.

During training, the differentiable mask generation mechanism allows us to jointly optimize both the sparsity predictor and the language model by minimizing the next-token prediction loss. We employ effective training strategies that leverage both pre-training and supervised fine-tuning data. At test time, only the selected parameters are activated for inference. Parameter selection can be performed either *per-input* or *per-task*: the input prompt can directly be used for parameter selection (Section 4.2), or a predefined task prompt can be used to select parameters shared across multiple inputs within the same task (Section 4.3).

We validate IFPRUNING through comprehensive experiments across diverse tasks in Section 4. Specifically, we fine-tune pre-trained language models of varying sizes (6B, 9B, and 12B parameters) using IFPRUNING and prune them to activate only 3B parameters. In particular, IFPRUNING consistently outperforms 3B dense models across tasks such as math, coding, tool use, MMLU (Hendrycks et al., 2021a) and AlpacaEval (Dubois et al., 2024). For example, when dynamically pruning the 9B model to 3B, our method improves over the 3B dense model by 8% on coding tasks and by 5% on math benchmarks, incurring only marginal performance degradation compared to the unpruned 9B model.

We conduct further analysis to better understand the pruning decisions. Specifically, we observe that instructions requiring similar skills or domain knowledge yield highly homogeneous pruning patterns. Inspired by this analysis, we explore per-task pruning in Section 4.3, where a single task prompt generates shared masks for all test instances within the same task. Results show that per-task pruning maintains robust performance while further reducing data loading overhead.

We also show that IFPRUNING can significantly improve the LLM inference efficiency in Section 4.4. Compared to the full model, IFPRUNING reduces the time-to-first token by up to 57% and the generate time by up to 41%. In addition, the overhead introduced by dynamic pruning and parameter caching is negligible, adding less than 0.1 seconds per example and accounting for only 1–2% of the total generation time.

2. Related Work

In this section, we provide an overview of prior research that closely relates to and partially motivates our work, including model pruning, contextual sparsity, and mixture-of-experts.

Model pruning Pruning has been extensively studied to compress neural networks and improve their efficiency (Han et al., 2015; Zhu & Gupta, 2017). Previous work has explored different pruning techniques for both unstructured pruning (Narang et al., 2017; Frankle & Carbin, 2018; Li et al., 2020; Chen et al., 2020) and structured pruning (Wen et al., 2016; Voita et al., 2019; Louizos et al., 2018; Wang et al., 2020). As structured pruning removes entire compo-

nents in the model such as channels, attention heads, and feed-forward neural network intermediate dimensions, it is more hardware-friendly than unstructured pruning to compress the large models.

Various methods have been proposed for structured pruning of LLMs (Yang et al., 2024b; Kim et al., 2024; Kurtić et al., 2024; Dery et al., 2024). LLM-PRUNER (Ma et al., 2023) adopt the gradient information to find unimportant components in LLMs and remove them. SLICEGPT transforms each weight matrix in transformer blocks into a smaller one by applying orthogonal transformations to reduce the embedding dimensions of weight matrices. SHORTGPT (Men et al., 2024) proposes to identify and remove those less important layers, where the layer importance is measured by the similarity between inputs and outputs of that layer. In comparison, other optimization-based methods directly learn the parameter masks. For example, SHEARED LLAMA (Xia et al., 2024) use the HARDCONCRETE masking (Louizos et al., 2018; Wang et al., 2020) to generate differentiable masks and optimize the model and masks on pre-training data. Our method also directly optimize the sparsity predictor and the LLM, and we further extend static pruning to input-dependent pruning.

Contextual sparsity Our approach is also directly motivated by the contextual sparsity of LLMs (Liu et al., 2023; Akhauri et al., 2024; Lee et al., 2024). Previous work has identified the existence of input-dependent sub-networks (*e.g.*, attention heads and MLP parameters) within ReLU-based LLMs that can generate approximately the same output as the full model for an input. By predicting such sparsity patterns at each decoding step, we can achieve a favorable balance between accuracy and speedup. But state-of-the-art LLMs (Dubey et al., 2024; Liu et al., 2024a; Yang et al., 2024a) design MLP blocks based on more complex non-linear activation functions such as SwiGLU (Shazeer, 2020), SiLU (Elfwing et al., 2018; Ramachandran et al., 2017) and GELU (Hendrycks & Gimpel, 2016) that do not inherently induce sparsity (Mirzadeh et al., 2023; Song et al., 2024). Therefore, directly predicting the sparsity patterns can lead to significant performance degradation (Zhou et al., 2024; Dong et al., 2024). In comparison, we co-optimize the sparsity predictor and the LLM with non-ReLU activation functions to achieve better contextual sparsity with minimum performance degradation. Also, most contextual sparsity methods require predicting sparsity and loading different parameters at each decoding step. Our method eliminates this overhead by selecting the parameters based on the input or task description before decoding starts. The selected parameters are fixed for the entire decoding process, avoiding the parameter reloading cost.

Mixture-of-experts Mixture-of-Experts (MoE) have emerged as a popular architecture for scaling LLMs while managing inference costs (Lepikhin et al., 2020; Du et al., 2022; Fedus et al., 2022; Zhou et al., 2022; Dai et al., 2024; Liu et al., 2024b). These models organize every FFN layer into multiple large FFN blocks referred to as experts, and selectively activate a few experts for each input token via a routing mechanism (Lepikhin et al., 2020; Zoph et al., 2022; Sun et al., 2024). Our method share the same spirit as MoE by dynamically activating a subset of parameters. However, our method selects the activated parameters given the input prompt, and reuses the same activated parameters during decoding. Although this choice loses the flexibility of using different parameters per token, it significantly reduces weight loading costs for decoding. In this regard, our model is a sparse model designed for on-device scenarios where both memory and computational resources are constrained. Another difference is that our method performs more fine-grained selection of parameters by activating or pruning each FFN dimension independently, which enhances model expressivity.

3. Method

In this section, we elaborate on the details of IFPRUNING, including the architecture design, data mixture, and training method. We focus on pruning the feed forward blocks (FFNs) in this work, but our method can be easily extend to pruning other components such as attention heads.

3.1. Overview of Structured Pruning

Denote the hidden dimension of the LLM as d, the intermediate dimension of the FFN blocks as d_{ffn}, the input length as n, and $X \in \mathbb{R}^{n\times d}$ as the input of a transformer FFN block $F_{\text{ffn}}(\cdot)$. The goal of our structured pruning method is to reduce the FFN intermediate dimension from d_{ffn} to t_{ffn}. Without loss of generality, consider a standard FFN block defined as

$$F_{\text{ffn}}(X) = \text{FF}_2(\text{FF}_1(X)) = \sigma(XW_1)W_2, \qquad (1)$$

where $W_1 \in \mathbb{R}^{d\times d_{\text{ffn}}}$, $W_2 \in \mathbb{R}^{d_{\text{ffn}}\times d}$ are weight matrices, and σ is the non-linear activation function. The structured pruning of the FFN block can be expressed as applying a mask variable $\boldsymbol{m} \in \{0,1\}^{d_{\text{ffn}}}$ to the output of the first linear transformation

$$F_{\text{ffn}}(X, \boldsymbol{m}) = \text{FF}_2(\text{FF}_1(X) \odot \boldsymbol{m}). \qquad (2)$$

where $\odot$ is an element-wise multiplication between $\boldsymbol{m}$ and each row of $\text{FF}_1(X)$. For each dimension of $\boldsymbol{m}$, $m_i = 0$ indicates that the i-th column of W_1 and i-th row of W_2 are pruned. This is because the output $F_{\text{ffn}}(X, \boldsymbol{m})$ is equivalent to the output of the FFN layer after we prune the i-th column of W_1 and i-th row of W_2. Here $\boldsymbol{m}$ satisfies the

sparsity constraint $\sum_i m_i = t_{\text{ffn}}$, where t_{ffn} is the target intermediate dimension of the FFN blocks after pruning.

3.2. Architecture

As shown in Figure 1, our architecture comprises two key components: a sparsity predictor and a dense LLM to be dynamically pruned. For any given user prompt, the sparsity predictor generates masks that are applied to the LLM backbone, pruning the corresponding rows and columns of the FFN blocks.

Sparsity predictor The sparsity predictor consists of two modules: ❶ a much smaller LLM backbone to extract the features of user prompts and ❷ a mask prediction head. Specifically, the LLM backbone takes the prompt $\boldsymbol{x} = (x_1, \ldots, x_n)$ as input with length n, and we use the hidden states of the last token x_n in the last layer to represent the prompt. The mask prediction head is a two-layer MLP, which predicts the masks given the prompt presentations. The output of the FFN mask prediction head is the masking score $\boldsymbol{z} \in \mathbb{R}^{L \times d_{\text{ffn}}}$, where L is the number of layers of the LLM. We include more details about the architecture of the sparsity predictor in Appendix A.1.

Given the predicted masking score $\boldsymbol{z}$, a mask generation operator will be applied to $\boldsymbol{z}$ to convert it to the mask $\boldsymbol{m} \in [0,1]^{L \times d_{\text{ffn}}}$, which contains t_{ffn} nonzero elements. In this paper, we use the SoftTopK (Lei et al., 2023; Ainslie et al., 2023) algorithm to generate a differentiable $\boldsymbol{m}$, but we also acknowledge that other algorithms such as the HardConcrete masking (Louizos et al., 2018; Wang et al., 2020) are also applicable. Particularly, given the FFN masking score $\boldsymbol{z}$, SoftTopK converts it to masks $\boldsymbol{m}$ via:

$$\boldsymbol{\lambda}^{(i)} = g(\boldsymbol{z}^{(i)}),\ \boldsymbol{m}^{(i)} = \boldsymbol{\lambda}^{(i)} \odot \text{Top}(\boldsymbol{\lambda}^{(i)}, t_{\text{ffn}}). \quad (3)$$

Here $\boldsymbol{z}^{(i)}$, $\boldsymbol{\lambda}^{(i)}$ and $\boldsymbol{m}^{(i)}$ represent the i-th row of each matrix, $g(\cdot) : \mathbb{R}^{d_{\text{ffn}}} \to [0,1]^{d_{\text{ffn}}}$ is a normalization function, and $\text{Top}(\cdot, t_{\text{ffn}}) \in \{0,1\}^{d_{\text{ffn}}}$ is an indicator function that returns a binary mask indicating the top-k values in λ. The normalization function $g(\cdot)$ ensures that λ satisfies the sparsity constraint, *i.e.*, $\sum_k \lambda_k^{(i)} = t_{\text{ffn}}$, where t_{ffn} is the target size of the FFN layers. More details of SoftTopK can be found in the previous work (Lei et al., 2023; Ainslie et al., 2023).

Masked LLM During training, the LLM takes the masks $\boldsymbol{m}$ as an additional input and prune its FFN blocks. We use standard next token prediction loss computed over tokens within a training batch, and we co-optimize the LLM and the sparsity predictor.

3.3. Model Training

The training of IFPRUNING incorporates two stages. We first perform continued pre-training in which we initialize our model using a pretrained dense model, and then perform supervised fine-tuning (SFT) on instruction-following data. In what follows, we elaborate on the details of the two training stages.

Continued pre-training Learning to select input-specific sub-networks may require a lot of training data. Instead of directly training the models on the SFT data only, we first use pre-training data to jointly optimize the sparsity predictor and masked LLM. Specifically, denoting the input text as $\boldsymbol{x} = (x_1, \ldots, x_n)$, we split it into K consecutive chunks with fixed size:

$$\boldsymbol{x}^{(k)} = x_{(k-1)s+1}, \ldots, x_{ks},\ k = 1, \ldots, K, \quad (4)$$

where $s = n/K$ is the fixed size of each chunk. We then use the each chunk to select parameters of the LLM for the next token predictions in the next chunk, *i.e.*,

$$\mathcal{L} = \sum_{k=1}^{K-1} \sum_{x_i \in \boldsymbol{x}^{(k+1)}} \ell\left[f(\boldsymbol{x}_{<i}; \boldsymbol{\theta}, \boldsymbol{m}^{(k)}), x_i\right], \quad (5)$$

where $\boldsymbol{\theta}$ refers to the parameters of the LLM, $\boldsymbol{m}^{(k)}$ is the predicted mask based on chunk $\boldsymbol{x}^{(k)}$, $f(\boldsymbol{x}_{<i}; \boldsymbol{\theta}, \boldsymbol{m}^{(k)})$ is the next token prediction distribution from the LLM with $\boldsymbol{m}^{(k)}$ applied, and $\ell(\cdot)$ is the Cross-Entropy loss. Since the chunks are consecutive, the sparsity predictor can learn to utilize the contextual information of each chunk to predict which parameters of the LLM are best suited for the next token prediction in the next text chunk. Because both the sparsity predictor and LLM are co-optimized in this stage, it provides a good initialization for the fine-tuning stage.

Supervised fine-tuning Starting from the models after the first stage, we train the sparsity predictor and the LLM on a supervised fine-tuning dataset that contains several million examples. During training, the input prompts will be fed into the sparsity predictor which predicts the masks for each input. The LLM is then masked and optimized to predict the target outputs conditioned on the input prompts. Our SFT data contains a diverse set of prompts to predict the sub-networks. Some prompts specify the task with a task description and an input, while others include few-shot examples along with the input. Lastly, many examples only contain a task description, like "write a comprehensive blog post about the top 10 most eco-friendly cities in the world.".

For multi-turn conversational data, we only use the first human message as the prompt for sub-network selection. During training, with the exception of removing all instances of personal data, all these prompts are fed directly into the

sparsity predictor without any additional processing. This approach maximizes flexibility during inference, allowing the predictor to generate a sub-network regardless of the prompt format during inference. The training objective follows the standard SFT approach, namely minimizing the cross-entropy loss on the target outputs. Through this process, the model learns to selectively activate the most suited parameters for different input examples.

4. Experiment

In this section, we conduct empirical evaluations to assess the effectiveness of our proposed method.

4.1. Experiment Setup

Dataset and backbone models Our models are trained on an internal SFT dataset with several million examples. We also follow the setup used in TÜLU 2 and sample additional 800K examples from the FLAN-V2 collection (Chung et al., 2024) to enhance task prompt diversity. The experiments are conducted using a series of pre-trained LLMs. Particularly, for the sparsity prediction component, we initialize it with a 302M model that has been pre-trained on web-crawled data. To test the performance of our method across various model scales, we separately train three different models that use 6B, 9B, and 12B parameters, respectively, in the masked LLM component. For all these models, our approach activates 3B parameters (and prunes the rest of the parameters). More details about the model architecture are given in Appendix A.1.

Comparison baselines We compare our method with the following models: ❶ DENSE-3B. We compare with a 3B dense LLM that is trained using twice as many pretraining tokens compared to our models. ❷ PRUNING+DISTILL. We also compare our method with static pruning approaches. Similar to recent work such as Sheared LLaMA (Xia et al., 2024), LLAMA 3.2 (Meta AI, 2024), and MINITRON(Sreenivas et al., 2024), we include a baseline where a 3B dense LLM is pruned and distilled from a larger pre-trained LLM. we first prune the larger LLM into a dense 3B model by learning masks on the FFN layers similar to Sheared LLaMA (Xia et al., 2024). After pruning, the model undergoes further continuous pre-training through knowledge distillation (Hinton, 2015), using a larger dense model with 12B parameters as the teacher model. Therefore, this approach serves as a stronger baseline compared to models using pruning alone. More details of the knowledge distillation can be found in Appendix A.2. ❸ DENSE-9B. We also include a 9B dense model without pruning as a reference for upper-bound performance.

Implementation We use the AXLearn (Apple, 2023) framework and JAX (Bradbury et al., 2018) for model training. Following the previous work (Dubey et al., 2024), all the pre-trained model used in our experiments are achieved by performing two-stage pre-training. All models are pre-trained with a batch size of 2048 and a total number of 5T tokens, except that the DENSE-3B is trained for 9T tokens. The SFT training for the baselines and our method is performed with a batch size of 1024 for 60k training steps. We use the same pre-train and SFT data mixture for all models.

Evaluation configurations We include the following tasks for evaluation:

- Instruction-following. We include `IFEval` (Zhou et al., 2023), `AlpacaEval 2.0` (Dubois et al., 2024), and `Arena-Hard-Auto` (Li et al., 2024) for evaluation. We report the prompt-level and instruction-level accuracy on IFEval, the length-controlled win rate on AlpacaEval 2.0, and win rate on Arena-Hard-Auto.

- Coding tasks. We evaluate the pass@1 performance on `HumanEval-python` (Chen et al., 2021), `mbpp` (Austin et al., 2021), and `MultiPL-E` (Cassano et al., 2022). For MultiPL-E benchmark, we use the Swift subset for evaluation.

- Math. We use `GSM8K` (Cobbe et al., 2021) and `MATH` (Hendrycks et al., 2021b) to evaluate the math capabilities of LLMs. we report the accuracy with few-shot examples on both datasets (8-shot for GSM8K and 4-shot for MATH).

- Core Text. We include a set of tasks to evaluate the model's core capabilities of natural language understanding, scientific knowledge, and reasoning. We report the zero-shot performance on `ARC-challenge` (Clark et al., 2018), `ARC-easy` (Clark et al., 2018), `HellaSwag` (Zellers et al., 2019), `WinoGrande` (Sakaguchi et al., 2021), `PiQA` (Bisk et al., 2020), `LAMBADA-OpenAI` (Paperno et al., 2016), and `SciQ` (Welbl et al., 2017).

- MMLU (Hendrycks et al., 2021a). We evaluate the 5-shot performance and report the multiple-choice accuracy.

- Tool use. We evaluate the tool use performance on `MMAU` (Yin et al., 2024) and report the performance on Tool Execution and Tool Planning.

We mainly use `LM-Evaluation-Harness` (Gao et al., 2021) to evaluate the tasks, with the exception of instruction-following and tool-use tasks, which are based on their official implementations.

Table 1. Performance comparison between IFPRUNING and other dense LLMs. PRUNING+DISTILL 3B is first pruned from larger model into 3B parameters and undergoes continuous pre-training with knowledge distillation using a 12B LLM as the teacher model. Three versions of IFPRUNING are included which activate 3B parameters of LLMs with 6B, 9B, and 12B parameters. The **best** results are highlighted in **bold** and the second-best results are underlined. DENSE-9B is included for reference.

Category	Dataset	DENSE 3B	PRUNING+ DISTILL 3B	IFPRUNING 6B→3B	IFPRUNING 9B→3B	IFPRUNING 12B→3B	DENSE 9B
Instruction Following	IFEval-Instruction	85.0	**86.7**	83.9	85.9	85.3	87.5
	IFEval-Prompt	77.8	**80.8**	77.6	78.9	78.6	81.7
	AlpacaEval 2.0	27.3	30.0	29.0	31.3	**32.5**	38.6
	Arena-Hard-Auto	15.8	16.4	18.0	18.6	**19.8**	24.8
Coding	HumanEval	35.2	37.1	41.0	42.4	**43.3**	46.5
	MultiPL-E	39.0	37.9	37.6	41.8	**43.0**	44.0
	MBPP	28.8	38.0	37.4	41.8	**42.8**	42.2
	Average	34.3	37.7	38.7	42.0	**43.0**	44.2
Tool Use	Tool Execution	74.8	74.4	**75.8**	75.7	73.9	76.5
	Tool Planning	25.2	17.5	36.6	**45.4**	37.5	46.5
Math	GSM8K	69.3	70.0	**72.2**	72.0	70.2	75.4
	MATH	31.8	32.7	36.2	36.7	**37.1**	37.3
Core Text	ARC-Challenge	47.4	46.2	50.4	50.4	**51.9**	53.9
	ARC-Easy	79.3	79.9	81.9	81.4	**82.3**	83.4
	HellaSwag	53.0	53.0	54.1	55.5	**55.8**	57.7
	LAMBDA	66.5	68.2	68.8	68.9	**69.0**	70.8
	PiQA	77.4	77.3	77.7	78.0	**78.7**	79.4
	SciQ	95.9	96.0	96.4	**96.7**	96.5	96.9
	WinoGrande	**69.8**	69.1	67.7	67.0	68.4	74.3
	Average	69.9	70.0	71.0	71.1	**71.8**	73.8
MMLU	MMLU	61.8	62.8	63.1	65.5	**66.1**	67.8

4.2. Evaluation with Input-Specific Masks

In this section, we evaluate our model using input-specific masks, which align with the training scheme. For all examples across the included datasets, we generate masks by feeding the testing question and few-shot examples (if applicable) into the sparsity predictor. The LLM is then pruned with the resulting mask and performs inference on the same input. The datasets provide a diverse range of inputs for the sparsity predictor, including combinations of few-shot examples and testing questions (*e.g.*, MATH and MMLU) and question-only formats (*e.g.*, AlpacaEval and IFEval).

Overall comparison We visualize the evaluation results in Table 1. We highlight the following observations. First, with an equivalent number of activated parameters, IFPRUNING significantly outperforms the dense LLM, demonstrating its ability to select the most relevant parameters for various inputs effectively. Specifically, IFPRUNING achieves a 5% and 4% higher win rate over the dense LLM on AlpacaEval and Arena Hard, respectively. Also, our method improves upon the dense baseline by 6% to 14% on coding tasks and by 3% to 5% on math benchmarks. We also observe substantial improvement on Tool Use and the MMLU benchmark. Finally, IFPRUNING shows strong performance on Core Text tasks, highlighting its broad applicability. The effectiveness of our approach is further underscored by its performance relative to the 9B-parameter "upper bound" model. Notably, on coding, math, and MMLU benchmarks, our method closely approaches upper-bound performance.

Second, IFPRUNING demonstrates superior performance compared to the structured pruning method. Please note that the structured pruning model, PRUNING+DISTILL, benefits from additional training signals due to knowledge distillation from a larger teacher model as the teacher. Nevertheless, IFPRUNING consistently outperforms this baseline. Our method achieves higher performance across a variety of benchmarks, including AlpacaEval, Arena Hard, math problems, coding tasks, tool use, MMLU, and most Core Text tasks.

Third, we observe a clear improvement in performance with IFPRUNING as the size of the LLM increases. As the source model size scales from 6B to 9B and then to 12B, there is a noticeable performance boost on most of the datasets.

Scaling behavior of dense models and IFPRUNING We illustrate the scaling behavior of dense LLMs (without pruning) and our IFPRUNING method in relation to model size

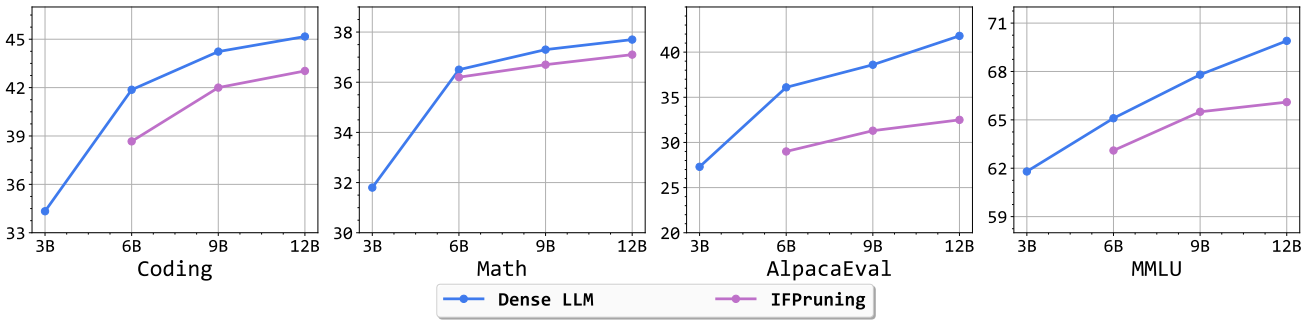

Figure 2. Scaling behavior of dense models and IFPRUNING. IFPRUNING activates 3B parameters for each input. The x-axis represents the total number of LLM parameters for dense models and IFPRUNING, while the y-axis indicates the performance scores.

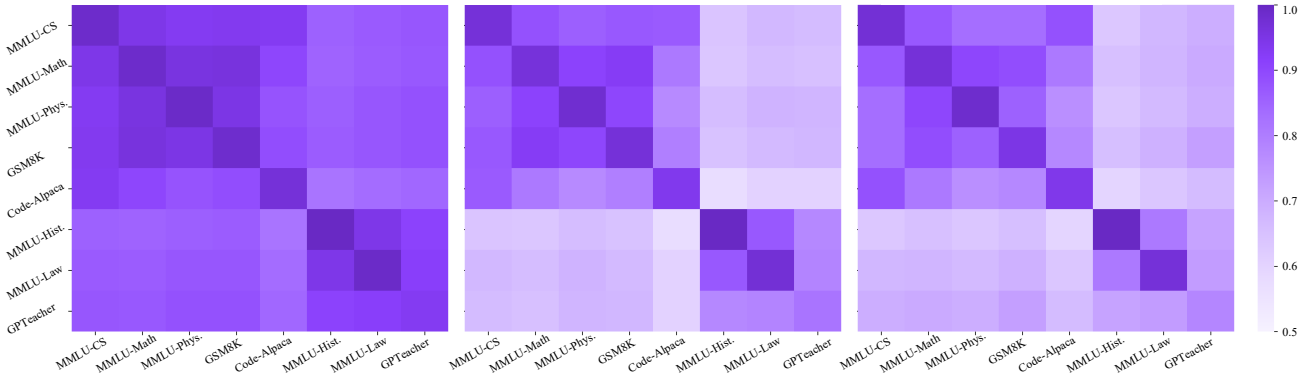

Figure 3. Sub-network overlap rates for the first layer (left), layer 16 (middle), and the last layer (right) of the LLM.

(total number of parameters) in Figure 2. In our approach, we consistently activate 3B parameters across LLMs with varying numbers of total parameters. In general, increasing the size of the LLM leads to performance improvements. This trend is especially clear on Math, coding, and MMLU tasks, where IFPRUNING achieves performance levels close to the upper bound. In contrast, we observe less performance gain on the AlpacaEval dataset, suggesting an opportunity for improvement and/or better understanding of the scaling behavior in future work.

Interpretability of parameter selection In this section, we examine the parameter pruning and selection patterns across different domains and visualize the similarities between them. We measure the similarity between two pruned models based on their *overlap rate*, defined as the proportion of parameters commonly activated by both models. More details of the implementation can be found in Appendix A.3. We perform this analysis of IFPRUNING using our 6B→3B model. We include the following datasets as our testing domains. For math, we use GSM8K and the college mathematics subset from MMLU (denoted as MMLU-Math). For computer science, we use the college computer science subset from MMLU (denoted as MMLU-CS) and Code-Alpaca (Chaudhary, 2023). We also include the college physics (MMLU-Phys.), high school European history (MMLU-Hist.), and international law (MMLU-Law) subsets from MMLU. For general instructions, we use the GPTeacher (Teknium et al., 2024) dataset that is not included in our training data. The overlap rates for the first, the last, and a middle layer (layer 16) of the pruned models are visualized in Figure 3.

We highlight the following findings. First, in the lower layers, especially the first layer, the LLM tends to activate very similar sub-networks for different inputs. As we move to higher layers, the parameter selection becomes more diverse. Second, as shown in the figure, IFPRUNING activates distinct sub-networks for different domains in higher layers. For instance, models pruned for MMLU-CS have substantial overlap with those for Code-Alpaca, significantly more than with other domains. Similarly, models for MMLU-Math, MMLU-Physics, and GSM8K share a high proportion of activated parameters, which diverge notably from the activation patterns for MMLU-History, MMLU-Law, and GPTTeacher. Third, the overlap rate along the diagonal of the heatmap is very high, reflecting the strong similarity between models for inputs within the same domain. Finally, as expected, GPTTeacher is an instruction-following dataset that covers a wide range of domains, therefore it does not exhibit significant domain-specific characteristics, resulting in a self-overlap rate that is lower than in other domains.

Table 2. Performance comparison between IFPRUNING with per-input masks and per-task masks. In the per-input setting, IFPRUNING prunes the LLMs and activates different sub-networks for each input. In the per-task setting, IFPRUNING selects a single sub-network for all inputs within the same dataset based on a human-written task instruction.

Category	Dataset	DENSE 3B	6B→3B		9B→3B		12B→3B	
			Per-Input	Per-Task	Per-Input	Per-Task	Per-Input	Per-Task
Coding	HumanEval	35.2	41.0	39.0	42.4	40.9	43.3	45.3
	MultiPL-E	39.0	37.6	38.5	41.8	42.8	43.0	44.2
	MBPP	28.8	37.4	39.0	41.8	38.2	42.8	40.4
	Average	34.3	38.7	38.8	42.0	40.6	43.0	43.3
Math	MATH	31.8	36.2	36.6	36.7	36.8	37.1	37.8
MMLU	MMLU-physics	49.6	50.8	50.7	55.5	54.2	55.7	54.4
	MMLU-math	43.7	42.8	41.8	44.0	45.8	44.2	45.5
	MMLU-history	73.7	75.9	75.2	78.0	74.7	78.5	77.8
	MMLU-health	59.7	61.9	62.0	64.4	63.4	65.1	63.7
	MMLU-business	75.3	73.5	75.3	76.6	75.3	75.5	76.1
	MMLU-economics	60.0	59.7	59.2	66.8	65.6	66.3	63.0
Translation	EN-DE	33.9	35.5	35.8	35.9	35.8	33.5	36.4
	EN-ES	26.9	27.4	27.2	27.0	27.2	26.8	27.5
	EN-FR	45.2	47.2	46.9	47.0	47.0	47.6	49.3
	EN-IT	28.8	30.2	30.1	30.3	30.5	31.0	30.2
	EN-PT	46.6	47.0	47.9	47.1	47.2	47.4	48.1
	EN-ZH	35.0	41.7	39.1	41.5	40.4	42.5	40.9
	Average	36.0	38.2	37.8	38.1	38.0	38.1	38.7

In summary, the activated parameters are interpretable and reveal clear patterns in different domains. It aligns with the high performance of our method, as the LLM dynamically activates the parameters most suitable for each input.

Ablation study We also conduct an ablation study on the impact of continued pre-training in Appendix A.4. Incorporating continued pre-training before SFT leads to consistent and substantial improvements across all benchmarks.

4.3. Evaluation with Task-Specific Masks

An additional noteworthy capability of IFPRUNING is task-specific pruning. While IFPRUNING can effectively prune a model on a per-input basis, the algorithm also demonstrates the ability to select one sub-network that can be used for different inputs within the same task or domain. We design the following experiment to demonstrate the performance of our method with task-specific pruning.

Specifically, we evaluate our models on math problems, coding tasks, MMLU, and machine translation tasks from Flores-101 (Goyal et al., 2022). For each dataset, we manually design a concise instruction that describes the task. For example, the instruction for the `HumanEval` dataset can be: "You are an expert Python programmer. Write the code which should pass the tests." The sparsity predictor takes this instruction as input and selects the corresponding sub-network. This sub-network is then applied consistently to process all testing examples from that dataset, and the model performance on the dataset is evaluated based on the predictions made by the sub-network. The task-specific instructions used for each dataset are provided in Table 6 in Appendix A.7. For the MMLU dataset, which consist of multiple subsets representing diverse domains, we group similar subsets into broader domains and assign a single instruction to each domain. For example, the subsets "astronomy," "college physics," "conceptual physics," and "high school physics" are grouped into the domain "MMLU-physics," and a single instruction is written for this domain. A detailed list of the subsets included in each MMLU domain is provided in Appendix A.6.

The performance is shown in Table 2. We compare the task-specific pruning capabilities of our algorithm with the dense LLM with 3B parameters and the standard IFPRUNING with per-input mask. Our key findings are as follows. First, IFPRUNING can generate high-quality task-specific masks without additional training. we observe substantial performance improvements of the task-specific IFPRUNING over the dense baseline across all datasets. When applying per-task masks, IFPRUNING still outperforms the dense LLM by 5%-12% on coding tasks, 5%-6% on MATH, and 1% - 5% on MMLU subsets. For translation tasks, IFPRUNING selects the appropriate sub-networks without additional training, achieving an improvement of 4.8 points in BLEU score on average. Note that these task-specific masks are directly predicted by the sparsity predictor, requiring no additional fine-tuning of either the sparsity predictor or the LLM for each task. This highlights the "zero-shot" pruning

capability of our method. Second, compared to the standard input-specific IFPRUNING, the task-specific IFPRUNING exhibit minimal performance degradation across math problems, coding tasks, and MMLU. The performance gaps are mostly under 1%, indicating the robustness of our method.

4.4. Inference Latency Evaluation

Given the strong performance of IFPRUNING, we further analyze and evaluate its efficiency improvement in this section. The inference pipeline of our method consists of three steps: ❶ Parameter selection: given an input, the sparsity predictor selects a subset of parameters to activate. ❷ Parameter loading: the selected parameters are loaded, and the corresponding sub-network is cached as a dense model. ❸ Generation: the pruned model will encode the input (prefill) and generate a response (decode), following the same procedure as standard dense or MoE models. Sub-network selection only involves a forward pass through the sparsity predictor, which is lightweight (300M parameters in our experiments), so the time cost is minimal. For parameter loading, we maintain a small base model and update its weights with the selected parameters. The generation step is identical to standard inference, allowing our method to achieve the same latency as a smaller dense model.

To illustrate the inference speedup, we run the following simulation. Specifically, we evaluate the inference latency of the models used in our main experiments: DENSE-9B, DENSE-3B, and IFPRUNING 9B→ 3B. Additionally, we test our method on an open-source LLM, LLaMA-3.1-8B-Instruct, where the FFN layers are pruned to reduce the model from 8B to 3B parameters, while keeping other components unchanged. Although the tests are done on GPUs, we used batch size 1 and 4 generations per query, reflecting on-device usage. The input length is set to 4,000 and the generation length is set to 100. We report the time-to-first-token (TTFT) and the decoding time, both measured in seconds. For dense models (8B and 3B), the TTFT consists of pre-filling only. For our method, we break down TTFT into its components: parameter selection, parameter loading, and pre-filling. The evaluation results are in Table 3.

We highlight the following conclusions. First, similar to standard structured pruning, our dynamic pruning scheme introduced can also significantly reduces inference latency for large-scale models. By pruning a 9B or 8B LLM to 3B, the TTFT is decreased by up to 57% and decoding time is decreased by up to 41%. In total, IFPRUNING can achieve up to 1.8x speedup compared to the original dense LLMs. Second, the overhead introduced by dynamic pruning and parameter caching is negligible. For each input, the parameter selection and loading only take less than 0.1s in total (1-2% of the total generation time). Finally, despite dynamic masking, the runtime of IFPRUNING is on par with static pruning (DENSE-3B baseline), while offering input-specific adaptivity and superior accuracy. The inference latency evaluation has further verified that our method is suitable for practical deployment on edge devices, as it accelerates inference while achieving superior model performance.

Table 3. Inference latency evaluation. The source models are reduced to 3B by pruning the FFN layers.

Model	Parameter Selection	Parameter Loading	Prefill	TTFT	Decode
Our model					
DENSE-9B	-	-	0.846	0.846	7.16
DENSE-3B	-	-	0.356	0.356	5.68
IFPRUNING	0.070	0.043	0.358	0.471	5.58
Llama-3.1-8b					
DENSE-8B	-	-	0.702	0.702	5.47
DENSE-3B	-	-	0.317	0.317	3.52
IFPRUNING	0.070	0.016	0.315	0.402	3.53

Finally, we compare the inference latency of our method and an MoE model with similar number of activated parameters in Appendix A.5. We show that for the on-device inference scenario with small inference batch size, the cost of MoE is multiple times higher than a dense model and our method.

5. Conclusion

In this paper, we extend the structured pruning for LLMs with a dynamic scheme, where the LLM is pruned into different sub-networks given the prompts. With a simple architecture and straightforward training process, our method can significantly improve the model performance compared to dense LLMs with the same number of activated parameters. In the future, we will test our method when pruning other components of LLMs such as attention heads and hidden dimensions.

This work also opens several promising directions for future exploration. First, we focus on dynamically pruning LLMs based on contextual information. While this approach is well suited for models on consumer-facing devices such as phones, additional challenges remain for server-side serving when the input is a batch of user requests containing different tasks. One possible solution is to cluster user requests so that requests within the same batch share similar activated sub-networks. Second, the current training method relies on end-to-end optimization, which may not fully utilize the training examples. The performance could be improved by adopting more advanced training strategies. For instance, using contrastive loss could encourage higher overlap rates among sub-networks for similar inputs, making the model more robust.

Acknowledgements

Shiyu Chang acknowledges support from National Science Foundation (NSF) Grant IIS-2338252, NSF Grant IIS-2207052, and NSF Grant IIS-2302730.

Impact Statement

In this paper, our primary goal is to develop an algorithm that can dynamically select the most suited parameters of an LLM given an input prompt. Our method is designed to improve both inference efficiency and the performance of LLMs. The training data has been carefully filtered to ensure quality and safety; for instance, all instances of personal data in the SFT data were removed to uphold privacy standards. All the data collection process strictly adheres to ethical guidelines for data use, ensuring that no private or sensitive information is included in the training or evaluation process.

Also, The sparsity-inducing mechanism proposed in this work does not introduce additional risks of bias or harm with the underlying large language model. Furthermore, our method enhances computational efficiency, potentially reducing the environmental impact of large-scale model inference. While acknowledging that any machine learning model has the potential for misuse, we focus on safe and task-specific applications, such as math, coding, and tool use. We encourage further research into mitigating biases and unintended consequences in language models and remain committed to the responsible and ethical advancement of AI technologies.

References

Ainslie, J., Lei, T., de Jong, M., Ontanon, S., Brahma, S., Zemlyanskiy, Y., Uthus, D. C., Guo, M., Lee-Thorp, J., Tay, Y., et al. Colt5: Faster long-range transformers with conditional computation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 5085–5100, 2023.

Akhauri, Y., AbouElhamayed, A. F., Dotzel, J., Zhang, Z., Rush, A. M., Huda, S., and Abdelfattah, M. S. Shadowllm: Predictor-based contextual sparsity for large language models. *arXiv preprint arXiv:2406.16635*, 2024.

Apple. The axlearn library for deep learning, 2023. URL https://github.com/apple/axlearn.

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Bisk, Y., Zellers, R., Bras, R. L., Gao, J., and Choi, Y. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., et al. Jax: composable transformations of python+ numpy programs, v0. 3.13, 2018.

Cassano, F., Gouwar, J., Nguyen, D., Nguyen, S., Phipps-Costin, L., Pinckney, D., Yee, M.-H., Zi, Y., Anderson, C. J., Feldman, M. Q., et al. Multipl-e: A scalable and extensible approach to benchmarking neural code generation. *arXiv preprint arXiv:2208.08227*, 2022.

Chaudhary, S. Code alpaca: An instruction-following llama model for code generation. https://github.com/sahil280114/codealpaca, 2023.

Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. Evaluating large language models trained on code. 2021.

Chen, T., Frankle, J., Chang, S., Liu, S., Zhang, Y., Wang, Z., and Carbin, M. The lottery ticket hypothesis for pre-trained bert networks. *Advances in neural information processing systems*, 33:15834–15846, 2020.

Chung, H. W., Hou, L., Longpre, S., Zoph, B., Tay, Y., Fedus, W., Li, Y., Wang, X., Dehghani, M., Brahma, S., et al. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53, 2024.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Dai, D., Deng, C., Zhao, C., Xu, R., Gao, H., Chen, D., Li, J., Zeng, W., Yu, X., Wu, Y., et al. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. *arXiv preprint arXiv:2401.06066*, 2024.

Dery, L., Kolawole, S., Kagy, J.-F., Smith, V., Neubig, G., and Talwalkar, A. Everybody prune now: Structured pruning of llms with only forward passes. *arXiv preprint arXiv:2402.05406*, 2024.

Dong, H., Chen, B., and Chi, Y. Prompt-prompted mixture of experts for efficient llm generation. *arXiv preprint arXiv:2404.01365*, 2024.

Du, N., Huang, Y., Dai, A. M., Tong, S., Lepikhin, D., Xu, Y., Krikun, M., Zhou, Y., Yu, A. W., Firat, O., et al. Glam: Efficient scaling of language models with mixture-of-experts. In *International Conference on Machine Learning*, pp. 5547–5569. PMLR, 2022.

Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Dubois, Y., Galambosi, B., Liang, P., and Hashimoto, T. B. Length-controlled alpacaeval: A simple way to debias automatic evaluators. *arXiv preprint arXiv:2404.04475*, 2024.

Elfwing, S., Uchibe, E., and Doya, K. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks*, 107:3–11, 2018.

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

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2018.

Gao, L., Tow, J., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., McDonell, K., Muennighoff, N., et al. A framework for few-shot language model evaluation. *Version v0. 0.1. Sept*, 10:8–9, 2021.

Goyal, N., Gao, C., Chaudhary, V., Chen, P.-J., Wenzek, G., Ju, D., Krishnan, S., Ranzato, M., Guzmán, F., and Fan, A. The flores-101 evaluation benchmark for low-resource and multilingual machine translation. *Transactions of the Association for Computational Linguistics*, 10:522–538, 2022.

Gunter, T., Wang, Z., Wang, C., Pang, R., Narayanan, A., Zhang, A., Zhang, B., Chen, C., Chiu, C.-C., Qiu, D., et al. Apple intelligence foundation language models. *arXiv preprint arXiv:2407.21075*, 2024.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

Hendrycks, D. and Gimpel, K. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021a.

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the math dataset. *NeurIPS*, 2021b.

Hinton, G. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Kim, B.-K., Kim, G., Kim, T.-H., Castells, T., Choi, S., Shin, J., and Song, H.-K. Shortened llama: A simple depth pruning for large language models. *arXiv preprint arXiv:2402.02834*, 11, 2024.

Kurtić, E., Frantar, E., and Alistarh, D. Ziplm: Inference-aware structured pruning of language models. *Advances in Neural Information Processing Systems*, 36, 2024.

Lee, J.-Y., Lee, D., Zhang, G., Tiwari, M., and Mirhoseini, A. Cats: Contextually-aware thresholding for sparsity in large language models. *arXiv preprint arXiv:2404.08763*, 2024.

Lei, T., Bai, J., Brahma, S., Ainslie, J., Lee, K., Zhou, Y., Du, N., Zhao, V., Wu, Y., Li, B., et al. Conditional adapters: Parameter-efficient transfer learning with fast inference. *Advances in Neural Information Processing Systems*, 36: 8152–8172, 2023.

Lepikhin, D., Lee, H., Xu, Y., Chen, D., Firat, O., Huang, Y., Krikun, M., Shazeer, N., and Chen, Z. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

Li, T., Chiang, W.-L., Frick, E., Dunlap, L., Wu, T., Zhu, B., Gonzalez, J. E., and Stoica, I. From crowdsourced data to high-quality benchmarks: Arena-hard and benchbuilder pipeline. *arXiv preprint arXiv:2406.11939*, 2024.

Li, Z., Wallace, E., Shen, S., Lin, K., Keutzer, K., Klein, D., and Gonzalez, J. Train big, then compress: Rethinking model size for efficient training and inference of transformers. In *International Conference on machine learning*, pp. 5958–5968. PMLR, 2020.

Liu, A., Feng, B., Wang, B., Wang, B., Liu, B., Zhao, C., Dengr, C., Ruan, C., Dai, D., Guo, D., et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts

language model. *arXiv preprint arXiv:2405.04434*, 2024a.

Liu, L., Kim, Y. J., Wang, S., Liang, C., Shen, Y., Cheng, H., Liu, X., Tanaka, M., Wu, X., Hu, W., et al. Grin: Gradient-informed moe. *arXiv preprint arXiv:2409.12136*, 2024b.

Liu, Z., Wang, J., Dao, T., Zhou, T., Yuan, B., Song, Z., Shrivastava, A., Zhang, C., Tian, Y., Re, C., et al. Deja vu: Contextual sparsity for efficient llms at inference time. In *International Conference on Machine Learning*, pp. 22137–22176. PMLR, 2023.

Louizos, C., Welling, M., and Kingma, D. P. Learning sparse neural networks through l_0 regularization. In *International Conference on Learning Representations*, 2018.

Ma, X., Fang, G., and Wang, X. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36:21702–21720, 2023.

Men, X., Xu, M., Zhang, Q., Wang, B., Lin, H., Lu, Y., Han, X., and Chen, W. Shortgpt: Layers in large language models are more redundant than you expect. *arXiv preprint arXiv:2403.03853*, 2024.

Meta AI. Llama 3 and vision: Bringing next-gen ai to edge and mobile devices, 2024. URL https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/. Accessed: 2024-11-10.

Mirzadeh, I., Alizadeh, K., Mehta, S., Del Mundo, C. C., Tuzel, O., Samei, G., Rastegari, M., and Farajtabar, M. Relu strikes back: Exploiting activation sparsity in large language models. *arXiv preprint arXiv:2310.04564*, 2023.

Muralidharan, S., Sreenivas, S. T., Joshi, R. B., Chochowski, M., Patwary, M., Shoeybi, M., Catanzaro, B., Kautz, J., and Molchanov, P. Compact language models via pruning and knowledge distillation. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Narang, S., Elsen, E., Diamos, G., and Sengupta, S. Exploring sparsity in recurrent neural networks. *arXiv preprint arXiv:1704.05119*, 2017.

Paperno, D., Kruszewski, G., Lazaridou, A., Pham, N.-Q., Bernardi, R., Pezzelle, S., Baroni, M., Boleda, G., and Fernández, R. The lambada dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1525–1534, 2016.

Ramachandran, P., Zoph, B., and Le, Q. V. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

Sakaguchi, K., Bras, R. L., Bhagavatula, C., and Choi, Y. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Shazeer, N. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Song, C., Han, X., Zhang, Z., Hu, S., Shi, X., Li, K., Chen, C., Liu, Z., Li, G., Yang, T., et al. Prosparse: Introducing and enhancing intrinsic activation sparsity within large language models. *arXiv preprint arXiv:2402.13516*, 2024.

Sreenivas, S. T., Muralidharan, S., Joshi, R., Chochowski, M., Patwary, M., Shoeybi, M., Catanzaro, B., Kautz, J., and Molchanov, P. Llm pruning and distillation in practice: The minitron approach. *arXiv preprint arXiv:2408.11796*, 2024.

Sun, H., Lei, T., Zhang, B., Li, Y., Huang, H., Pang, R., Dai, B., and Du, N. Ec-dit: Scaling diffusion transformers with adaptive expert-choice routing. *arXiv preprint arXiv:2410.02098*, 2024.

Team, Q. Qwen2.5: A party of foundation models, September 2024. URL https://qwenlm.github.io/blog/qwen2.5/.

Teknium, Machina, L. L., and Ashimine, I. E. A collection of modular datasets generated by gpt-4, 2024.

Voita, E., Talbot, D., Moiseev, F., Sennrich, R., and Titov, I. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 5797–5808, 2019.

Wang, Z., Wohlwend, J., and Lei, T. Structured pruning of large language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 6151–6162, 2020.

Welbl, J., Liu, N. F., and Gardner, M. Crowdsourcing multiple choice science questions. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pp. 94–106, 2017.

Wen, W., Wu, C., Wang, Y., Chen, Y., and Li, H. Learning structured sparsity in deep neural networks. *Advances in neural information processing systems*, 29, 2016.

Xia, M., Gao, T., Zeng, Z., and Chen, D. Sheared llama: Accelerating language model pre-training via structured

pruning. In *The Twelfth International Conference on Learning Representations*, 2024.

Yang, A., Yang, B., Hui, B., Zheng, B., Yu, B., Zhou, C., Li, C., Li, C., Liu, D., Huang, F., et al. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024a.

Yang, Y., Cao, Z., and Zhao, H. Laco: Large language model pruning via layer collapse. *arXiv preprint arXiv:2402.11187*, 2024b.

Yin, G., Bai, H., Ma, S., Nan, F., Sun, Y., Xu, Z., Ma, S., Lu, J., Kong, X., Zhang, A., et al. Mmau: A holistic benchmark of agent capabilities across diverse domains. *arXiv preprint arXiv:2407.18961*, 2024.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, 2019.

Zhou, J., Lu, T., Mishra, S., Brahma, S., Basu, S., Luan, Y., Zhou, D., and Hou, L. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

Zhou, Y., Lei, T., Liu, H., Du, N., Huang, Y., Zhao, V., Dai, A. M., Le, Q. V., Laudon, J., et al. Mixture-of-experts with expert choice routing. *Advances in Neural Information Processing Systems*, 35:7103–7114, 2022.

Zhou, Y., Chen, Z., Xu, Z., Lin, V., and Chen, B. Sirius: Contextual sparsity with correction for efficient llms. *arXiv preprint arXiv:2409.03856*, 2024.

Zhu, M. and Gupta, S. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

Zoph, B., Bello, I., Kumar, S., Du, N., Huang, Y., Dean, J., Shazeer, N., and Fedus, W. St-moe: Designing stable and transferable sparse expert models. *arXiv preprint arXiv:2202.08906*, 2022.

A. Appendix

A.1. Implementation Details

LLM architecture The LLMs used in this paper follow standard LLM design (Meta AI, 2024; Team, 2024), such as grouped-query attention and RMSNorm, with no custom components. We list the detailed model architecture of the pre-trained models used in our experiments in Table 4. All models use the same model dimension, attention dimensions and the number of Transformer layers. The only difference is the feed-forward dimension. Accordingly, our method learns to select 6656 FFN dimensions from the 6B and 9B model.

Table 4. Detailed Architecture of the pre-trained models used in our experiments

	3B	6B	9B
Model Dimension	2048	2048	2048
FFN Dimension	6656	16384	24576
Head Dimension	128	128	128
Num query heads	16	16	16
Num key/value heads	2	2	2
Num layers	56	56	56

Sparsity predictor The predictor is built upon a lightweight LLM with 302M parameters, built on a pre-trained LM backbone. It consists of:

- A small pre-trained language model with 302M parameters as the feature extractor. Given an input, the last hidden state of the final input token will be used as the features of the input.
- A Two-layer MLP as the prediction head. Given the extracted feature vector with dimension d, the first layer transform the input from d to 128. The second layer transform the output of the first layer into $L * d_{\text{ffn}}$, where L is the number of layers of the source LLM being pruned and d_{ffn} is the intermediate dimension of the source LLM. Then we reshape the output vector into a l-by-d_{ffn} matrix. Each row of the matrix will be processed by the SOFTTOPK (Lei et al., 2023) operator independently to get the FFN mask for the corresponding layer.

A.2. Training of the Structured Pruning Baseline

Our pruning baseline, PRUNING+DISTILL, first prunes the given LLM into a dense 3B model then perform continuous pre-training through knowledge distillation. For knowledge distillation. we apply KL divergence between the output distributions of the student and the teacher model for each output token. The teacher distribution only keeps the highest-scoring tokens, similar to Minitron (Muralidharan et al., 2024). We minimize a combined loss of the standard next-token prediction and KL divergence loss.

A.3. Interpretability of Parameter Selection

We provide more details of the visualization of the parameter selection pattern in Figure 3. First, we sample 128 inputs per dataset (MMLU, GSM8K, CodeAlpaca-20K, GPTTeacher). Inputs to the sparsity predictor are formatted with in-context examples (MMLU: 5-shot, GSM8K: 8-shot) or raw prompts (CodeAlpaca, GPTTeacher). After that, each input is passed through the sparsity predictor, which selects a fixed number of FFN units (a binary mask) at each layer, producing 128 sub-networks per dataset. To calculate the overlap, We compare every pair of sub-networks within the 128 examples. For each pair, we compute the fraction of selected FFN units they share. The final overlap rate is the average of these pairwise overlaps.

A.4. Ablation Study on the Continued Pre-training

In this section, we conduct a ablation study on the impact of the continued pre-training. We apply our method and activate 3B parameters of a DENSE-6B LLM. We train two variants of models: one with the SFT phase ony and one with both continued pre-training and SFT phases. The experiment results are shown in Table 5. We observe consistent and notable

Table 5. Ablation study on the impact of the continued pre-training. We apply IFPRUNING to a dense LLM with 6B parameters and activate 3B parameters for each input.

	HumanEval	MBPP	MultiPL-E	GSM8K	MATH	MMLU
SFT Only	25.3	24.4	15.3	50.9	13.5	55.2
Continued pre-training + SFT	31.9	35.3	22.4	61.3	20.1	59.3

gains across all benchmarks, demonstrating the effectiveness of the continued pre-training phase.

A.5. Inference Latency Comparison with MoE

In this section, we compare the latency of our method with the open-source model, Qwen1.5-MoE-A2.7B. It has 14.3B parameters in total and activates 2.7B parameters per token. For our method, we prune LLaMA-3-8B to 3B parameters. The experiment configurations are exactly the same as the experiment in Table 3. Specifically, we evaluate the latency on a single NVIDIA RTX A6000 GPU and report time-to-first-token (TTFT) and decoding time with input length = 4k, generation length = 100, and sample 4 responses for each query. The experiment results are shown in Table 4.

Figure 4. Inference latency evaluation. The source models are reduced to 3B by pruning the FFN layers.

Model	Param. selection	Param. loading	Prefill	TTFT	Decode
DENSE-9B	-	-	0.846	0.846	7.16
MOE-A2.7B	-	-	0.621	0.621	28.43
DENSE-3B	-	-	0.356	0.356	5.68
IFPRUNING	0.070	0.043	0.358	0.471	5.58

We can see the dense baseline and our method have significantly better latency and throughput than MoE. The main reason is that MoE models are not efficient for on-device inference when generating responses given a single query, where the decoding is bottlenecked by weight loading. Since MoE requires reading many expert weights (e.g., 2 for each token), the cost of MoE is multiple times higher than a dense model and our method. This further demonstrates that our method is more suited for on-device scenarios.

A.6. Subsets included in MMLU domains

We list the subsets in each MMLU domain for the experiment in Table 2.

MMLU-physics: astronomy, college physics, conceptual physics, and high school physics.

MMLU-math: abstract algebra, college mathematics, elementary mathematic, high school mathematics, and high school statistics.

MMLU-history: high school european history, high school us history, high school world history, and prehistory.

MMLU-health: anatomy, clinical knowledge, college medicine, human aging, medical genetics , nutrition, professional medicine, and virology.

MMLU-business: business ethics, management, and marketing.

MMLU-economics: econometrics, high school macroeconomics, and high school microeconomics.

A.7. Detailed Per-task Prompts

In this section, we list the full prompts for the task-specific pruning in Section 4.3 in Table 6.

A.8. License of Datasets and Use of Generative AI

We include a diverse set of datasets for evaluation, and their licenses are detailed below. The IFEval dataset is released under the Apache License 2.0. AlpacaEval 2.0 and Arena Hard are also under the Apache-2.0 License. HumanEval, GSM8K, MATH, HellaSwag, WinoGrande, and LM-Evaluation-Harness are under the MIT License. The mbpp dataset is distributed under the Creative Commons Attribution 4.0 license, while MultiPL-E is under the BSD 3-Clause License with Machine Learning Restriction. The ARC dataset is provided under the Creative Commons Attribution Share Alike 4.0 license, and SciQ is under the Creative Commons Attribution Non-Commercial 3.0 license. PiQA is licensed under the Academic Free

Table 6. Task prompt for task-specific pruning experiments.

MATH	You are a Math expert. You will be given a math problem in domains such as algebra, probability, geometry and number theory. Reason and give a final answer to the problem. Your response should end with "The answer is [answer]" where [answer] is the response to the problem.
MMLU-Physics	You are an expert in physics. You will be given multiple choice questions in subjects such as astronomy, conceptual physics, and college physics. Select the correct answer to each question.
MMLU-History	You are an expert in history. You will be given multiple choice questions in subjects such as european history, us history and prehistory. Select the correct answer to each question.
MMLU-Economics	You are an expert in economics. You will be given multiple choice questions in subjects such as econometrics, macroeconomics and microeconomics. Select the correct answer to each question.
Translation (EN-DE)	You are a skilled translator who specializes in English to German translations. Your task is to accurately translate the provided English text into German while preserving the meaning and context.
Translation (others)	Same as above. Replace the language names with the language pair being tested.
Multiple-E Swift	You are an expert Swift programmer. You will be given a Swift function definition in documentation comments after ///. Write the code to complete the function. Here is an example input: ``` swift /// Write a swift function to count inversions in an array. func get_Inv_Count(arr: [Int]) → Int
HumanEval-Python	You are an expert Python programmer. You will be given a Python function definition and some test examples in triple quotes """. Write the code which should pass the tests. Here is an example input: ``` python def greatest_common_divisor(a: int, b: int) -¿ int: """Return a greatest common divisor of two integers a and b >>> greatest_common_divisor(3, 5) 1 >>> greatest_common_divisor(25, 15) 5 """
Mbpp	You are an expert Python programmer. You will be given a Python function definition and some test examples in triple quotes """. Write the code which should pass the tests. Here is an example input: ``` python """ Write a function to find the similar elements from the given two tuple lists. assert similar_elements((3, 4, 5, 6),(5, 7, 4, 10)) == (4, 5) """

License v. 3.0. Additionally, MMLU is under the MIT License, and MMAU is distributed under the Creative Commons Attribution 4.0 license.

The usage of all datasets and packages in this work aligns with their intended purposes, specifically the evaluation of LLMs.

We utilized GPT-4 to assist in checking and refining grammar and clarity across all sections. The core ideas, analyses, and textual composition remain entirely the work of the authors.