MAPLE: MoE Adaptive Plug-and-play Layer-wise Expert allocation

Anonymous authors
Paper under double-blind review

Abstract

Sparsely-activated Mixture-of-Experts (MoE) Transformers universally fix the same number of routed experts across all layers, a convention that ignores the well-documented heterogeneity in layer-wise redundancy. We demonstrate that this uniformity is suboptimal and propose **MAPLE**, a plug-and-play framework that reallocates the routed-expert budget heterogeneously across layers of any pretrained MoE LLM, pruning computation rather than parameters: no expert weight is modified, removed, or retrained. Our core contribution is a closed-form sensitivity-guided allocation: we probe each layer's response to variation in expert count, quantify sensitivity, and derive a budget assignment that directs capacity towards sensitive layers and absorbs reductions in redundant layers. This closed-form solution is further refined by a sensitivity-constrained genetic search that uses layer-wise sensitivity as a prior guide for exploration, yielding faster convergence and superior allocation quality. The allocation also creates an execution regime that uniform routing never encounters, which we exploit with a dedicated top-1 routing GPU kernel integrated into SGLang and vLLM. On four MoE models spanning different scales and architectures, **MAPLE** outperforms uniform and pruning-based baselines under a 75% total expert-activation budget. Notably, on DeepSeek-MoE-16B, **MAPLE** uses only 75% of the experts yet surpasses the original 100% expert-uniform baseline on ARC-E, ARC-C, and BoolQ, improving accuracy from 65.09 to 71.40, 48.49 to 51.50, and 80.03 to 82.38, respectively. These accuracy gains translate into measured deployment efficiency: implementing **MAPLE** in SGLang reduces single-GPU end-to-end serving latency by 32.2% and improves throughput by 47.4%. These results show that well-designed heterogeneous allocation can be more effective than simply activating more experts, making it a practical axis for improving MoE efficiency.

1 Introduction

Plug-and-play adaptability, formalised in signal processing (Venkatakrishnan et al., 2013), refers to the ability to improve an existing system by inserting a lightweight module without redesigning it from scratch. Deep learning has repeatedly benefited from this principle: Batch Normalization (Ioffe & Szegedy, 2015), residual connections (He et al., 2016), and Dropout (Srivastava et al., 2014) became widely adopted because they brought substantial practical benefits with minimal architectural modification. As neural networks evolved, this philosophy became increasingly targeted. Modules such as Squeeze-and-Excitation blocks (Hu et al., 2018) and CBAM (Woo et al., 2018) selectively recalibrate informative feature responses rather than intervening uniformly across the network. These successes suggest a common principle: lightweight plug-and-play interventions can provide substantial practical benefits while requiring only small changes to the original model.

As Transformer-based language models continued to scale (Devlin et al., 2019; Raffel et al., 2020; Brown et al., 2020; Chowdhery et al., 2023; Touvron et al., 2023a;b), full retraining became increasingly expensive, making plug-and-play adaptation more attractive. Adapter layers (Houlsby et al., 2019), LoRA (Hu et al., 2022), Prefix tuning (Li & Liang, 2021), prompt tuning (Lester et al., 2021), and BitFit (Zaken et al., 2022) offered efficient alternatives by updating only a small subset of parameters. Yet despite evidence that different model components contribute unequally (Bahdanau et al., 2015; Vaswani et al., 2017; Michel et al., 2019; Voita et al., 2019), these methods still largely treat adaptation uniformly rather than tailoring it to the parts of the model that matter most.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

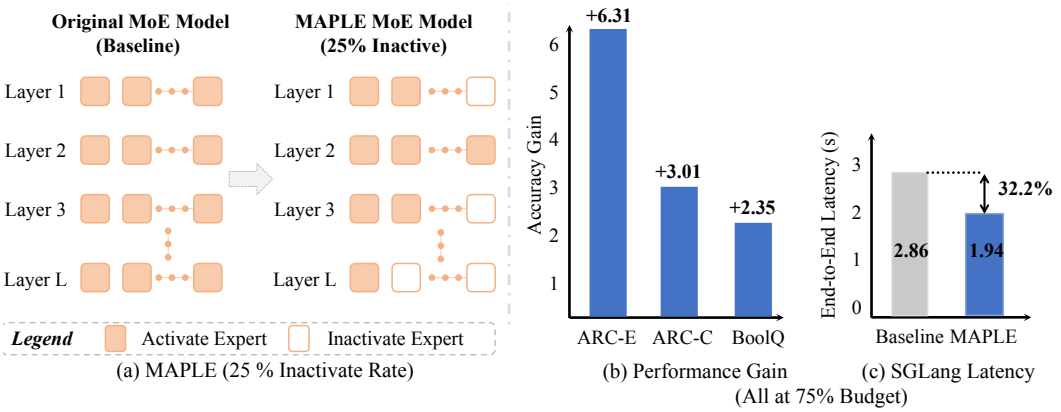

Figure 1: Overview and key results of **MAPLE**. **(a)** MAPLE replaces uniform expert allocation with sensitivity-guided heterogeneous allocation, with 25% of experts inactive. **(b)** At a 75% expert budget, MAPLE improves accuracy by 6.31, 3.01, and 2.35 points on three datasets, respectively. **(c)** MAPLE reduces end-to-end latency by 32.2% on SGLang (Details are provided in Section 4).

This untapped dimension becomes especially important in Mixture-of-Experts (MoE) LLMs (Jacobs et al., 1991; Shazeer et al., 2017), where model capacity is exposed explicitly through sparse expert activation. By replacing dense feed-forward layers with sparsely activated expert pools, MoE models decouple parameter capacity from per-token compute (Fedus et al., 2022; Lepikhin et al., 2021; Du et al., 2022). With continued improvements in stability and routing (Zoph et al., 2022; Zhou et al., 2022), MoE architectures have also been adopted in a number of recent high-performing models, including OLMoE (Muennighoff et al., 2024), DeepSeek-MoE (Dai et al., 2024), and Moonlight (Liu et al., 2025). Yet despite this flexibility, current pretrained MoE LLMs still typically use a fixed per-layer number of active experts, even though recent work suggests that expert importance and redundancy vary substantially across layers (Guo et al., 2023; Lu et al., 2024; Zhang et al., 2026).

Existing evidence also points to substantial layer-wise heterogeneity. In parameter-efficient fine-tuning, higher layers have been shown to benefit from more LoRA experts (Gao et al., 2024), suggesting that adaptation demands are not uniform across depth. More importantly, recent studies on expert pruning and layer compression indicate that redundancy in MoE models varies substantially across layers (Lu et al., 2024; Liu et al., 2024; Dong et al., 2025; Cao et al., 2024). Taken together, these findings suggest that routed expert capacity should not be allocated uniformly, but instead determined in a layer-aware manner.

However, existing approaches make such layer-aware allocation costly. Pruning-based methods (Han et al., 2015; Frantar & Alistarh, 2023; Sun et al., 2024; Xie et al., 2024; Zhang et al., 2026; Liu et al., 2024) either reconstruct weights from calibration data or permanently remove experts, modifying the pretrained checkpoint and precluding plug-and-play deployment. The deployment side is equally constrained: fused MoE kernels and serving frameworks (Gale et al., 2022; Zhao et al., 2025; Zheng et al., 2024; Kwon et al., 2023) are all optimised around a uniform, configuration-invariant top-k, so a heterogeneous allocation cannot be fully exploited without modifying the execution path on hardware platforms.

In this work, we propose **MAPLE** (**M**oE **A**daptive **P**lug-and-play **L**ayer-wise **E**xpert allocation), which determines heterogeneous per-layer expert allocations in pretrained MoE Transformers without modifying weights, architectures, or requiring retraining. **MAPLE** formulates expert reallocation as a constrained optimisation problem: a lightweight layer-wise sensitivity scan, a closed-form allocation under a global expert budget, and a sensitivity-guided light genetic refinement. Throughout this process, no expert weight is updated or removed. The allocation also creates an execution regime that uniform routing never encounters in MoE models, since layers driven to a single expert reduce expert selection to taking the highest-scoring expert, which we exploit with a dedicated top-1 GPU routing kernel integrated into SGLang and vLLM. Figure 1 gives an overview: **MAPLE** assigns a different number of active experts to each layer (a), and at a 75% expert budget it exceeds the full-budget baseline by 6.31 points on ARC-E (b), while cutting end-to-end latency by 32.2% (c).

Our main contributions are as follows:

- We identify heterogeneous per-layer expert allocation as an underexplored yet effective axis for improving pretrained MoE LLMs. Unlike existing work (Frantar & Alistarh, 2023; Sun et al., 2024; Zhang et al., 2024; Liu et al., 2024; Zhang et al., 2026), we propose **MAPLE**, a fully plug-and-play framework that requires no weight updates, retraining, or model architectural change.

- We introduce a sensitivity-guided optimisation strategy for expert reallocation, in which a lightweight layer-wise sensitivity scan enables a closed-form allocation under a global expert budget, with genetic search for subsequent refinement.

- We develop a MAPLE-aware framework integrated into SGLang (Zheng et al., 2024) and vLLM (Kwon et al., 2023), which specialises GPU kernels according to the layer-wise routing degree and eliminates redundant dispatch and aggregation for single-expert layers, yielding latency and throughput gains beyond nominal FLOP reductions.

2 Related Work

Layer-wise Sensitivity Analysis. Understanding layer-wise response to capacity reduction is fundamental to model compression. Prior work has shown that network redundancy is substantial but highly non-uniform across layers (Han et al., 2015; Frankle & Carbin, 2019; Guo et al., 2023). Sensitivity-based analyses therefore motivate non-uniform compression, where capacity is allocated according to functional importance rather than reduced uniformly (Engelbrecht, 2001; Ma et al., 2023). Evidence from structured pruning and attention-head analysis further supports this view, showing that layers and sub-components contribute unequally to performance (Michel et al., 2019; Voita et al., 2019).

Mixture-of-Experts Models and Expert Compression. MoE models improve parameter efficiency through sparse expert activation (Shazeer et al., 2017; Fedus et al., 2022), as shown by recent architectures such as Mixtral (Jiang et al., 2024) and DeepSeekMoE (Dai et al., 2024). To reduce deployment cost, early work adapted dense-model pruning methods to MoE experts, including SparseGPT (Frantar & Alistarh, 2023), Wanda (Sun et al., 2024), and router-aware pruning in MoE-Pruner (Xie et al., 2024). Because unstructured sparsity often brings limited practical acceleration, later studies shifted to structured expert-level compression. These include activation-frequency-based pruning in Not All Experts are Equal (Lu et al., 2024), evolutionary expert search in EEP (Liu et al., 2024), and expert merging in MC-SMoE (Li et al., 2024). More recent methods further introduce lightweight replacements or calibration-guided compression, such as MoNE (Zhang et al., 2026), ConDense-MoE (Cao et al., 2024) and domain-specific expert pruning approaches (Dong et al., 2025). However, these methods generally modify model weights or structure, rely on calibration data, and often overlook heterogeneous layer-wise redundancy. LExI (Chitty-Venkata et al., 2025) is more closely related to MAPLE: it also preserves the pretrained model and assigns heterogeneous active-expert counts across layers, but derives the allocation in a data-free manner from model-weight statistics. Our method instead reallocates the active expert budget across layers based on sensitivity to improve model performance, without altering model weights.

MoE Deployment. Efficient MoE serving relies on specialised GPU kernels for token routing, expert computation, and output combination. MegaBlocks (Gale et al., 2022) develops block-sparse matrix-multiplication kernels to execute dynamically routed experts without the padding overhead of conventional grouped computation. ScatterMoE (Tan et al., 2024) further introduces the ParallelLinear primitive, whose scatter-to-scatter kernels fuse token reordering with expert linear operations to reduce intermediate copies and improve expert-computation efficiency. Complementary to compute-side optimisation, DeepEP (Zhao et al., 2025) provides specialised All-to-All GPU kernels for MoE dispatch and combine, targeting the communication overhead of routing tokens to experts and returning their outputs. Serving frameworks such as SGLang (Zheng et al., 2024) and vLLM (Kwon et al., 2023) integrate such primitives into fused MoE execution paths comprising top-k routing, expert computation, and weighted output combination. These optimisations accelerate the generic multi-expert pipeline under a fixed routing configuration. **MAPLE** instead varies that configuration across layers, and exploits the resulting structure with a specialised argmax-and-forward path for single-expert layers.

3 METHOD

We present **MAPLE**, a plug-and-play framework for sensitivity-guided heterogeneous expert allocation in budget-constrained MoE Transformers. Rather than pruning weights, it prunes activations, selecting which router-ranked experts to evaluate at inference while leaving all expert weights intact. The allocation algorithm (Section 3.1) estimates layer-wise sensitivity and each layer's optimum, derives a closed-form allocation under a global budget, and refines it with a lightweight genetic search. Since this drives the least sensitive layers to a single expert, it creates an execution regime that uniform routing never encounters, which the deployment design (Section 3.2) exploits with a top-1 routing kernel we implement in CUDA, turning the allocation into measured speedups rather than theoretical FLOP savings.

3.1 ALGORITHM DESIGN: SENSITIVITY-GUIDED HETEROGENEOUS ALLOCATION

3.1.1 LAYER-WISE SENSITIVITY MODELLING

MoE models typically use a uniform expert count across layers, although layers play distinct roles and exhibit varying redundancy (Guo et al., 2023; Lu et al., 2024; Zhang et al., 2026). Because routing is performed independently at each layer (Qiu et al., 2025), we probe layers individually. Specifically, while fixing all other layers at the baseline expert count k_0, we vary the expert count of layer l over $\mathcal{E} = \{1, \dots, k_0\}$, where $e \in \mathcal{E}$ denotes the number of routed experts assigned to layer l, and record the accuracy $A_l(e)$. The single-layer optimum is $k_l^* = \min\{e \mid A_l(e) = \max_{e' \in \mathcal{E}} A_l(e')\}$.

The optimum k_l^* alone does not indicate how strongly layer l responds to expert-count variation. Prior work in pruning, quantisation, and adaptive fine-tuning similarly reports layer-dependent sensitivity (LeCun et al., 1990; Molchanov et al., 2019; Dong et al., 2019; 2020; Wu et al., 2020; Zhang et al., 2023). We therefore consider three lightweight sensitivity measures: $s_l^{\text{acc}} = \max_{e \in \mathcal{E}} A_l(e) - \min_{e \in \mathcal{E}} A_l(e)$, $s_l^{w} = \frac{1}{|W_l|} \sum_{W \in W_l} \mathrm{Var}(W)$, and $s_l^{\text{out}} = \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} \mathrm{MSE}\big(H_l(e), H_l(k_0)\big)$. Here, W_l is the set of parameter tensors in layer l, and $H_l(e)$ is its output when using e experts. These metrics capture performance-, parameter-, and representation-level sensitivity. We use accuracy range by default and evaluate the alternatives in the ablation study; s_l denotes the selected metric.

3.1.2 SENSITIVITY-GUIDED CLOSED-FORM EXPERT ALLOCATION

Given the preferred expert counts $\{k_l^*\}_{l=1}^L$, sensitivities $\{s_l\}_{l=1}^L$, and global budget K, we seek an allocation $\mathbf{k} = [k_1, \dots, k_L]$ that stays close to each layer's optimum, with more sensitive layers deviating less. This gives the weighted least-deviation program

$$\min_{\mathbf{k}} \quad \mathcal{J}(\mathbf{k}) = \sum_{l=1}^{L} w_l \big(k_l - k_l^*\big)^2 \qquad \text{s.t.} \qquad \sum_{l=1}^{L} k_l = K, \tag{1}$$

where $w_l = \max(s_l, \epsilon)$, with $\epsilon = 10^{-8}$, so layers with higher sensitivity receive larger weights in the allocation objective and are adjusted less.

Introducing a Lagrange multiplier λ for the budget constraint in equation 1 gives

$$\mathcal{L}(\mathbf{k}, \lambda) = \mathcal{J}(\mathbf{k}) + \lambda \left(\sum_{l=1}^{L} k_l - K \right). \tag{2}$$

Setting $\partial \mathcal{L} / \partial k_l = 0$ in equation 2 yields the stationarity condition

$$k_l^{\text{cont}} = k_l^* - \frac{\lambda}{2 w_l}, \tag{3}$$

and substituting equation 3 back into the budget constraint determines the multiplier

$$\lambda = \frac{2\Big(\sum_{l=1}^{L} k_l^* - K\Big)}{\sum_{l=1}^{L} 1/w_l}. \tag{4}$$

Together, equation 3 and equation 4 give a closed-form continuous allocation, which we next clip to the feasible range and round to integers. Let $\Delta = K - \sum_l k_l^*$ and $\Gamma = \sum_l k_l^* - K$. Any surplus or deficit relative to K is assigned preferentially to low-sensitivity layers:

$$k_l^{\text{cont}} = \begin{cases} k_l^* + \Delta \cdot \dfrac{1/w_l}{\sum_{j=1}^{L} 1/w_j}, & \text{if } \sum_l k_l^* < K, \\ k_l^* - \Gamma \cdot \dfrac{1/w_l}{\sum_{j=1}^{L} 1/w_j}, & \text{if } \sum_l k_l^* > K. \end{cases} \tag{5}$$

We clip and round k_l^{cont} while enforcing $k_{\min} \le k_l \le k_{\max}$ and $\sum_l k_l = K$.

3.1.3 Sensitivity-constrained Genetic Refinement

The closed-form solution $\mathbf{k}^{\text{sens}}$ provides a principled initialization but may retain rounding artefacts and residual cross-layer interactions. We therefore apply a lightweight genetic algorithm with accuracy $A(\mathbf{k})$ as fitness. Candidate allocations are generated by sensitivity-controlled Gaussian mutation,

$$\tilde{k}_l = k_l^{\text{sens}} + \epsilon_l^{\text{g}}, \qquad \epsilon_l^{\text{g}} \sim \mathcal{N}(0, \sigma_l^2), \qquad \sigma_l^2 \propto \frac{1}{s_l + \varepsilon_{\text{g}}}, \tag{6}$$

so that high-sensitivity layers receive smaller perturbations. At each generation, we retain the top two candidates as elites and generate the remaining population using the same perturbation, without crossover. The best candidate gives the final allocation $\mathbf{k}^{\text{best}}$.

Algorithm. The pseudocode and algorithmic pipeline can be found in Appendix A.1.

3.2 Deployment Design: Efficient MAPLE Execution on Hardware

MoE inference is executed by GPU kernels specialised to a fixed routing degree: the number of experts each token selects is baked into the token permutation, the expert GEMM grouping, and the output combination. Existing kernels assume a single configuration-invariant top-k, so a naive deployment runs every layer through the same generic path and discards the structure that we produce.

Sensitivity-guided allocation, however, does not spread the reduction evenly: the least sensitive layers absorb most of it, and some reach the lower bound $k_l = 1$. Such single-expert layers are common under **MAPLE** and open room for further latency optimisation. We exploit this with **MAPLE**'s top-1 routing kernel, implemented in CUDA and integrated into vLLM and SGLang, which assigns one block of B threads to each of the T tokens in a batch: rather than maintaining a sorted k_l-element candidate list per thread and merging these across the block, at $O(Bk_l)$ shared memory, each thread keeps one running maximum, and the block reduces them in $\log B$ steps with $O(B)$ shared memory. The kernel's benefit compounds with the allocation's. Reducing k_l already shrinks the downstream work, since a layer allocated k_l experts permutes $k_l T$ rows, runs its GEMM over $k_l T$ rows, and combines k_l contributions per token; the top-1 kernel removes what remains at the entry point and emits a $T \times 1$ routing tensor, so the downstream path sees T rows rather than a padded $k_l T$ buffer. The same holds under expert parallelism, where a $k_l = 1$ layer issues one transfer per token instead of a k_l-way scatter. Both paths are bound at construction rather than per call, so the forward pass adds no dispatch logic and both backends see the shapes they would under uniform routing. Table 1 summarises the two regimes.

Table 1: Execution requirements of the routed branch under the two regimes.

	Routing kernel		Downstream path		Expert-parallel
Path	**Selection**	**Shared memory**	**Routing metadata**	**GEMM rows**	**Destinations per token**
Generic ($k_l > 1$)	top-k_l	$O(Bk_l)$	$T \times k_l$	$k_l T$	k_l
MAPLE ($k_l = 1$)	$\arg\max$	$O(B)$	$T \times 1$	T	1

4 EXPERIMENTS

4.1 EXPERIMENTAL SETTINGS

Model architectures. We evaluate **MAPLE** on four representative MoE models: DeepSeek-MoE-16B (Dai et al., 2024), DeepSeek-V2-Lite (DeepSeek-AI, 2024), Moonlight-16B-A3B (Liu et al., 2025), and OLMoE-1B-7B (Muennighoff et al., 2024). We retain 75% of the total expert-activation budget across MoE layers, counting both shared and routed experts, while keeping shared experts unchanged. The resulting routed budgets are 108/162 for DeepSeek-MoE-16B, 104/156 for DeepSeek-V2-Lite and Moonlight-16B-A3B, and 96/128 for OLMoE-1B-7B. Thus, routed-expert retention is 66.7% for the first three models and 75% for OLMoE. Throughout the paper, budget percentages refer to the total expert-activation budget, as defined in Appendix A.2.

Evaluation benchmarks. We evaluate all four models on five short-form MCQA benchmarks: RTE (Wang et al., 2019), BoolQ (Clark et al., 2019), PIQA (Bisk et al., 2020), and ARC-E/ARC-C (Clark et al., 2018). On DeepSeek-MoE-16B, we additionally evaluate long-form generative reasoning with GSM8K (0-shot CoT) (Cobbe et al., 2021) and BBH (3-shot CoT) (Srivastava et al., 2022; Suzgun et al., 2023).

Implementation and deployment details. All experiments use NVIDIA RTX A6000 GPUs with 48 GB memory and a 75% total expert-activation budget, with accuracy range as the default sensitivity metric. **MAPLE** computes a sensitivity-guided allocation and refines it using 10 candidates over 10 generations, retaining the top two elites per generation while enforcing the global budget and per-layer bounds. For end-to-end serving, we implement **MAPLE** in SGLang 0.4.6 and vLLM 0.27.1, and evaluate single-GPU, two-GPU pipeline-parallel (PP), and two-GPU expert-parallel (EP) configurations. Latency and throughput are averaged over five runs.

Detailed configurations and algorithmic details are provided in Appendix A.4.

4.2 EXPERIMENTAL RESULTS

Overall performance. According to Table 2, **MAPLE** improves average accuracy over the full-budget baseline by 2.63, 6.44, and 4.13 points on DeepSeek-MoE-16B, DeepSeek-V2-Lite, and OLMoE-1B-7B, respectively, while using only 75% of the original expert-activation budget; on Moonlight-16B-A3B, it is 0.62 points below baseline but still the best-performing method at this budget. Compared with the strongest baseline in each column, **MAPLE** improves average accuracy by 3.52, 5.79, 1.14, and 3.61 points across the four models, achieving the best average accuracy on every model evaluated. These results show that uniform expert allocation is often suboptimal, and that sensitivity-guided heterogeneous allocation can match or exceed it with fewer active experts.

Table 2: Mean over five runs at a 75% expert-activation budget. Baseline: full budget; bold: best compressed method per model–task.

	DeepSeek-MoE-16B						DeepSeek-V2-Lite					
Method	**ARC-C**	**ARC-E**	**BoolQ**	**PIQA**	**RTE**	**Avg.**	**ARC-C**	**ARC-E**	**BoolQ**	**PIQA**	**RTE**	**Avg.**
Baseline	48.49	65.09	80.03	80.09	84.84	71.71	44.82	64.74	67.40	79.82	53.43	62.04
SparseGPT	46.15	62.11	79.69	79.33	81.22	69.70	44.48	64.91	67.34	78.98	57.76	62.69
Wanda	45.81	62.98	79.72	79.27	82.67	70.09	44.15	64.91	67.31	78.67	54.51	61.91
RIA	46.48	63.16	79.72	79.49	80.87	69.94	45.15	64.04	65.08	78.99	49.82	60.62
EEP	47.49	66.14	79.60	77.86	83.03	70.82	42.47	62.46	67.74	78.40	49.82	60.18
MoNE	47.15	64.04	76.52	76.44	81.46	69.12	42.81	62.11	65.86	78.30	53.42	60.50
LExI	47.49	67.19	79.85	78.35	74.37	69.45	47.16	64.21	68.78	76.82	54.15	62.22
MAPLE	**51.50**	**71.40**	**82.38**	**80.85**	**85.55**	**74.34**	**50.84**	**68.60**	**73.43**	**80.20**	**69.31**	**68.48**
	Moonlight-16B-A3B						**OLMoE-1B-7B**					
Method	**ARC-C**	**ARC-E**	**BoolQ**	**PIQA**	**RTE**	**Avg.**	**ARC-C**	**ARC-E**	**BoolQ**	**PIQA**	**RTE**	**Avg.**
Baseline	77.26	88.60	76.54	79.71	69.68	78.36	41.47	54.91	65.96	78.35	77.98	63.73
SparseGPT	75.25	85.79	75.14	78.51	65.34	76.01	43.47	55.09	67.06	77.37	77.61	64.12
Wanda	73.58	85.61	75.08	78.18	64.62	75.41	43.48	54.61	66.88	77.20	77.26	63.89
RIA	76.25	84.25	74.20	78.45	68.31	76.29	43.14	55.96	64.74	78.34	79.06	64.25
EEP	76.60	85.82	74.50	79.65	66.43	76.60	43.83	57.19	66.45	75.19	76.17	63.77
MoNE	74.09	85.80	**77.37**	77.92	65.53	76.14	33.78	46.15	62.41	76.18	77.62	59.23
LExI	67.89	81.58	70.61	78.67	**71.12**	73.97	30.43	38.95	63.73	76.82	53.07	52.60
MAPLE	**77.59**	**86.26**	74.65	**80.16**	70.04	**77.74**	**44.81**	**63.16**	**68.62**	**78.94**	**83.75**	**67.86**

Long-form generative tasks. Beyond short-form MCQA, we evaluate **MAPLE** on GSM8K and BBH, both requiring multi-step chain-of-thought generation. At a 75% budget on DeepSeek-MoE-16B, **MAPLE** stays within 0.61 points of the full-budget GSM8K baseline while reducing total evaluation time by 53.0%, and improves BBH accuracy by 0.90 points with a 5.9% end-to-end reduction. The smaller BBH gain is largely due to its 3-shot prompting and associated non-expert overhead: as shown in Table 3, routed-expert time itself decreases by 21.5% (2,162.32 s to 1,696.54 s). Appendix A.5 provides the full wall-clock breakdown. These results show that **MAPLE** remains effective on long-form generative reasoning.

Table 3: Long-form generation results on DeepSeek-MoE-16B at a 75% total expert-activation budget. Routed-expert time is obtained from the BBH wall-clock profile.

	GSM8K (0-shot)				BBH (3-shot CoT)					
Config.	**Acc. (%)**	**Δ**	**Total (s)**	**Δ**	**Acc. (%)**	**Δ**	**Total (s)**	**Δ**	**Expert (s)**	**Δ**
Full (100%)	58.53	–	492.26	–	43.43	–	5,776.97	–	2,162.32	–
MAPLE (75%)	57.92	−0.61	**231.48**	**−53.0%**	**44.33**	**+0.90**	**5,434.15**	**−5.9%**	**1,696.54**	**−21.5%**

Efficiency analysis. Tables 4 and 5 characterise **MAPLE**'s offline–online efficiency trade-off. Table 4 reports the mean per-task calibration and genetic-search cost per model across all five tasks details in Appendix A.6, while Table 5 measures realised serving gains in SGLang and vLLM. Our allocation-aware execution path further specialises $k_l = 1$ layers by removing redundant routing metadata, token permutation, and multi-source combine operations. In vLLM, we further fuse top-1 routing with expert execution to reduce launch and intermediate-materialisation overhead, yielding additional latency and throughput gains across single-GPU, pipeline-parallel, and expert-parallel settings. **MAPLE** keeps the full expert pool and only changes how many experts run per token, so the gain is in reduced active computation and latency.

Table 4: Mean per-task calibration and genetic-search time for each model on a single RTX A6000.

Model	**Total (h)**
DeepSeek-MoE-16B	1.48
DeepSeek-V2-Lite	1.36
Moonlight-16B-A3B	1.96
OLMoE-1B-7B	1.22

Table 5: DeepSeek-MoE-16B serving on ARC-Easy at 75% budget. Gains vs. Full; PP/EP: pipeline/expert parallelism. † Generic Kernel; MAPLE: Optimized Kernel.

Engine	**Setting**	**Full (s)**	**MAPLE (s)†**	**MAPLE (s)**	**Time ↓**	**Tput ↑**
SGLang	1 GPU	2.860	2.680	**1.940**	32.2%	47.4%
	2 GPU, PP	3.660	3.660	**3.470**	5.2%	5.5%
	2 GPU, EP	9.056	8.809	**8.701**	3.9%	4.1%
vLLM	1 GPU	3.915	3.745	**3.736**	4.58%	4.80%
	2 GPU, PP	13.703	11.819	**11.762**	14.16%	16.50%
	2 GPU, EP	16.512	15.388	**15.203**	7.93%	8.61%

4.3 Ablation Study

Sensitivity metrics. Table 6 compares three sensitivity signals: accuracy range, weight variance, and output deviation. Against the 84.84% baseline, accuracy range alone is strongest, gaining 0.71 points by generation two. Adding weight variance shrinks the gain to 0.36 and delays convergence to generation four. Substituting output deviation performs worse, falling 0.72 points below baseline at generation six. Without accuracy range, weight and output variance together perform worst, losing 1.08 points over ten generations. All three combined recover the same gain as accuracy range alone,

Table 6: Ablation of sensitivity metrics (Acc.: accuracy range, Weight: weight variance, Output: output deviation). Best Gen. is the generation at which the best accuracy is reached.

	Acc.	**Weight**	**Output**	**Best Gen.**	**Best Acc.**	**Δ**
1				–	84.84	–
2	✓			2	85.55	+0.71
3	✓	✓		4	85.20	+0.36
4	✓		✓	6	84.12	-0.72
5		✓	✓	10	83.75	-1.08
6	✓	✓	✓	9	85.55	+0.71

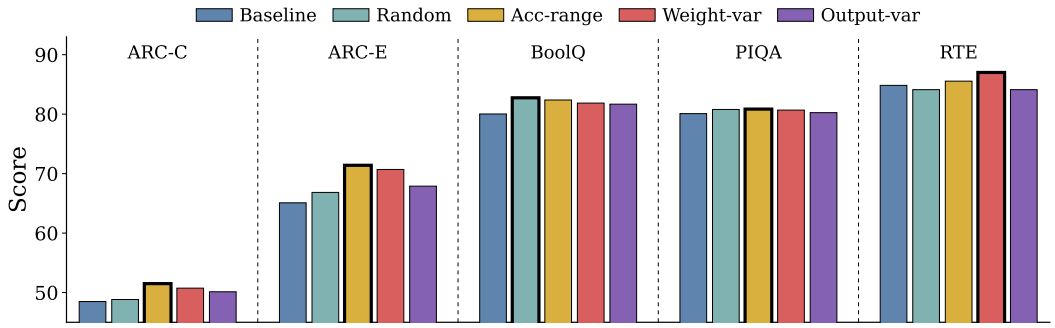

Figure 2: Performance of different sensitivity metrics in **MAPLE** under the 75% expert budget across five tasks. Bold bar outlines indicate the best result within each task.

but need over four times as many generations. Figure 2 shows that unguided genetic search already outperforms the uniform baseline on most tasks under the same 75% expert budget, confirming the benefit of heterogeneous layer-wise allocation. This also indicates that the gains are not solely tied to a particular sensitivity metric. Nevertheless, accuracy-range guidance directs the search towards stronger allocations, with especially clear improvements on ARC-C and ARC-E, and achieves the best overall performance. We therefore adopt accuracy range as the default sensitivity metric.

Refinement and stability. Table 7 ablates refinement under a fixed budget and sensitivity metric. At a 75% activation budget, the closed-form solution averages 71.40%, only 0.31 points below the 100% baseline (71.71%), preserving near-full-budget performance before search. Refinement improves every task by 0.67–7.01 points; across five seeds, each gain exceeds its standard deviation, indicating stable improvements. Figure 3 examines the refinement dynamics across tasks. The search converges within a handful of generations on every benchmark, despite substantially different layer-wise sensitivity landscapes, supporting the use of lightweight local refinement rather than an expensive global search.

Table 7: Effect of refinement and seed stability. Closed-form denotes the initial allocation, +Refinement applies the proposed refinement step, and Seed Std. reports the standard deviation of +Refinement accuracy across five independent refinement seeds.

Task	Closed-form	+Refinement	Δ	Seed Std.
ARC-C	50.83	51.50	+0.67	±0.11
ARC-E	64.39	71.40	+7.01	±0.90
BoolQ	79.20	82.38	+3.18	±0.45
PIQA	79.92	80.85	+0.93	±0.21
RTE	82.67	85.55	+2.88	±0.75

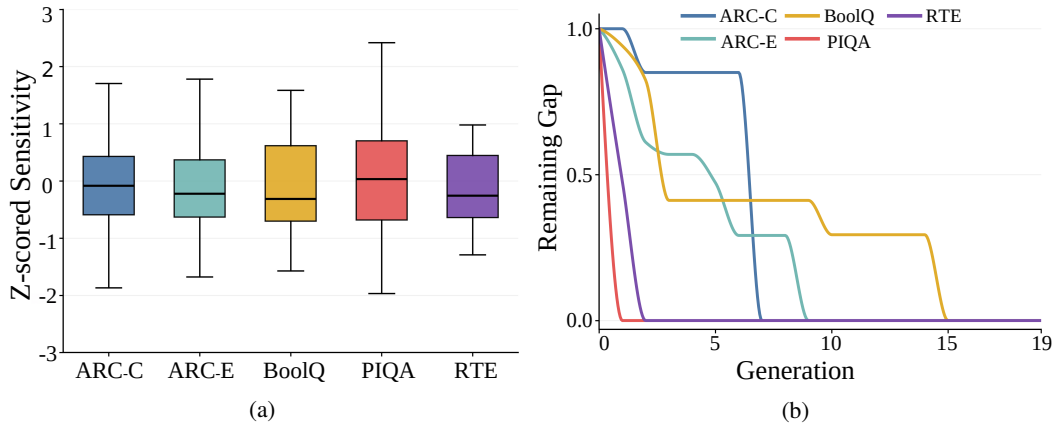

Figure 3: (a) Task-wise standardised sensitivity distributions under the 75% expert budget. (b) Normalised convergence trends across tasks, where Remaining Gap denotes the performance difference from the best-found solution.

Robustness of the calibration signal. Table 8 tests whether **MAPLE**'s gains hold under weaker calibration signals, on DeepSeek-MoE-16B at a 75% budget. A single allocation jointly calibrated across all five tasks (*Reusable*) still delivers strong accuracy without per-task recalibration. **MAPLE** also transfers well: applying each task's searched allocation directly to the other four (*Cross-task transfer*, averaged over the 20 off-diagonal source–target pairs below) performs on par with established pruning methods despite never seeing the target task, indicating the sensitivity patterns it identifies are genuine, transferable properties of the model's layers rather than a fit to one evaluation set. As an adversarial control, we replace task accuracy with negative token-level NLL on two generic corpora, C4 and FineWeb-Edu, sharing no examples, labels, or format with any evaluation task. Together, these results show **MAPLE** needs no per-task recalibration, generalises across tasks, and is not exploiting the search procedure itself.

Table 8: Robustness of **MAPLE**'s calibration signal on DeepSeek-MoE-16B at a 75% budget. Reusable jointly calibrates a single allocation on all five tasks; C4/FineWeb-Edu use generic-corpus NLL. Transfer reports source-to-target reuse, with the final row averaging the 20 off-diagonal pairs.

Setting	Signal / Source	ARC-C	ARC-E	BoolQ	PIQA	RTE	Avg.
Calibration	**Reusable (5-task)**	49.83	68.95	82.05	78.62	83.39	**72.57**
	C4-NLL	44.15	65.26	78.99	79.43	80.14	69.59
	FineWeb-Edu-NLL	44.48	62.11	76.45	79.76	81.59	68.88
Transfer	ARC-C	51.50	64.56	79.24	78.40	79.06	70.55
	ARC-E	44.82	71.40	78.75	78.67	76.90	70.11
	BoolQ	40.80	64.56	82.38	79.16	81.95	69.77
	PIQA	44.48	64.74	79.14	80.85	79.42	69.73
	RTE	44.48	66.49	78.17	79.00	85.55	70.74
	Cross-task avg.	43.65	65.09	78.83	78.81	79.33	**69.14**

Expert budget. **MAPLE**'s advantage over Wanda and SparseGPT holds at every budget level and widens at the tighter 50% budget, suggesting heterogeneous reallocation becomes more valuable as pruning is more likely to remove capacity from layers that still need it. Heterogeneous allocation also improves over the uniform baseline at 100% budget, indicating uniform allocation itself leaves room for better capacity placement independent of any budget reduction. We emphasise that **MAPLE** is itself a pruning method, not fine-tuning: like Wanda and SparseGPT, it reduces active computation under an explicit budget with no gradient updates. The difference is what it prunes: activations rather than weights, deciding which router-ranked experts run at inference time. This is why **MAPLE** keeps the full expert pool and needs no sparse-matrix kernels, while still competing directly with weight-level pruning at the same nominal budget.

Table 9: ARC-C / RTE accuracy (%) on DeepSeek-MoE-16B under different total expert-activation budgets. The 100% uniform baseline is 48.49 / 84.84.

Budget	Wanda	SparseGPT	MAPLE
50%	41.14 / 72.92	39.46 / 72.20	**46.49 / 79.78**
75%	45.81 / 82.67	46.15 / 81.22	**51.50 / 85.55**
100%	–	–	**51.84 / 88.09**

5 Conclusions

We introduced **MAPLE**, a plug-and-play, training-free framework that reformulates expert allocation in MoE Transformers from a uniform design choice into a principled layer-wise optimisation problem. By combining sensitivity-aware profiling, closed-form budget allocation, and lightweight evolutionary refinement, **MAPLE** adapts routing capacity to the heterogeneous demands of individual layers without weight updates or architectural modifications. Across four MoE models, **MAPLE** consistently outperforms pruning-based alternatives under a 75% total expert-activation budget and, on three models, even surpasses the full-budget uniform configuration while activating fewer experts. Beyond algorithmic allocation, we develop a MAPLE-aware framework for SGLang and vLLM that specialises GPU kernels to the layer-wise routing degree, eliminating redundant routing and aggregation overhead and converting structured expert sparsity into latency and throughput gains beyond nominal FLOP reductions. Collectively, these results challenge uniform expert allocation as the default paradigm and establish algorithm–system co-design for layer-wise capacity redistribution as a promising direction for accurate and efficient MoE inference.

AI USE STATEMENT

We used generative AI tools for two purposes: editing and debugging code written by the authors, and improving the readability and language of the manuscript. They were not used for any other part of the research, including research ideas and hypotheses, the conceptual framework, experimental design, method implementation, mathematical claims or derivations, data generation or cleaning, and interpretation of results. Translation and qualitative data analysis are not applicable. The authors reviewed and tested all AI-assisted code and edits, and take full responsibility for the content of this paper.

ETHICS STATEMENT

This work focuses on improving the inference efficiency of mixture-of-experts language models. It does not involve human subjects, the collection of personal or sensitive data, or the release of a new dataset. The experiments are conducted using existing models and evaluation benchmarks under their respective licenses and usage conditions. To the best of our knowledge, this work does not raise additional ethical concerns beyond those already associated with the use of the underlying models and benchmarks.

REPRODUCIBILITY STATEMENT

We have taken several steps to support the reproducibility of this work. The main paper describes the MAPLE allocation objective, routing-budget formulation, and deployment design, while the experimental sections provide the model configurations, evaluation settings, and metrics used in our experiments. Additional implementation details and ablation studies are provided in the appendix.

We also provide the implementation code and experimental configuration files in the supplementary materials to facilitate reproduction of the reported results. The supplementary materials include the core code required to reproduce the MAPLE allocation procedure and the corresponding experiments.

REFERENCES

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*, 2015.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

Tom B. Brown, Benjamin Mann, Nick Ryder, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:1877–1901, 2020.

Mingyu Cao, Gen Li, Jie Ji, Jiaqi Zhang, Xiaolong Ma, Shiwei Liu, and Lu Yin. Condense, don't just prune: Enhancing efficiency and performance in MoE layer pruning. *arXiv preprint arXiv:2412.00069*, 2024.

Krishna Teja Chitty-Venkata, Sandeep Madireddy, Murali Emani, and Venkatram Vishwanath. Lexi: Layer-adaptive active experts for efficient moe model inference, 2025. URL `https://arxiv.org/abs/2509.02753`.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, et al. PaLM: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24:1–113, 2023.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge, 2018.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Damai Dai, Chengqi Deng, Chenggang Zhao, R. X. Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y. Wu, et al. DeepSeekMoE: Towards ultimate expert specialization in mixture-of-experts language models, 2024.

DeepSeek-AI. DeepSeek-V2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pp. 4171–4186, 2019.

Zhen Dong, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. HAWQ: Hessian AWare quantization of neural networks with mixed-precision. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 293–302, 2019.

Zhen Dong, Zhewei Yao, Daiyaan Arfeen, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. HAWQ-V2: Hessian aware trace-weighted quantization of neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pp. 18518–18529, 2020.

Zican Dong, Han Peng, Peiyu Liu, Wayne Xin Zhao, Dong Wu, Feng Xiao, and Zhifeng Wang. Domain-specific pruning of large mixture-of-experts models with few-shot demonstrations. *arXiv preprint arXiv:2504.06792*, 2025.

Nan Du, Yanping Huang, Andrew M. Dai, et al. GLaM: Efficient scaling of language models with mixture-of-experts. In *Proceedings of ICML*, pp. 5547–5569, 2022.

Andries P. Engelbrecht. A new pruning heuristic based on variance analysis of sensitivity information. *IEEE Transactions on Neural Networks*, 12(6):1386–1399, 2001.

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

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations (ICLR)*, 2019.

Elias Frantar and Dan Alistarh. SparseGPT: Massive language models can be accurately pruned in one-shot. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, pp. 10323–10337, 2023.

Trevor Gale, Deepak Narayanan, Cliff Young, and Matei Zaharia. Megablocks: Efficient sparse training with mixture-of-experts, 2022. URL `https://arxiv.org/abs/2211.15841`.

Chongyang Gao, Kezhen Chen, Jinmeng Rao, Baochen Sun, Ruibo Liu, Daiyi Peng, Yawen Zhang, Xiaoyuan Guo, Jie Yang, and V. S. Subrahmanian. Higher layers need more LoRA experts. *arXiv preprint arXiv:2402.08562*, 2024.

Suhan Guo, Bilan Lai, Suorong Yang, Jian Zhao, and Furao Shen. Sensitivity pruner: Filter-level compression algorithm for deep neural networks. *Pattern Recognition*, 140:109508, 2023. doi: 10.1016/j.patcog.2023.109508.

Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 28, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, et al. Parameter-efficient transfer learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pp. 2790–2799, 2019.

Edward J. Hu, Yelong Shen, Phillip Wallis, et al. LoRA: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2022.

Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7132–7141, 2018.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pp. 448–456, 2015.

Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention, 2023. URL `https://arxiv.org/abs/2309.06180`.

Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 2, pp. 598–605, 1990.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, et al. GShard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2021.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of EMNLP*, pp. 3045–3059, 2021.

Pingzhi Li, Zhenyu Zhang, Prateek Yadav, Yi-Lin Sung, Yu Cheng, Mohit Bansal, and Tianlong Chen. Merge, then compress: Demystify efficient SMoE with hints from its routing policy. In *International Conference on Learning Representations (ICLR)*, 2024.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 4582–4597, 2021.

Enshu Liu, Junyi Zhu, Zinan Lin, Xuefei Ning, Matthew B Blaschko, Shengen Yan, Guohao Dai, Huazhong Yang, and Yu Wang. Efficient expert pruning for sparse mixture-of-experts language models: Enhancing performance and reducing inference costs. *arXiv preprint arXiv:2407.00945*, 2024.

Jingyuan Liu, Jianlin Su, Xingcheng Yao, et al. Muon is scalable for LLM training, 2025.

Xudong Lu, Qi Liu, Yuhui Xu, Aojun Zhou, Siyuan Huang, Bo Zhang, Junchi Yan, and Hongsheng Li. Not all experts are equal: Efficient expert pruning and skipping for mixture-of-experts large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. LLM-Pruner: On the structural pruning of large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, 2023.

Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019.

Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11264–11272, 2019.

Niklas Muennighoff et al. OLMoE: Open mixture-of-experts language models. *arXiv preprint arXiv:2409.02060*, 2024.

Zihan Qiu, Zeyu Huang, Shuang Cheng, Yizhi Zhou, Zili Wang, Ivan Titov, and Jie Fu. Layerwise recurrent router for mixture-of-experts. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=eWNEqdH0vk.

Colin Raffel, Noam Shazeer, Adam Roberts, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations (ICLR)*, 2017.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615*, 2022.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

Mingjie Sun, Zhuang Liu, Anna Bair, and J. Zico Kolter. A simple and effective pruning approach for large language models. In *International Conference on Learning Representations (ICLR)*, 2024.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, and Jason Wei. Challenging BIG-bench tasks and whether chain-of-thought can solve them. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 13003–13051, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.824. URL https://aclanthology.org/2023.findings-acl.824/.

Shawn Tan, Yikang Shen, Rameswar Panda, and Aaron Courville. Scattered mixture-of-experts implementation, 2024. URL https://arxiv.org/abs/2403.08245.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, et al. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023a.

Hugo Touvron, Louis Martin, Kevin Stone, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017.

Singanallur V. Venkatakrishnan, Charles A. Bouman, and Brendt Wohlberg. Plug-and-play priors for model based reconstruction. In *IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pp. 945–948, 2013.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 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 (ACL)*, pp. 5797–5808, 2019.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding, 2019.

Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. CBAM: Convolutional block attention module. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 3–19, 2018.

Hao Wu, Patrick Judd, Xiaojie Zhang, Mikhail Isaev, and Paulius Micikevicius. Integer quantization for deep learning inference: Principles and empirical evaluation. *CoRR*, abs/2004.09602, 2020. URL https://arxiv.org/abs/2004.09602.

Yanyue Xie, Zhi Zhang, Ding Zhou, Cong Xie, Ziang Song, Xin Liu, Yanzhi Wang, Xue Lin, and An Xu. Moe-pruner: Pruning mixture-of-experts large language model using the hints from its router, 2024. URL https://arxiv.org/abs/2410.12013.

Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language models. *arXiv preprint arXiv:2106.10199*, 2022.

Geng Zhang, Yuxuan Han, Yuxuan Lou, Yiqi Zhang, Wangbo Zhao, and Yang You. MoNE: Replacing redundant experts with lightweight novices for structured pruning of MoE. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2507.00390.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. AdaLoRA: Adaptive budget allocation for parameter-efficient fine-tuning. In *International Conference on Learning Representations (ICLR)*, 2023.

Yingtao Zhang, Haoli Bai, Haokun Lin, Jialin Zhao, Lu Hou, and Carlo Vittorio Cannistraci. Plug-and-play: An efficient post-training pruning method for large language models. In *International Conference on Learning Representations (ICLR)*, 2024.

Chenggang Zhao, Shangyan Zhou, Liyue Zhang, Chengqi Deng, Zhean Xu, Yuxuan Liu, Kuai Yu, Jiashi Li, and Liang Zhao. Deepep: an efficient expert-parallel communication library. https://github.com/deepseek-ai/DeepEP, 2025.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. Sglang: Efficient execution of structured language model programs, 2024. URL https://arxiv.org/abs/2312.07104.

Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew M. Dai, Quoc V. Le, and James Laudon. Mixture-of-experts with expert choice routing. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, 2022.

Barret Zoph, Irwan Bello, Sameer Kumar, et al. ST-MoE: Designing stable and transferable sparse expert models. *arXiv preprint arXiv:2202.08906*, 2022.

A Appendix

A.1 Full Algorithm

Algorithm 1 summarises the complete three-step **MAPLE** pipeline.

A.2 Expert Budget

We measure the expert budget by the total number of active expert units per token across all MoE layers, including both shared experts and activated routed experts. Let $\mathcal{M}$ denote the set of MoE layers and $L_{\mathrm{MoE}} = |\mathcal{M}|$. For each layer $l \in \mathcal{M}$, let h_l denote the number of shared expert units, $k_l^{(0)}$ the original number of active routed experts, and k_l the number assigned by MAPLE. The original and allocated budgets are

$$\begin{aligned} B_{\mathrm{full}} &= \sum_{l \in \mathcal{M}} \left(h_l + k_l^{(0)} \right), \\ B(\mathbf{k}) &= \sum_{l \in \mathcal{M}} \left(h_l + k_l \right). \end{aligned} \tag{7}$$

Algorithm 1: Pseudocode for **MAPLE**

Input: MoE model with L layers; base expert count k_0; total budget K; valid range $[k_{\min}, k_{\max}]$; calibration set $\mathcal{D}_{\text{cal}}$; population size P; generations G
Output: Final allocation $\mathbf{k}^{\text{best}}$

Step 1: Layer-wise Sensitivity Modelling;
for $l \leftarrow 1$ **to** L **do**
  Probe layer l by varying its expert count from 1 to k_0, with all other layers fixed at k_0;
  Record $A_l(e)$ for each candidate count e, and set the preferred count $k_l^* = \min\{e \mid A_l(e) = \max_{e'} A_l(e')\}$;
  Compute the sensitivity score $s_l = \max_e A_l(e) - \min_e A_l(e)$, $w_l = \max(s_l, \epsilon)$ (accuracy range; see Section 3.1.1 for alternative metrics);
Step 2: Sensitivity-guided Closed-form Expert Allocation;
if $\sum_{l=1}^{L} k_l^* < K$ **then**
  Let $\Delta = K - \sum_{l=1}^{L} k_l^*$;
  Allocate the surplus to less sensitive layers: $k_l^{\text{cont}} = k_l^* + \Delta \cdot \frac{1/w_l}{\sum_{j=1}^{L} 1/w_j}$ for each layer l;
else
  Let $\Gamma = \sum_{l=1}^{L} k_l^* - K$;
  Remove the excess from less sensitive layers: $k_l^{\text{cont}} = k_l^* - \Gamma \cdot \frac{1/w_l}{\sum_{j=1}^{L} 1/w_j}$ for each layer l;
Clip $\mathbf{k}^{\text{cont}}$ to $[k_{\min}, k_{\max}]$ and project it to integers via the largest-remainder method to obtain $\mathbf{k}^{\text{sens}}$;
Step 3: Sensitivity-constrained Genetic Refinement;
Initialise the population $\mathcal{P}$ around $\mathbf{k}^{\text{sens}}$ by sampling $\tilde{k}_l = k_l^{\text{sens}} + \epsilon_l^{\text{g}}$, with $\epsilon_l^{\text{g}} \sim \mathcal{N}(0, \sigma_l^2)$ and $\sigma_l^2 \propto \frac{1}{s_l + \varepsilon_{\text{g}}}$;
for $g \leftarrow 1$ **to** G **do**
  Evaluate each candidate $\mathbf{k} \in \mathcal{P}$ on $\mathcal{D}_{\text{cal}}$ and compute its fitness $f(\mathbf{k}) = A(\mathbf{k})$;
  Keep the top-2 candidates in $\mathcal{P}$ as elites;
  Refill $\mathcal{P}$ by perturbing elite solutions with sensitivity-controlled Gaussian noise, then project each result back to the feasible set;
return $\mathbf{k}^{\text{best}} = \arg\max_{\mathbf{k} \in \mathcal{P}} f(\mathbf{k})$;

Table 10: Expert activation budgets at $\rho = 0.75$. L_{MoE} is the number of MoE layers; h and $k^{(0)}$ are the shared expert units and original active routed experts per MoE layer, respectively. H is the fixed shared budget, K is the allocated routed budget, and $B = H + K$ is the final total budget. All total budgets are counted per token across MoE layers.

Model	L_{MoE}	h	$k^{(0)}$	B_{full}	H	K	B
DeepSeek-MoE-16B	27	2	6	216	54	108	162
DeepSeek-V2-Lite	26	2	6	208	52	104	156
Moonlight-16B-A3B	26	2	6	208	52	104	156
OLMoE-1B-7B	16	0	8	128	0	96	96

Shared experts remain unchanged. Writing their total contribution as $H = \sum_{l \in \mathcal{M}} h_l$, the routed budget for a retained-budget ratio ρ is

$$K = \rho B_{\text{full}} - H, \qquad \sum_{l \in \mathcal{M}} k_l = K. \tag{8}$$

Thus, a 75% budget retains 75% of the combined shared and routed activation budget. The constraint applies to the sum across MoE layers; individual layers may retain different numbers of routed experts. Dense feed-forward layers are excluded from this accounting.

Table 10 gives the budget accounting for the four models. For example, DeepSeek-MoE-16B has 27 MoE layers, each with two shared expert units and six active routed experts in the original configu-

ration. The full budget is therefore $27(2 + 6) = 216$. At $\rho = 0.75$, the total budget is 162. Keeping all 54 shared expert units leaves a routed budget of $K = 108$.

For the first three models, the routed budget retains two-thirds of its original value, while the combined budget retains exactly 75%. For OLMoE, which has no shared experts, these two retention ratios coincide. Shared expert units follow the model configuration, including when multiple units are implemented as a single wider shared module. This accounting describes expert activations, rather than stored parameters or end-to-end FLOPs.

A.3 Benchmark and Evaluation Details

We evaluate MAPLE on seven benchmarks spanning five short-form benchmarks across all models and two long-form generative benchmarks on DeepSeek-MoE-16B, spanning short-form classification and multiple-choice question answering (MCQA), as well as long-form generative reasoning. The evaluation suite is designed to cover both tasks whose outputs are short categorical decisions and tasks that require autoregressive generation over substantially longer reasoning trajectories.

Short-form MCQA and classification. We use RTE (Wang et al., 2019), BoolQ (Clark et al., 2019), PIQA (Bisk et al., 2020), ARC-Easy (ARC-E), and ARC-Challenge (ARC-C) (Clark et al., 2018). These benchmarks primarily require the model to select or produce a short final answer, and therefore place relatively limited demand on long autoregressive decoding.

RTE is a textual entailment benchmark in which the model determines whether a given premise entails a hypothesis. BoolQ consists of yes/no questions paired with passages and tests whether the model can infer the correct binary answer from the provided context. PIQA evaluates physical commonsense reasoning by asking the model to choose between two candidate solutions to an everyday physical problem. ARC-E and ARC-C are multiple-choice science question-answering benchmarks derived from standardised science examinations. ARC-E contains comparatively easier questions, whereas ARC-C contains a more challenging subset that typically requires stronger scientific knowledge and reasoning.

Long-form generative reasoning. To evaluate MAPLE under workloads with substantially longer decoding sequences, we additionally use GSM8K (Cobbe et al., 2021) and BBH (3-shot CoT) (Srivastava et al., 2022; Suzgun et al., 2023). Unlike the short-form benchmarks above, these evaluations use chain-of-thought (CoT) generation, causing the model to generate intermediate reasoning steps before producing its final answer. This distinction matters for our study because longer generations increase the number of decoding steps over which the per-layer routing configuration is repeatedly exercised.

GSM8K consists of grade-school mathematical word problems that typically require several arithmetic and reasoning steps. We evaluate GSM8K using a **0-shot CoT** setting. Here, the model receives the test problem and an instruction that elicits step-by-step reasoning, but it is not provided with any solved demonstration examples from the task. The model must therefore infer the required reasoning procedure from the instruction and the problem itself, generate an intermediate reasoning trace, and finally produce the answer.

BBH is a collection of challenging reasoning tasks selected from BIG-Bench and covers a broader range of capabilities, including logical reasoning, symbolic manipulation, algorithmic reasoning, and multi-step inference. We evaluate BBH using a **3-shot CoT** setting. In contrast to 0-shot CoT, the prompt contains three worked examples before the test instance. Each demonstration provides an example input together with its reasoning process and answer, after which the model generates a corresponding reasoning trajectory for the target instance. Consequently, the 3-shot BBH setting introduces both a longer input context and a long autoregressive output sequence.

0-shot versus 3-shot CoT. The distinction between the two settings concerns the number of in-context demonstrations rather than whether chain-of-thought reasoning is used. Both GSM8K and BBH require the model to generate intermediate reasoning steps. In the **0-shot CoT** setting used for GSM8K, no solved examples are included in the prompt; the model receives only the task instruction and the target problem. In the **3-shot CoT** setting used for BBH, three solved examples with reasoning traces are prepended to the target instance. Thus, 3-shot CoT increases the prefill

workload through the additional demonstration tokens, while both settings exercise autoregressive decoding through the generated reasoning trajectory.

This distinction also makes the two generative benchmarks complementary for evaluating serving efficiency. GSM8K isolates long-form reasoning without the additional context introduced by in-context demonstrations, whereas BBH evaluates the system under a combination of a longer prompt and multi-step generation. We therefore use the 3-shot BBH workload for the detailed serving-time breakdown in Section A.5.

A.4 Work Protocol

For each task, we use 128 examples from the training split for calibration. These examples are excluded from final evaluation. Because BBH has no official training split, we reserve a fixed subset of 600 examples for calibration and search, and evaluate on the remaining 5,911 examples. We use 0-shot chain-of-thought prompting for GSM8K and 3-shot chain-of-thought prompting for BBH.

Table 11: Calibration, search, and final evaluation protocol. Calibration and search examples are excluded from final evaluation.

Task	Calibration/search subset	Final evaluation split	#Eval.
RTE	128 training examples	Validation	277
BoolQ	128 training examples	Validation	3,270
PIQA	128 training examples	Validation	1,838
ARC-E	128 training examples	Validation	570
ARC-C	128 training examples	Validation	299
GSM8K	128 training examples	Test	1,319
BBH	600 held-out examples	Remaining examples	5,911

Allocation constraints. The final allocation can redistribute expert capacity across layers. We set

$$k_{\min} = 1, \qquad k_{\max} = 20,$$

and require

$$1 \le k_l \le 20, \qquad \sum_{l=1}^{L} k_l = K.$$

After computing the continuous closed-form allocation, we clip and round it to an integer allocation satisfying both the per-layer bounds and the global budget constraint.

Genetic refinement. The refined allocation is initialized with a population of 10 feasible candidates and optimized for 10 generations. The two candidates with the highest objective values are retained as elites in each generation. The remaining candidates are generated using sensitivity-controlled Gaussian perturbations and projected back to the feasible set. The final allocation is the candidate with the highest accuracy. We repeat the refinement with five independent random seeds for the stability analysis.

All baselines (SparseGPT, Wanda, RIA, EEP, MoNE, and LExI) are evaluated under the same model checkpoints, total expert-activation budgets, and evaluation protocols as MAPLE. Shared experts are kept unchanged, and the budget is applied to the routed-expert branch for all methods.

A.5 Per-Component Wall-Clock Breakdown on BBH

To understand where **MAPLE**'s serving-time reduction on the 3-shot BBH workload (Table 3) comes from, we profile the SGLang execution by component. Table 13 reports the percentage of total wall-clock time spent in each component under full-uniform routing and under **MAPLE** at a 75% budget.

In absolute time, routed-expert compute drops by 465.8 s under **MAPLE**, and framework/scheduler and router overhead also decrease slightly (by 96.1 s and 11.7 s, respectively). The remaining components increase modestly, by 230.1 s in total, mainly in attention (102.9 s) and normalisation/dense

Table 12: MAPLE search configuration.

Configuration	Setting
Sensitivity scan	$e = 1, \ldots, k_l^{(0)}$
Minimum active experts	$k_{\min} = 1$
Maximum active experts	$k_{\max} = 20$
Global budget	$\sum_l k_l = K$
Over-allocation	$k_l > k_l^{(0)}$ allowed
Population size	10
Number of generations	10
Elites per generation	2
Refinement seeds	5

Table 13: Per-component share of total wall-clock time on the complete 3-shot BBH evaluation (5,911 examples), DeepSeek-MoE-16B, SGLang single-GPU.

Component	**Full uniform (100%)**	**MAPLE (75%)**
Routed-expert compute	37.43%	31.22%
Attention (excl. KV writes)	13.09%	15.81%
KV-cache writes	1.11%	1.57%
Framework/scheduler gaps	30.08%	30.21%
Router/top-k/dispatch/scatter	7.21%	7.45%
Shared experts & other MoE	7.17%	8.17%
Norm/dense/residual	3.30%	4.49%
Embedding/logits, etc.	0.61%	1.07%

layers (53.4 s), with smaller contributions from shared experts, embedding/logits, and KV-cache writes. Overall, total wall-clock time falls from 5,776.97 s to 5,434.15 s, a saving of 342.8 s. This is the 5.9% end-to-end reduction reported in Table 3, and it is equivalent to a 6.3% throughput gain. Because the reduction in routed-expert compute alone exceeds the net saving, the BBH speedup is attributable to MAPLE's reduced routed-expert activation rather than to changes elsewhere in the serving pipeline. The breakdown is obtained from a single profiling run and is intended to characterise where time is spent, not to measure run-to-run variance.

A.6 Offline Cost Breakdown

For DeepSeek-MoE-16B, the sensitivity scan and closed-form allocation take 0.90 hours per task on average. Running the complete 10-generation genetic refinement takes 0.58 hours, giving a total offline cost of 1.48 hours per task. Across the five tasks, the best allocation is reached after 6.8 generations on average, corresponding to 0.39 hours of genetic search and 1.29 hours in total.

Table 14: Mean per-task offline time for DeepSeek-MoE-16B on a single RTX A6000. Full search runs for 10 generations, while the best allocation is reached after 6.8 generations on average.

Stage	**Full search (h)**	**To best (h)**
Sensitivity scan and closed form	0.90	0.90
Genetic refinement	0.58	0.39
Total	**1.48**	**1.29**