DECO: CLOSING THE GAP BETWEEN SPARSE MOE AND DENSE AT EQUAL TOTAL PARAMETERS

Anonymous authors
Paper under double-blind review

ABSTRACT

While sparse Mixture-of-Experts (MoE) scales model capacity without proportionally increasing computation, its massive total parameter footprint creates significant storage and memory-access bottlenecks, which hinder efficient edge-device deployment that simultaneously requires high performance, low computational cost, and small storage overhead. To achieve these properties, we present **DECO**, a **DE**nse-**CO**mpetitive sparse MoE architecture that closes the gap between existing MoE designs and dense Transformers, under a matched total parameter budget and an identical number of training tokens. We argue that this gap is not an inherent cost of sparsity but an artifact of how existing designs enforce it: some impose a hard TopK selection that blocks gradients and fixes one activation ratio for every token, others apply a heavy sparsity regularization that costs performance. DECO instead lets sparsity arise from the architecture itself through differentiable ReLU-based routing, along with learnable expert-wise scaling that balances routed against shared experts. Its experts are non-gated MLPs, and their activation function, NormSiLU, keeps the routed-expert activation ratio low and stable, so only light regularization is needed. Experiments demonstrate that the established MoE baselines consistently leave a performance gap to an equally-budgeted dense model, and DECO removes it while activating only 20% of routed experts across six parameter scales from 0.11B to 7B. Our specialized acceleration kernel delivers a 2.93$\times$ speedup on Jetson AGX Orin compared with dense inference. Code and checkpoints will be released.

1 INTRODUCTION

The scale of large language models (LLMs) has grown rapidly to achieve consistent performance gains across diverse tasks. The rising training and deployment costs for massive LLMs have made mixture-of-experts (MoE) an increasingly prominent model architecture (Shazeer et al., 2017). The key property of MoE is sparse activation, namely, activating a small subset of expert modules from a large pool of parameters. Therefore, MoE retains high capacity and strong performance while substantially reducing computation costs.

MoE has been extensively studied, from architecture design (Liu et al., 2024; Cai et al., 2025) to scaling laws and compute-optimal settings (Krajewski et al., 2024; Tian et al., 2025). Prior work primarily pursues two objectives: **high performance** and **low computation cost**. However, when it comes to edge-device deployment, a third non-negligible objective emerges: **small storage overhead**. Concretely, MoE with a large total parameter count demands substantial storage space. More critically, large MoE models may incur high memory-access costs when transferring experts between GPU high-bandwidth memory and shared memory (Li et al., 2025), or when moving offloaded parameters from disk/flash storage to GPU/NPU memory of edge devices. Such latency erodes the efficiency gains afforded by sparsity.

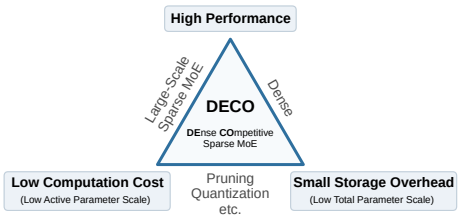


Figure 1: The "ideal triangle" of edge-device LLM: high performance, low computation cost, and a small total-parameter (storage) footprint must hold simultaneously. Each edge denotes the approaches that reach only the two properties it connects.

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.

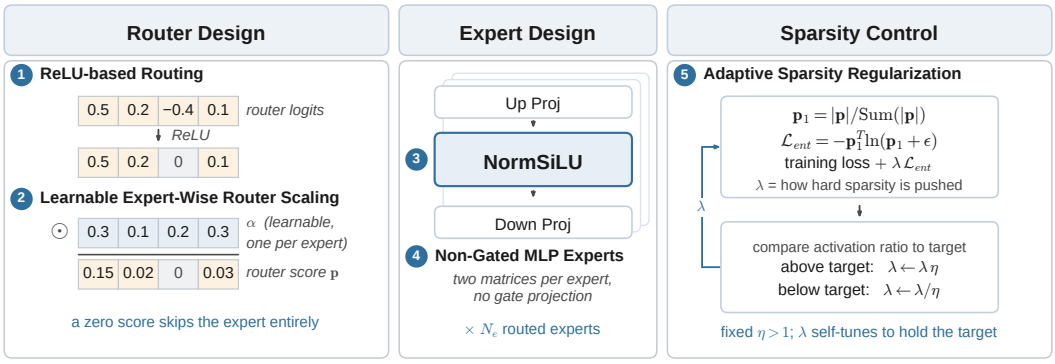


Figure 2: The overall architecture of DECO. For router design, we adopt ReLU-based routing enhanced by learnable expert-wise router scaling. For expert design, we propose NormSiLU as a better routed-expert activation function and employ non-gated MLP experts. For precise sparsity control, we employ adaptive sparsity regularization.

Therefore, as shown in Figure 1, an ideal MoE model for edge-device deployment should satisfy the above three objectives. To pursue this "ideal triangle", we ask two sharper questions:

> *Under matched total parameters and training tokens, established MoE designs still trail dense Transformers. Is this gap an inherent cost of sparsity or an artifact of how those designs enforce sparsity? What design changes remove the gap?*

A closely related study by Li et al. (2025) identifies the optimal settings of DeepSeek-V3-style MoE architectures that enable them to surpass dense models under matched total parameters and computation budget. However, due to the low per-token computation of sparse MoE, in that work, MoE settings are trained on substantially more tokens under the same computation budget. We adopt a stricter setting that requires exactly the same number of training tokens.

To achieve this goal, we propose **DECO** (Figure 2), a sparse MoE architecture that achieves **DE**nse-**CO**mpetitive performance by changing how sparsity is enforced.

For router design, conventional MoE models generally adopt TopK routing, whose discrete expert-selection decision is non-differentiable and fixes the number of active experts per token. To overcome both issues, we adopt **ReLU-based routing**, a differentiable paradigm in which continuous router scores are trained end-to-end and token-dependent activation ratios emerge by thresholding the scores at zero. Moreover, to mitigate output scale imbalances between shared and routed experts, while accounting for expert heterogeneity, we introduce **learnable expert-wise router scaling**, whose factors calibrate the contribution of individual routed experts.

The expert design is similarly optimized for stability and efficiency. Empirical analysis reveals that coupling ReLU-based routing with vanilla SiLU-activated experts results in two critical problems: a surging routed-expert activation ratio (Figure 4) and vanishing SiLU output magnitudes (Figure 6). To resolve these issues, we propose **NormSiLU**, which applies dual-stage normalization prior to the SiLU operator. NormSiLU stabilizes activation trends and reduces the activation ratio, alleviating the need for aggressive sparsity regularization. It also produces more stable and significant SiLU output magnitudes, promoting better utilization of expert parameters. Beyond the activation function, we employ **non-gated MLP experts** rather than standard gated variants, as they exhibit superior empirical compatibility with ReLU-based routing. Finally, to precisely control the activation ratio, we design an **adaptive sparsity regularization** that auto-scales the regularization strength.

As shown in Section 4, at the activation ratio of 20%, DECO closes the performance gap between established MoE designs and dense models when matched for total parameters and training tokens, while surpassing those baselines at all scales where baseline results are available.

Finally, we implement a tailored acceleration kernel for DECO to assess its practical acceleration benefit. Based on CUTLASS (Thakkar et al., 2023), the kernel leverages tensor cores to improve computational throughput and reduces memory-access overhead by exploiting the sparse activation. Overall, the kernel achieves a speedup of 2.93× compared with vanilla dense inference on Jetson AGX Orin, an actual edge device.

2 Related Work

To achieve high performance while curbing computational cost, recent work has increasingly adopted MoE as a mainstream architecture. An MoE typically comprises three components: a router, a set of experts, and an auxiliary training objective.

Router design. The router computes the weight of each expert and selects which to activate. It generally consists of a linear projection, an activation function, and post-processing of router scores.

The activation function controls the expert selection pattern. Many MoE designs use TopK, which forces each token to activate a fixed number of experts (Jiang et al., 2024; Dai et al., 2024). However, TopK is criticized for its inflexibility (an input-invariant number of active experts) and non-differentiability. TopP (Huang et al., 2024) selects experts by a threshold p, activating experts until their cumulative router score reaches at least p, thereby permitting token-dependent activation ratios. MoE++ (Jin et al., 2024) retains TopK but introduces zero-computation experts, which indirectly allows variable computation cost. To improve differentiability, ReMoE (Wang et al., 2024b) and BlockFFN (Song et al., 2025b) adopt ReLU for expert selection. Since ReLU naturally produces considerable zero outputs while remaining differentiable, it enables smoothly learnable activation ratios and delivers performance advantages.

Post-processing of router scores primarily normalizes expert weights to preserve a consistent output scale. Most designs use Softmax as the score normalizer. DeepSeek-V3 (Liu et al., 2024) instead applies element-wise Sigmoid followed by unit-sum normalization, which helps mitigate extremely skewed score distributions. Notably, DeepSeek-V3 also introduces a scalar scaling factor applied to router scores, which helps balance contributions between shared and routed experts.

Expert design. In most mainstream MoE models, each expert is a standard gated MLP with SiLU activation (SwiGLU) (Shazeer, 2020). DeepSeekMoE (Dai et al., 2024) shows the benefits of introducing fine-grained experts and shared experts but retains the SwiGLU backbone.

Auxiliary training objective. Aside from the language modeling loss, MoE models generally introduce auxiliary training objectives. The most common is a load-balancing loss, typically the auxiliary loss proposed by Fedus et al. (2022). To alleviate auxiliary-loss interference with language modeling, DeepSeek-V3 adopts a loss-free load-balancing policy without a differentiable objective (Wang et al., 2024a). MoE models with variable activation ratios often incorporate a sparsification objective. For example, ReMoE applies adaptive L1-norm regularization, and BlockFFN employs chunk-wise sparsification (Wang et al., 2024b; Song et al., 2025b).

Our position. DECO combines ReLU-based routing with learnable expert-wise scaling in place of a fixed scalar factor, replaces the expert activation with NormSiLU to remove the surging activation ratio and vanishing SiLU magnitudes, pairs it with non-gated MLP experts, and controls sparsity through an adaptive router-entropy regularization rather than an L1-norm penalty.

3 Methodology of DECO

We propose DECO, a **DE**nse-**CO**mpetitive sparse MoE architecture that closes the performance gap to dense, under matched total parameters and training tokens. Our design has three components: the router (Section 3.1), experts (Section 3.2), and adaptive sparsity regularization (Section 3.3).

3.1 Router Design

ReLU-based routing. Distinct from conventional TopK routing, DECO incorporates ReLU in the router to determine expert activation. As demonstrated in prior studies (Yao et al., 2025; Song et al., 2025b), ReLU is differentiable almost everywhere, inherently induces sparsity, and supports token-dependent activation ratios. These attributes render ReLU a robust and flexible routing function.

Learnable expert-wise router scaling. To balance the output scales of routed and shared experts, DECO applies a scaling operator to the routing scores before they are multiplied by the expert outputs. We extend the fixed scalar scaling factor of DeepSeek-V3 (Liu et al., 2024) to a learnable vectorized one. This modification accommodates the potential heterogeneity across routed experts by assigning them distinct, learnable coefficients.

Formally, given the hidden dimension d_h, the number of routed experts N_e, and the input hidden state $\mathbf{x} \in \mathbb{R}^{d_h}$, the router score $\mathbf{p}$ can be computed as follows:

$$\mathbf{p} = \boldsymbol{\alpha} \odot \mathrm{ReLU}(\mathbf{W}_{router}^T \mathbf{x}), \tag{1}$$

where $\mathbf{W}_{router} \in \mathbb{R}^{d_h \times N_e}$ and $\boldsymbol{\alpha} \in \mathbb{R}^{N_e}$ are learnable weights, and $\odot$ is element-wise multiplication.

3.2 Expert Design

Non-gated MLP experts. While gated MLP is widely considered superior to the non-gated variant (Shazeer, 2020), we observe that non-gated experts exhibit more favorable properties within the specific context of ReLU-based routing. Concretely, in a ReLU-activated MoE, non-gated experts obtain a more stable trend of activation ratio, whereas gated variants exhibit a sharply increasing trend. This inherent stability implies that a significantly lower regularization penalty is required to achieve a target sparsity threshold, thereby alleviating the negative impact on performance.

NormSiLU. We introduce NormSiLU as an enhanced activation for MoE experts, prepending a dual-stage normalization to the SiLU non-linearity within each expert. The complete pseudocode is provided in Algorithm 1 in Appendix I.

First, inter-expert mean normalization is mathematically equivalent to centering the per-expert up-projection weights around the cross-expert mean, which makes each expert's pre-activation zero-centered with respect to it. Second, intra-expert RMS normalization holds the pre-activation scale constant. We find that this dual-stage normalization not only prevents the SiLU output magnitudes inside experts from vanishing, but also promotes a steady routed-expert activation ratio. We analyze this empirically in Section 4.3 and provide a theoretical motivation in Appendix I.

Given the expert intermediate dimension d_e, the structure of a DECO expert is formally defined as:

$$\begin{aligned} \mathbf{x}_{up} &= \mathrm{SparseLinear}(\mathbf{x}, \mathbf{W}_{up}), \\ \mathbf{x}'_{up} &= \mathrm{NormSiLU}(\mathbf{x}, \mathbf{W}_{up}, \mathbf{x}_{up}), \\ \mathbf{y} &= \mathrm{SparseLinear}(\mathbf{x}'_{up}, \mathbf{W}_{down}), \end{aligned} \tag{2}$$

where $\mathbf{W}_{up} \in \mathbb{R}^{N_e \times d_e \times d_h}$ and $\mathbf{W}_{down} \in \mathbb{R}^{N_e \times d_h \times d_e}$ are the up-projection and down-projection weights, respectively. $\mathrm{SparseLinear}$ operator performs the linear projections for active experts only at inference time.

3.3 Adaptive Sparsity Regularization

To effectively control the sparsity level, we adopt an adaptive sparsity regularization, based on the router entropy loss and a dynamic scaling algorithm for the coefficient.

Router entropy is a sparsification loss applied to a normalized router score, which is calculated as:

$$\mathbf{p}_1 = |\mathbf{p}|/\mathrm{Sum}(|\mathbf{p}|), \mathcal{L}_{ent} = -\mathbf{p}_1^T \ln(\mathbf{p}_1 + \epsilon) \tag{3}$$

where $\mathbf{p}$ is the router score defined in Equation 1. The router entropy loss, $\mathcal{L}_{ent}$, is multiplied by a coefficient λ and added to the training objective. By minimizing the entropy of the normalized router score $\mathbf{p}_1$, we concentrate the distribution mass toward a smaller number of experts, and thus increase the sparsity level.

Instead of using a static coefficient, inspired by ReMoE (Wang et al., 2024b), we adaptively scale λ according to the current sparsity level. Specifically, if the current activation ratio exceeds the target, λ is scaled by a factor $\eta > 1$ for the subsequent iteration; otherwise, λ is divided by η. In this way, DECO maintains a stable activation ratio centered precisely at the desired sparsity level.

4 Experiments

4.1 Main Results

To validate the architectural design of DECO, we introduce the following baselines: Dense (i.e., a standard LLaMA-style Transformer using SwiGLU FFNs (Touvron et al., 2023)), TopP (Huang et al.,

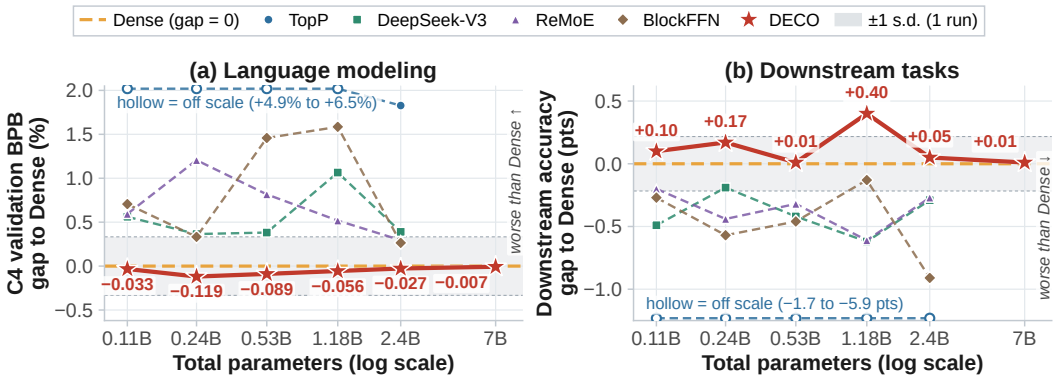


Figure 3: Performance gap to the dense baseline at six scales, on (a) language modeling and (b) downstream tasks. Each point is the gap to the dense run of the *same total-parameter budget*; we report bits-per-byte (BPB) so that the axis has the same meaning at every scale (Eq. 4, Appendix G). The grey band is one standard deviation of a single training run under seed noise (Appendix F). The 7B group includes DECO and Dense alone. DeepSeek-V3 uses gated MLP experts, and ReMoE uses non-gated ones, because these configurations outperform the alternatives; see Section 4.4.

2024), DeepSeek-V3 (Liu et al., 2024), ReMoE (Wang et al., 2024b), and BlockFFN (Song et al., 2025b). We consider six total-parameter scales: 0.11B, 0.24B, 0.53B, 1.18B, 2.4B, and 7B.

All settings are trained on the same high-quality data mixture. The model performance is evaluated by three metrics: perplexity (PPL) and bits-per-byte (BPB) on the C4 English validation set (Raffel et al., 2020), and average accuracy across a suite of commonsense reasoning benchmarks. See Appendix C for more details about experimental settings and Appendix F for repeated experiments.

To ensure a rigorous comparison, within each group of *the same total parameter count* (matched to within 0.5% of the non-embedding parameters, 1.5% at 0.53B; Appendix D), *the number of training tokens is also held consistent* at 32–41 times the parameter count. All non-FFN components, including attention and embedding layers, are identical within each group. All MoE settings in a group share the same routed-expert activation ratio (*around 20% on the training data*) and shared-expert dimension, and have close routed-expert parameter counts, so expert granularity is consistent.

Bits-per-byte. Alongside perplexity, we report bits-per-byte,

$$\mathrm{BPB} = \frac{\mathcal{L}}{\beta \ln 2} = \frac{\log_2 \mathrm{PPL}}{\beta}, \tag{4}$$

where $\mathcal{L}$ is the token-level cross-entropy and β is the average number of UTF-8 bytes per token produced by the tokenizer on the C4 validation set. BPB measures compression over the raw byte stream and does not depend on how text is segmented into tokens; we use it for every comparison spanning more than one scale, and keep PPL where a comparison stays inside one scale. Per-scale values of both are listed in Appendix G.

From the results shown in Figure 3 (raw numbers in Appendix G, per-benchmark results in Appendix J), we derive the following conclusions:

(1) *Gap elimination*: With an average routed-expert activation ratio of 20%, DECO closes the performance gap between established MoE designs and an equally-budgeted dense model. Every MoE baseline is worse than dense in *every* cell we measure. DECO lands on the better side of dense at all six scales on both metrics in single-seed runs (Figure 3). According to the five-seed study of Appendix F, a single run has a standard deviation of 0.333% BPB and 0.217 points; all six of DECO's BPB residuals and five of its six accuracy residuals fall inside that band, as do its 0.11B ($+0.21\%$ BPB) and 0.24B (-0.22% BPB) cells under five-seed means. **We claim parity with dense, not superiority over it.**

(2) *Performance superiority*: Under the same routed-expert activation ratio, shared-expert dimension and expert granularity, DECO attains both the lowest BPB and the highest task accuracy in all 20 baseline-scale cells, leading the strongest baseline by 0.29%–0.59% relative BPB, 4–18 times DECO's own residual to dense and worth 3%–7% of the parameter budget (Appendix H).

Table 1: Comparison against a dense model built from DECO's own architectural components. "Dense (DECO-style)" removes the router and the sparsity regularization from DECO while retaining learnable expert-wise scaling, NormSiLU, and non-gated MLP experts. "Dense (SwiGLU)" is the standard LLaMA-style baseline from Figure 3.

Setting	0.11B		0.24B		0.53B	
	$\text{PPL}_\downarrow$	$\text{Task}_\uparrow$	$\text{PPL}_\downarrow$	$\text{Task}_\uparrow$	$\text{PPL}_\downarrow$	$\text{Task}_\uparrow$
Dense (SwiGLU)	34.40	36.80	27.85	39.01	21.93	42.80
Dense (DECO-style)	35.34	36.64	28.13	38.64	22.42	42.26
DECO	**34.36**	**36.90**	**27.74**	**39.18**	**21.87**	**42.81**

(3) *Scale alone does not close the gap*: growing the model does not drive the baselines' gap to zero. Over 0.11B–1.18B it widens for BlockFFN ($0.71\% \rightarrow 1.59\%$), DeepSeek-V3 ($0.56\% \rightarrow 1.07\%$) and TopP ($4.95\% \rightarrow 6.46\%$), and is flat for ReMoE ($0.59\% \rightarrow 0.52\%$). The magnitude varies, the sign never does: no baseline cell reaches dense. We therefore make no claim that the gap closes on its own at larger scale, which is what motivates an architectural remedy.

(4) *MoE-specific design*: DECO also changes the FFN itself, so we additionally compare against **Dense (DECO-style)**, a dense model built from DECO's non-routing components (Table 1). It trails both Dense and DECO at the three scales tested (0.11B, 0.24B, and 0.53B), indicating that DECO's gains require sparse routing rather than a better dense FFN.

4.2 Effect of Expert-Wise Router Scaling

Table 2: Ablation on learnable expert-wise router scaling. "Fixed" uses one constant scaling factor; "Scalar" uses one shared learnable scalar factor.

	0.11B		0.24B	
	$\text{PPL}_\downarrow$	$\text{Task}_\uparrow$	$\text{PPL}_\downarrow$	$\text{Task}_\uparrow$
Fixed	34.43	36.74	27.94	39.08
Scalar	35.16	36.63	27.92	38.54
DECO	**34.36**	**36.90**	**27.74**	**39.18**

Table 3: Ablation on NormSiLU. "w/o" removes one normalization step; "SiLU" is the vanilla operator without normalization.

	0.11B		0.24B	
	$\text{PPL}_\downarrow$	$\text{Task}_\uparrow$	$\text{PPL}_\downarrow$	$\text{Task}_\uparrow$
w/o Mean	34.57	36.85	27.77	38.88
w/o RMS	35.77	36.78	27.97	39.04
SiLU	35.69	36.46	28.05	38.85
NormSiLU	**34.36**	**36.90**	**27.74**	**39.18**

To demonstrate the effect of DECO's router scaling design, we experiment on two ablation settings: "*Fixed*" adopts a constant scaling factor for all routed experts, and "*Scalar*" involves a single learnable scalar scaling factor shared by experts. Both ablation settings are initialized with the same values as DECO. Evaluation results in Table 2 reveal the performance benefits of learnable vectorized scaling factors. We further explore the impact of initialization values in Appendix E.

As empirical justification for this design, we analyze expert output norms on the C4 validation set and find clear heterogeneity across the output scales of routed experts. For example, in the first MoE layer of DECO (0.24B), the average output norm of all experts is 0.23, while the maximum reaches 1.14, roughly five times the mean. This validates our hypothesis that expert-specific learnable factors are essential to accommodate the varying output scale across experts.

4.3 Effect of NormSiLU

To validate the effect of NormSiLU, which incorporates both inter-expert mean normalization and intra-expert RMS normalization, we evaluate three ablation settings: "*w/o Mean*" removes inter-expert mean normalization, "*w/o RMS*" removes intra-expert RMS normalization, and "*SiLU*" is the standard SiLU operator without any normalization.

As shown in Table 3, both normalization steps contribute positively, with intra-expert RMS normalization providing the larger gain. To investigate the mechanisms of NormSiLU, we track three variables throughout training: the routed-expert activation ratio, the sparsity regularization coefficient, and the average absolute SiLU output magnitude within experts.

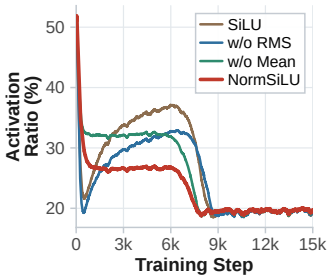


Figure 4: The trend of activation ratio of DECO (0.11B) and ablation settings each removing one step of NormSiLU. The baseline "SiLU" and "w/o RMS" settings show *surging routed-expert activation ratio* before being pulled back by regularization.

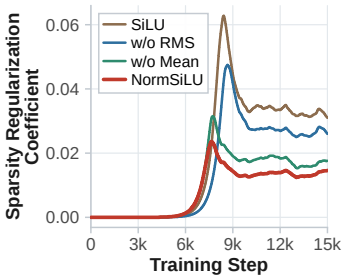


Figure 5: The trend of the regularization coefficient of DECO (0.11B) and ablation settings each removing one step of NormSiLU. The settings "SiLU" and "w/o RMS" show significantly higher coefficients, which potentially harm performance.

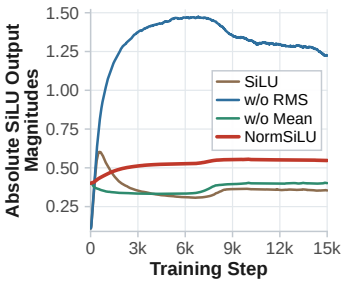


Figure 6: The average absolute output magnitudes of SiLU within routed experts of DECO (0.11B) and ablation settings each removing one step of NormSiLU. The settings "SiLU" and "w/o Mean" show *vanishing SiLU output magnitudes*.

As illustrated in Figure 4, the activation ratios of "SiLU" and "w/o RMS" surge rapidly during the initial training phase. While the adaptive regularization pulls back this surge, Figure 5 reveals that these settings require a significantly higher regularization coefficient, which typically degrades overall performance. Conversely, NormSiLU and "w/o Mean" maintain stable activation trends, with NormSiLU maintaining the lowest activation ratio before regularization takes effect. We conclude that **intra-expert RMS normalization mitigates the uncontrolled growth of activation ratios, while inter-expert mean normalization further promotes sparsity level**.

Moreover, analysis of the internal SiLU output magnitudes (Figure 6) reveals that "SiLU" and "w/o Mean" exhibit considerably lower magnitudes. This suggests that expert neurons (i.e., parameter columns/rows) in these settings are potentially under-utilized and less significantly activated. In contrast, **inter-expert mean normalization effectively addresses this issue, ensuring more robust activation and utilization of expert neurons**.

4.4 Effect of Expert Gating

Table 4: The ablation results on the effect of expert gating. "GA" and "NG" indicate gated MLP experts and non-gated MLP experts, respectively. "DS-V3" indicates DeepSeek-V3.

Setting	0.11B		0.24B		0.53B	
	PPL$_\downarrow$	Task$_\uparrow$	PPL$_\downarrow$	Task$_\uparrow$	PPL$_\downarrow$	Task$_\uparrow$
DS-V3 (GA)	**35.09**	36.31	**28.19**	**38.82**	**22.19**	**42.38**
DS-V3 (NG)	35.16	**36.73**	28.28	38.47	22.33	42.02
ReMoE (GA)	36.02	**36.61**	29.55	38.15	22.78	42.27
ReMoE (NG)	**35.13**	36.60	**28.99**	**38.57**	**22.49**	**42.48**
DECO (GA)	39.77	35.78	32.46	37.29	22.13	42.29
DECO (NG)	**34.36**	**36.90**	**27.74**	**39.18**	**21.87**	**42.81**

To investigate whether expert gating significantly influences MoE performance, we conduct experiments on three architectures: DeepSeek-V3, ReMoE, and DECO. DeepSeek-V3 is a well-performing MoE architecture using TopK routing, while ReMoE and DECO use ReLU-based routing to implement a flexible activation ratio. For each architecture, we compare non-gated MLP experts (NG) against gated MLP experts (GA).

As demonstrated by Table 4, **for ReLU-based routing, non-gated MLP experts generally surpass gated counterparts**. Conversely, for standard TopK routing (e.g., DeepSeek-V3), gated experts provide a marginal performance advantage.

We attribute this disparity to the training dynamics that emerge when gated experts are paired with flexible, threshold-based routing. As illustrated in Figure 7, DECO (GA) exhibits highly unstable

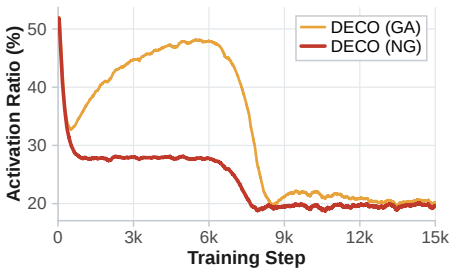


Figure 7: The trend of activation ratio of DECO (0.11B) using different expert gating policies.

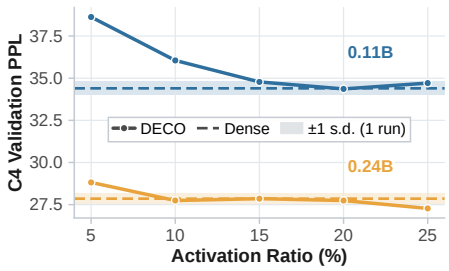


Figure 8: The impact of the routed-expert activation ratio on the performance of DECO.

activation trends, characterized by a drastic surge in the routed-expert activation ratio that must be aggressively counteracted by sparsity regularization. In contrast, DECO (NG) maintains a stable activation trajectory, requiring substantially less regularization and thereby preserving performance.

Mechanistically, gated experts (e.g., SwiGLU) contain more multiplicative interactions that produce highly dynamic output scales, propagating larger gradients to the router. Since ReLU-based routing couples the activation decision directly to the sign of the router score, this destabilizes the activation ratio. TopK routing masks the same instability by fixing the number of active experts per token, and a dense FFN has no activation decision to destabilize at all — consistent with Dense (DECO-style) underperforming the SwiGLU dense baseline (Table 1). Expert gating is therefore not harmful in itself; it becomes destabilizing only when the activation decision is tied directly to the router score.

4.5 Effect of Activation Ratio

We evaluate DECO across a wide range of routed-expert activation ratios, from 5% to 25%. The experimental results, illustrated in Figure 8, yield two primary observations:

(1) Positive correlation with performance: DECO improves steeply at low activation ratios and saturates once enough experts are active. This aligns with standard TopK MoE, where a higher activation ratio means more per-token computation and hence better performance, with diminishing returns if the number of active experts continues to increase after saturation.

(2) Scale-dependent activation thresholds: The activation ratio at which DECO reaches dense-level performance varies by scale: the 0.11B setting matches the dense baseline at 20%, or at 15% if we only require it to enter the dense run's seed-noise band, whereas 0.24B enters it at 10%. This suggests DECO may close the gap at lower activation ratios as it scales up, consistent with recent literature reporting that the optimal activation ratio decreases with total parameter count (Zhao et al., 2025a). Further investigation on larger-scale models is required to validate this scaling trend.

Besides activation ratio, we also study the effect of other MoE-specific factors, including expert granularity and shared expert size; see Appendix B.

5 Practical Inference Acceleration

Table 5: Single-GPU decoding speeds on Spec-Bench (token/sec) and average speedup ratios of our acceleration kernel (denoted as "DECO") relative to "Dense" on different devices.

Device	Setting	MT.	Trans.	Summ.	QA	Math	RAG	Average
RTX 4090 24GB	Dense	86.79	88.01	86.38	88.08	87.98	86.42	**87.28**
	TopK	194.12	203.75	193.27	196.70	195.97	198.12	**196.99**
	DECO	223.38	221.76	217.34	227.56	230.94	221.49	**223.75**
	Speedup	2.57×	2.52×	2.52×	2.58×	2.62×	2.56×	**2.56**×
Jetson AGX Orin 64GB	Dense	15.00	15.21	14.93	15.24	15.22	14.94	**15.09**
	TopK	39.28	41.70	38.94	40.03	40.12	40.16	**40.04**
	DECO	44.05	43.99	42.69	45.21	45.92	43.57	**44.24**
	Speedup	2.94×	2.89×	2.86×	2.97×	3.02×	2.92×	**2.93**×

We implement an acceleration kernel that exploits sparse activation to cut the computation and memory traffic of inactive routed experts, and evaluate it on Spec-Bench (Xia et al., 2024) using a single NVIDIA RTX 4090 (24GB) and a Jetson AGX Orin (64GB). We compare against two baselines: **Dense**, standard autoregressive decoding without sparsity optimizations, and **TopK**, which replaces our ReLU-based routing with conventional TopK. All three are implemented within FR-Spec (Zhao et al., 2025b), a CUDA-optimized inference framework, and run the same weights, precision, and batch size, differing only in how experts are processed: Dense reads and computes every expert, while TopK and our kernel share an identical average activation ratio of 20%. This benchmark uses a separate deployment-oriented DECO configuration whose routed experts hold about 90% of the non-embedding parameters, so the 20% ratio removes roughly 70% of the weights read per token.

As detailed in Table 5, our acceleration kernel substantially outperforms the Dense baseline, achieving an average speedup of 2.56× on the RTX 4090 and 2.93× on the Jetson AGX Orin. These results demonstrate that the sparse activation patterns induced by DECO effectively translate into tangible throughput improvements on real edge hardware. Furthermore, the decoding speed of our kernel is about 1.10×–1.14× the speed of a standard TopK under identical sparsity levels. This additional advantage stems from the element-wise, in-place execution of the ReLU, which incurs a lower computational complexity than the partial sort over expert scores required by TopK.

6 Discussion

Why should a sparse MoE reach dense-level performance under an equal budget? The property of activation sparsity, indicating that only a small fraction of parameters make non-negligible contributions to final outputs, also exists in dense models. Recent studies (Song et al., 2025a; Luo et al., 2024; Zhang et al., 2024) indicate that for each token, only about 30%–40% of neurons within a standard SwiGLU FFN provide non-negligible contributions, so dense models already tolerate a large fraction of inactive computation. A sparse MoE that activates its experts efficiently can in principle reach dense-level performance at a reduced activation ratio, provided the sparsity is not bought with blocked gradients or an excessive regularization cost.

Closing the gap may be easier under more heterogeneous training data. In our main experiments, we adopt a diverse data mixture for training, including web texts, code, math, and many other data categories. On such heterogeneous data, DECO matches or exceeds the dense baselines across various parameter scales (Figure 3). However, whether the gap fully closes may be sensitive to the underlying data composition. When trained on FineWeb (Penedo et al., 2024), a less heterogeneous dataset, DECO (0.11B) does not reach parity in language modeling: 30.14 vs. 29.76 PPL for Dense (0.11B), a residual of $+0.37\%$ relative BPB, just outside our seed-noise band, though it still exceeds Dense in task accuracy (38.65 vs. 38.03). A likely explanation is that high-entropy, multi-domain data suits sparse MoE, where domain-specific inputs can be handled by activating only a specialized subset of experts. More studies are needed to verify this.

Scope of our claims. Our results are established under a strict protocol: matched total parameters (Appendix D), matched training tokens, and a scale range from 0.11B to 7B parameters, which covers the regime relevant to edge-device deployment. Within this regime, at a 20% activation ratio, DECO reaches dense-level performance and outperforms every MoE baseline. We do not claim that the gap stays closed at frontier scale, under substantially longer token horizons, or after supervised fine-tuning and reinforcement learning. Dense-level performance is also conditional on hyperparameters, requiring sufficient expert granularity and a shared expert comparable in size to a routed expert (Appendix B). See Appendix A for more potential limitations of our work.

7 Conclusion

We propose DECO, a **DE**nse-**CO**mpetitive MoE architecture that closes the performance gap between existing MoE designs and a dense model at equal total parameters and training tokens. We argue that the gap stems from how existing designs enforce sparsity, by hard TopK selection or by regularization, and address it with three changes: ReLU-based routing with learnable expert-wise scaling, non-gated MLP experts, and the NormSiLU activation. At a 20% activation ratio, DECO reaches dense-level performance at every scale we measure, surpasses all MoE baselines under identical settings, and delivers measured speedups on Jetson AGX Orin, a representative edge device.

AI USE STATEMENT

In this work, we used generative AI tools for the following tasks that require disclosure: (i) assisting with the writing of the derivation in Appendix I and with formulating the mathematical claim discussed there; (ii) providing feedback on research methodology and experimental design; (iii) assisting with method implementation; and (iv) assisting with the interpretation of experimental results. We did not use generative AI tools to generate synthetic datasets, translate text, clean or reformat datasets, develop the theoretical model or conceptual framework underlying DECO, provide critical ingredients needed to establish the mathematical claim, propose or refine the research hypotheses, or conduct qualitative or thematic data analysis. The DECO architecture was conceived by the authors, and no reported numerical result was generated or altered by generative AI. Additionally, we used generative AI tools for tasks for which disclosure is recommended: creating figures, editing experimental code, refining the wording of this paper to improve readability, and identifying potentially relevant literature. We reviewed all AI-assisted outputs before including them in this work. LLM-generated code was verified and tested for correctness by two authors, and the AI-assisted derivation and mathematical claim were manually checked for correctness. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work presents DECO, a sparse MoE architecture optimized for edge-device deployment. By significantly reducing the active parameter scale and memory-access overhead without compromising representation power, our architecture promotes more energy-efficient deployment of LLMs. This efficiency translates to a reduced carbon footprint during inference and helps democratize advanced AI capabilities by making them accessible on resource-constrained edge hardware.

While our architectural improvements are fundamentally algorithmic, we also rely on large-scale web corpora (e.g., FineWeb, the Pile) and will inevitably inherit the historical biases, toxicity, and representational harms present in the pre-training data. We have conducted cleaning and detoxification on the data and used a rule-based program to remove personal information and offensive content. We encourage researchers and practitioners utilizing our architecture to apply rigorous safety alignment and bias mitigation protocols before releasing downstream models.

REPRODUCIBILITY STATEMENT

All checkpoints for the six scales reported in Section 4, together with the training code, will be released. The training details and hyperparameters are presented in Appendix C. The evaluation results can be reproduced by evaluating our checkpoints through LMEval (Gao et al., 2024).

REFERENCES

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. URL https://arxiv.org/pdf/1607.06450.

Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, et al. Deepseek LLM: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*, 2024. URL https://arxiv.org/pdf/2401.02954.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7432–7439, 2020. URL https://ojs.aaai.org/index.php/AAAI/article/view/6239/6095.

Weilin Cai, Juyong Jiang, Fan Wang, Jing Tang, Sunghun Kim, and Jiayi Huang. A survey on mixture of experts in large language models. *IEEE Transactions on Knowledge and Data Engineering*, 2025. URL https://arxiv.org/pdf/2507.11181.

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. *arXiv preprint arXiv:1803.05457*, 2018. URL https://arxiv.org/pdf/1803.05457.pdf.

Damai Dai, Chengqi Deng, Chenggang Zhao, RX Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Yu Wu, et al. DeepSeekMoE: Towards ultimate expert specialization in mixture-of-experts language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1280–1297, 2024. URL https://aclanthology.org/2024.acl-long.70.pdf.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. URL https://arxiv.org/pdf/2407.21783.

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. URL https://www.jmlr.org/papers/volume23/21-0998/21-0998.pdf.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The Pile: An 800GB dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020. URL https://arxiv.org/pdf/2101.00027.pdf.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, et al. MiniCPM: Unveiling the potential of small language models with scalable training strategies. *arXiv preprint arXiv:2404.06395*, 2024.

Quzhe Huang, Zhenwei An, Nan Zhuang, Mingxu Tao, Chen Zhang, Yang Jin, Kun Xu, Liwei Chen, Songfang Huang, and Yansong Feng. Harder tasks need more experts: Dynamic routing in MoE models. *arXiv preprint arXiv:2403.07652*, 2024. URL https://arxiv.org/pdf/2403.07652.

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. URL https://arxiv.org/pdf/2401.04088.

Peng Jin, Bo Zhu, Li Yuan, and Shuicheng Yan. MoE++: Accelerating mixture-of-experts methods with zero-computation experts. *arXiv preprint arXiv:2410.07348*, 2024. URL https://arxiv.org/pdf/2410.07348?

Jakub Krajewski, Jan Ludziejewski, Kamil Adamczewski, Maciej Pióro, Michał Krutul, Szymon Antoniak, Kamil Ciebiera, Krystian Król, Tomasz Odrzygóźdź, Piotr Sankowski, et al. Scaling laws for fine-grained mixture of experts. *arXiv preprint arXiv:2402.07871*, 2024. URL https://arxiv.org/pdf/2402.07871.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023. URL https://dl.acm.org/doi/pdf/10.1145/3600006.3613165.

Houyi Li, Ka Man Lo, Ziqi Wang, Zili Wang, Wenzhen Zheng, Shuigeng Zhou, Xiangyu Zhang, and Daxin Jiang. Can mixture-of-experts surpass dense llms under strictly equal resources? *arXiv preprint arXiv:2506.12119*, 2025. URL https://arxiv.org/pdf/2506.12119.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2024. URL https://arxiv.org/pdf/2412.19437.

Yuqi Luo, Chenyang Song, Xu Han, Yingfa Chen, Chaojun Xiao, Zhiyuan Liu, and Maosong Sun. Sparsing Law: Towards large language models with greater activation sparsity. *arXiv preprint arXiv:2411.02335*, 2024. URL https://arxiv.org/pdf/2411.02335.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc-Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. 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. URL https://aclanthology.org/P16-1144.pdf.

Guilherme Penedo, Hynek Kydlíček, Anton Lozhkov, Margaret Mitchell, Colin A Raffel, Leandro Von Werra, Thomas Wolf, et al. The FineWeb datasets: Decanting the web for the finest text data at scale. *Advances in Neural Information Processing Systems*, 37:30811–30849, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/370df50ccfdf8bde18f8f9c2d9151bda-Paper-Datasets_and_Benchmarks_Track.pdf.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text Transformer. *Journal of machine learning research*, 21(140):1–67, 2020. URL https://www.jmlr.org/papers/volume21/20-074/20-074.pdf.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 8732–8740, 2020. URL https://cdn.aaai.org/ojs/6399/6399-13-9624-1-10-20200517.pdf.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. SocialIQA: Commonsense reasoning about social interactions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 4463–4473, 2019. URL https://aclanthology.org/D19-1454.pdf.

Noam Shazeer. GLU variants improve Transformer. *arXiv preprint arXiv:2002.05202*, 2020. URL https://arxiv.org/pdf/2002.05202.pdf.

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*, 2017. URL https://openreview.net/pdf?id=B1ckMDqlg.

Chenyang Song, Xu Han, Zhengyan Zhang, Shengding Hu, Xiyu Shi, Kuai Li, Chen Chen, Zhiyuan Liu, Guangli Li, Tao Yang, and Maosong Sun. ProSparse: Introducing and enhancing intrinsic activation sparsity within large language models. In *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 2626–2644, January 2025a. URL https://aclanthology.org/2025.coling-main.180.pdf.

Chenyang Song, Weilin Zhao, Xu Han, Chaojun Xiao, Yingfa Chen, Yuxuan Li, Zhiyuan Liu, and Maosong Sun. BlockFFN: Towards end-side acceleration-friendly mixture-of-experts with chunk-level activation sparsity. *arXiv preprint arXiv:2507.08771*, 2025b. URL https://arxiv.org/pdf/2507.08771.

Dan Su, Kezhi Kong, Ying Lin, Joseph Jennings, Brandon Norick, Markus Kliegl, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. Nemotron-CC: Transforming common crawl into a refined long-horizon pretraining dataset. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2459–2475, 2025. URL https://aclanthology.org/2025.acl-long.123.pdf.

Vijay Thakkar, Pradeep Ramani, Cris Cecka, Aniket Shivam, Honghao Lu, Ethan Yan, Jack Kosaian, Mark Hoemmen, Haicheng Wu, Andrew Kerr, Matt Nicely, Duane Merrill, Dustyn Blasig, Fengqi Qiao, Piotr Majcher, Paul Springer, Markus Hohnerbach, Jin Wang, and Manish Gupta. CUTLASS, Jan 2023. URL https://github.com/NVIDIA/cutlass.

Changxin Tian, Kunlong Chen, Jia Liu, Ziqi Liu, Zhiqiang Zhang, and Jun Zhou. Towards greater leverage: Scaling laws for efficient mixture-of-experts language models. *arXiv preprint arXiv:2507.17702*, 2025. URL https://arxiv.org/pdf/2507.17702.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. LLaMA 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023. URL https://arxiv.org/pdf/2307.09288.pdf.

Lean Wang, Huazuo Gao, Chenggang Zhao, Xu Sun, and Damai Dai. Auxiliary-loss-free load balancing strategy for mixture-of-experts. *arXiv preprint arXiv:2408.15664*, 2024a. URL https://arxiv.org/pdf/2408.15664.

Ziteng Wang, Jianfei Chen, and Jun Zhu. ReMoE: Fully differentiable mixture-of-experts with ReLU routing. *arXiv preprint arXiv:2412.14711*, 2024b. URL https://arxiv.org/pdf/2412.14711.

Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. In *Findings of the Association for Computational Linguistics ACL 2024*, pp. 7655–7671, 2024. URL https://aclanthology.org/2024.findings-acl.456.pdf.

Feng Yao, Junxia Cui, Ruohan Zhang, Liyuan Liu, Shibo Hao, Li Zhang, Chengyu Dong, Shuohang Wang, Jianfeng Gao, Jingbo Shang, et al. DenseMixer: Improving moe post-training with precise router gradient. In *NeurIPS 2025 Workshop on Structured Probabilistic Inference $\{\backslash\&\}$ Generative Modeling*, 2025. URL https://openreview.net/pdf?id=88PI0SpVL3.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 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. URL https://aclanthology.org/P19-1472.pdf.

Zhengyan Zhang, Yixin Song, Guanghui Yu, Xu Han, Yankai Lin, Chaojun Xiao, Chenyang Song, Zhiyuan Liu, Zeyu Mi, and Maosong Sun. ReLU^2 wins: Discovering efficient activation functions for sparse LLMs. *arXiv preprint arXiv:2402.03804*, 2024. URL https://arxiv.org/pdf/2402.03804.pdf.

Guoliang Zhao, Yuhan Fu, Shuaipeng Li, Xingwu Sun, Ruobing Xie, An Wang, Weidong Han, Zhen Yang, Weixuan Sun, Yudong Zhang, et al. Towards a comprehensive scaling law of mixture-of-experts. *arXiv preprint arXiv:2509.23678*, 2025a. URL https://arxiv.org/pdf/2509.23678.

Weilin Zhao, Tengyu Pan, Xu Han, Yudi Zhang, Ao Sun, Yuxiang Huang, Kaihuo Zhang, Weilun Zhao, Yuxuan Li, Jianyong Wang, et al. FR-Spec: Accelerating large-vocabulary language models via frequency-ranked speculative sampling. *arXiv preprint arXiv:2502.14856*, 2025b. URL https://arxiv.org/pdf/2502.14856.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, et al. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025. URL https://arxiv.org/pdf/2507.18071.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody H Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. SGLang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/724be4472168f31ba1c9ac630f15dec8-Paper-Conference.pdf.

A LIMITATIONS

Our evaluation covers pre-training quality and few-shot task performance. We do not conduct supervised fine-tuning (SFT) or reinforcement learning (RL). Post-training remains an open question for MoE architectures in general: prior work reports RL instability caused by fluctuating router activations (Zheng et al., 2025). Whether ReLU-based routing behaves the same way is untested. We are extending DECO to post-training at edge-deployable scales, and identifying and mitigating any such issues is the main objective of that work.

We measure our kernel inside FR-Spec (Zhao et al., 2025b) under single-request, single-device decoding, the regime an edge device operates in. A mainstream serving stack such as vLLM (Kwon et al., 2023) or SGLang (Zheng et al., 2024) behaves differently: continuous batching may amortize the weight read that our sparsity removes. Our speedups are therefore claims about edge decoding rather than server-side throughput. Porting the kernel to a batched serving runtime, and locating the crossover in batch size, is left to future work.

Several observations in this work invite verification at a wider scope. Within the 0.11B–7B range we study, the baselines' gap to dense does not close on its own, and DECO reaches parity at every scale. Whether either behavior persists beyond that range remains to be determined. Two further questions are open: how the activation ratio threshold required for dense-level performance scales with model size, and how DECO's intrinsic sparsity and performance vary across data distributions.

DECO removes the performance gap, so the gap is not an inherent cost of sparsity. Heavy sparsity regularization is one mechanism by which existing designs incur it, and we establish this by controlled comparison, both within DECO (Section 4.3, Section 4.4) and on a baseline architecture (Table 6). The TopK-routed baselines in Figure 3 differ from DECO in more than their routing rule, so our case for hard TopK selection is mechanistic rather than experimental.

B EFFECT OF KEY MOE HYPERPARAMETERS

Compared with dense architectures, MoE models introduce unique hyperparameters that significantly influence performance, among which the activation ratio, expert granularity, and shared expert size are often considered the most important ones (Tian et al., 2025). Similarly, the performance of DECO is also sensitive to these factors, and the gap to dense closes only under certain conditions. We have already studied the effect of the routed-expert activation ratio in Section 4.5, which has a significant impact on MoE performance. In this section, we characterize the impact of the remaining two MoE-specific factors.

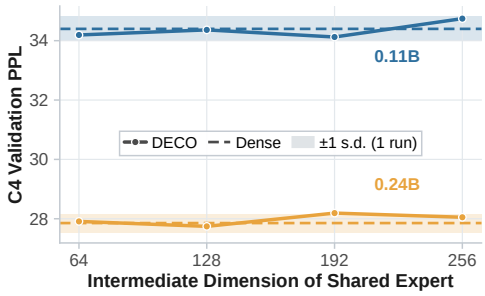


Figure 9: The impact of shared expert sizes on performance. Routed expert dimension is 64.

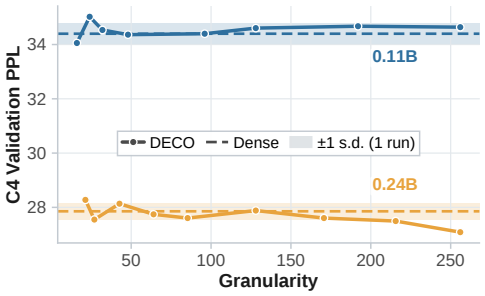


Figure 10: The impact of the expert granularity ($g = 4d_h/d_e$) on performance of DECO.

B.1 SHARED EXPERT SIZE

We evaluate the performance of DECO across various shared-expert sizes, expressed as a multiple of the routed-expert intermediate dimension d_e. For rigorous comparison, we hold the total parameter count approximately constant. This necessitates a trade-off between the capacity of the shared expert and the number of routed experts.

As illustrated in Figure 9, when the intermediate dimension of the shared expert is around 1–2 times that of a routed expert (fixed at 64), DECO reaches dense-level performance and exhibits relative insensitivity to the size of the shared expert. Beyond 2 times, the curves become uneven: the 0.24B setting leaves the seed-noise band of its dense baseline at 3 times, and at 4 times both settings sit on the worse side of dense, though still within that band. We attribute this to the reduced number of routed experts necessitated by the fixed parameter budget. This corroborates Tian et al. (2025): an oversized shared expert is unnecessary, and a single shared expert with a scale comparable to that of a routed expert appears to suffice.

B.2 Expert Granularity

Figure 10 reports the C4 validation perplexity of DECO as the expert granularity $g = 4d_h/d_e$ varies under a fixed total parameter budget. We make two observations.

First, **granularity has a considerably weaker effect than the activation ratio**. For the 0.11B setting, perplexity varies by only about 0.3 across $g \in [32, 256]$, less than half of its seed-noise band. The 0.24B setting is more responsive to granularity: its perplexity falls by 0.66, from 27.74 at $g = 64$ to 27.08 at $g = 256$. We attribute this asymmetry to expert capacity: the routed experts of the 0.11B setting are likely too low-capacity to develop the fine-grained specialization that finer granularity is intended to enable, or to capture the "long-tail" of the data distribution, whereas the 0.24B setting still benefits from it.

Second, DECO is never worse than its dense baseline by more than seed noise at any $g \geq 32$, which covers 13 of the 17 points plotted; the only two points that fall short of dense beyond the band are the coarsest ones, $g = 24$ at 0.11B and $g = 21$ at 0.24B. All six reported settings sit well inside this range. Keeping g at the value dictated by parameter matching is a protocol constraint rather than a tuned optimum, and at 0.24B a finer g would have helped, as noted above. Overall, expert granularity is a second-order factor for DECO once $g \gtrsim 32$, that is, once individual routed experts are kept small relative to the shared expert.

C Detailed Experimental Settings

Training datasets For all settings, we use the same pretraining dataset, which consists of a mixture of various data, including FineWeb (Penedo et al., 2024), Nemotron-CC (Su et al., 2025), Pile (Gao et al., 2020), Wikipedia, and many other collected raw corpora. The training data covers a wide range of categories, such as raw web text, math, and code. The data mixing ratio is carefully tuned through experiments on small-scale models.

Evaluation benchmarks We evaluate the accuracy of models on various benchmarks with LMEval (Gao et al., 2024), including PIQA (Bisk et al., 2020), SIQA (Sap et al., 2019), HellaSwag (Zellers et al., 2019), ARC-C, ARC-E (Clark et al., 2018), WinoGrande (Sakaguchi et al., 2020), and LAMBADA (Paperno et al., 2016). The evaluation parameters follow the default dataset configurations provided by LMEval. All dataset usage is restricted to academic intents, which does not violate the terms of use and is consistent with their intended use. The languages covered mainly include English and Chinese.

Baseline adjustments Since BlockFFN is originally designed to combine activation sparsity and speculative decoding for better acceleration, it introduces the chunk sparsification loss and activation locality loss to increase the union sparsity level of consecutive tokens as well as the activation similarity between neighboring tokens (Song et al., 2025b). However, as we do not target these objectives in this work, for fair comparison, we replace their original regularization with our own adaptive sparsity regularization.

To justify such an adjustment, we implement BlockFFN strictly using its original "two-loss" setting, tuning the loss coefficients via grid search. From the results in Table 6, at the same target activation ratio of 20%, our "one-loss" setting is consistently better than the original "two-loss" regularization, supporting our experimental setting for BlockFFN.

Table 6: The performance of BlockFFN using its original regularization (two-loss) and our own adaptive sparsity regularization (one-loss).

Setting	$\text{PPL}_{\downarrow}$	$\text{Task}_{\uparrow}$
0.11B (two-loss)	35.62	36.23
0.11B (one-loss)	**35.27**	**36.53**
0.24B (two-loss)	28.65	38.18
0.24B (one-loss)	**28.16**	**38.44**

Table 7: The hyperparameter settings of models. d_h, d_e, d_s, N_e, d_{ff}, N_{layer}, n_{step}, and n_{tokens} denote the hidden size, the intermediate dimension of one routed expert, the intermediate dimension of the shared expert, the number of routed experts, the intermediate dimension of the dense FFN layer, the total number of hidden layers, the number of training steps, and the number of training tokens, respectively. η and λ_{init} are the coefficient multiplier and the initial value of the regularization coefficient as used in the adaptive sparsity regularization. Within each scale, all architectures share the tokenizer, the data mixture, the token budget, and the schedule. The listed N_e and d_e are those of DECO, and follow $N_e = 3(d_{ff} - d_s)/(2d_e)$ so that the expert block matches the dense FFN stack it replaces; MoE baselines use a different (N_e, d_e) so as to keep their parameter count matched, and gated-expert variants use a smaller d_e to pay for the third matrix. Appendix D lists the geometry and the measured parameter count of every setting.

Scale	d_h	d_e	d_s	N_e	d_{ff}	N_{layer}	n_{step}	lr	Batch (tokens)	n_{tokens}	η	λ_{init}
0.11B	768	64	128	42	1,920	16	15,000	1.175×10^{-3}	2.62×10^5	3.93 B	1.002	1×10^{-8}
0.24B	1,024	64	128	57	2,560	20	15,000	9.3×10^{-4}	5.24×10^5	7.86 B	1.002	1×10^{-8}
0.53B	1,280	64	128	77	3,360	27	20,000	7.8×10^{-4}	1.05×10^6	21.0 B	1.001	1×10^{-8}
1.18B	1,792	64	128	102	4,480	32	23,000	6.54×10^{-4}	2.10×10^6	48.3 B	1.001	1×10^{-8}
2.4B	2,560	64	128	147	6,400	32	23,000	5.59×10^{-4}	4.19×10^6	96.5 B	1.001	1×10^{-8}
7B	3,584	64	128	207	8,960	47	24,000	4.407×10^{-4}	1.15×10^7	277 B	1.001	1×10^{-8}

Hyperparameters To find the optimal hyperparameters, we first conduct a grid search on small-scale models. Then, following DeepSeek LLM (Bi et al., 2024), we assume a power-law relationship of both the optimal learning rate and batch size with respect to the computation budget. Thereby, we can extrapolate the hyperparameters for large-scale models using those for small-scale models. The detailed experimental hyperparameters are shown in Table 7. All MoE baseline settings and DECO use a dense FFN at the first layer, and adopt sparse MoE for the remaining $N_{layer} - 1$ layers. We use the WSD learning rate scheduler along the training process (Hu et al., 2024; Dubey et al., 2024), with 100 warmup steps at the beginning and 1,000 decay steps at the last stage. Our models are trained on 8–128 NVIDIA A800 GPUs for 2–168 hours, depending on the model scale. The measured parameter count of every checkpoint, broken down by component, is reported in Appendix D.

D Parameter Accounting

Our central claim is comparison at a matched total parameter budget, so we report the exact parameter count of every trained checkpoint. All counts in this section are measured directly from the model weights evaluated in Figure 3; "non-embedding" excludes the input and output embeddings.

How the budget is matched. Within a scale, all architectures share the tokenizer, the embeddings, the attention stack, the LayerNorms, and the first-layer dense FFN; they differ only in the $N_{layer} - 1$ MoE blocks. The dense FFN of one layer holds $3d_h d_{ff}$ parameters (gated MLP). A DECO MoE block holds $2N_e d_e d_h$ parameters in its non-gated routed experts, $3d_h d_s$ in its gated shared expert, $N_e(d_h + 1)$ in the router ($\mathbf{W}_{router}$ and $\boldsymbol{\alpha}$), and d_e in the NormSiLU RMSNorm. Equating the expert capacity to the dense FFN it replaces, $3d_h d_{ff} = 3d_h d_s + 2N_e d_e d_h$, we have:

$$N_e = \frac{3\,(d_{ff} - d_s)}{2\,d_e}. \tag{5}$$

With $d_s = 128$ and $d_e = 64$ held fixed at every scale, Eq. 5 returns exactly 42, 57, 102, 147 and 207 for the 0.11B, 0.24B, 1.18B, 2.4B and 7B settings, which are the values we use (Table 7). At those

Table 8: Exact parameter decomposition of the Dense and DECO runs at all six scales. Components above the first rule are byte-identical between the two architectures. The "expert block" of DECO (first-layer dense FFN + routed experts + shared expert) matches the dense FFN stack exactly at five of six scales; the 0.53B excess of $5{,}324{,}800$ is the non-integral-N_e effect discussed in the text. The remaining difference is the router and the NormSiLU RMSNorm, which have no dense counterpart.

Component	0.11B	0.24B	0.53B	1.18B	2.4B	7B
Embedding (input + output, untied)	112,816,128	150,421,504	188,026,880	263,237,632	668,467,200	935,854,080
Attention	37,748,736	83,886,080	176,947,200	411,041,792	838,860,800	2,414,870,528
LayerNorm	25,344	41,984	70,400	116,480	166,400	340,480
Dense						
FFN, all N_{layer} layers	70,778,880	157,286,400	348,364,800	770,703,360	1,572,864,000	4,527,882,240
DECO						
First-layer dense FFN	4,423,680	7,864,320	12,902,400	24,084,480	49,152,000	96,337,920
Routed experts $\mathbf{W}_{up}, \mathbf{W}_{down}$	61,931,520	141,950,976	328,007,680	725,286,912	1,493,237,760	4,368,236,544
Shared expert	4,423,680	7,471,104	12,779,520	21,331,968	30,474,240	63,307,776
Router $\mathbf{W}_{router}$, $\boldsymbol{\alpha}$	484,470	1,110,075	2,564,562	5,669,466	11,670,477	34,136,370
NormSiLU RMSNorm	960	1,216	1,664	1,984	1,984	2,944
Expert block, DECO − Dense	0	0	5,324,800	0	0	0
Non-embedding total, Dense	108,552,960	241,214,464	525,382,400	1,181,861,632	2,411,891,200	6,943,093,248
Non-embedding total, DECO	109,038,390	242,325,755	533,273,426	1,187,533,082	2,423,563,661	6,977,232,562
Relative difference	+0.447%	+0.461%	+1.502%	+0.480%	+0.484%	+0.492%
DECO activated, non-embedding	59,493,174	128,764,974	270,867,282	607,303,552	1,228,973,453	3,482,643,327
as a fraction of Dense	54.8%	53.4%	51.6%	51.4%	51.0%	50.2%

five scales the DECO expert block therefore matches the dense FFN stack to the parameter, and the entire residual difference is the router plus the NormSiLU RMSNorm — two components a dense model does not have at all. Table 8 gives the full decomposition.

The 0.53B exception. At the 0.53B scale Eq. 5 evaluates to $3(3360 - 128)/128 = 75.75$, which is not an integer, so the expert block cannot be matched exactly while keeping $d_e = 64$. Our runs use $N_e = 77$, which overshoots by $(77 - 75.75) \cdot 2d_e d_h = 204{,}800$ parameters per MoE layer and $5{,}324{,}800$ in total, i.e. $+1.01\%$ of the non-embedding budget; DECO's total excess at 0.53B is therefore $+1.50\%$ instead of the $+0.48\%$ seen elsewhere. We report the 0.53B numbers as measured and flag this as the one scale where the budget is matched only approximately; Section D.1 quantifies how much of DECO's margin this could account for.

All settings. Table 9 lists the measured non-embedding parameter count of every setting reported in this paper, together with its expert geometry $N_e \times d_e$ and whether its routed experts are gated. Three points are worth noting. First, *every* MoE baseline also exceeds its dense reference by $+0.16\%$ to $+3.14\%$, from the router and the same non-integral-N_e rounding; the excess is therefore a property of sparse routing in general and not of DECO in particular. Second, DECO is never meaningfully larger than the strongest baseline at its scale: it is smaller by 0.40% at 0.11B and by 0.62% at 0.53B, and larger by at most $5{,}146$ parameters ($< 0.0005\%$) at 0.24B, 1.18B and 2.4B. Third, DECO and BlockFFN share the same $N_e \times d_e$ geometry and the same non-gated experts, so their parameter counts differ by exactly the NormSiLU RMSNorm — 960 to 1,984 parameters, or at most 0.0009% — while DECO improves BPB over BlockFFN by 0.727%, 0.446%, 1.530%, 1.620% and 0.289% at the five scales where both are available. This pair is an exactly parameter-matched ablation of the DECO expert design.

D.1 Could the Parameter Excess Explain DECO's Margin?

Because DECO carries a router that the dense baseline does not, it is fair to ask whether its residual advantage over dense is bought with those extra parameters rather than earned by the architecture. We answer this in both directions.

Against dense: the excess is of the same order as the margin, so we claim parity only. Fitting $\mathrm{BPB} \propto N^b$ over our dense runs gives $b \approx -0.0861$ (Appendix H), so a relative parameter excess δ is worth an equivalent relative BPB improvement of $b\ln(1+\delta)$. For δ as measured in Table 8, this

Table 9: Measured non-embedding parameter count of every setting in this paper. "Gated" marks routed experts that use a gated MLP (three matrices) rather than the non-gated variant (two matrices); see Section 4.4. Δ is relative to the dense run of the same scale. Note that N_e and d_e in Table 7 are those of DECO; baselines adjust d_e to keep the parameter count matched, as listed here.

Scale	Setting	$N_e \times d_e$	Gated	Non-emb. params	Δ (%)
0.11B	TopP	44×61		108,967,680	+0.382
	DeepSeek-V3	44×41	✓	109,475,220	+0.850
	DeepSeek-V3	44×61		108,968,340	+0.383
	ReMoE	44×41	✓	109,474,560	+0.849
	ReMoE	44×61		108,967,680	+0.382
	BlockFFN	42×64		109,037,430	+0.446
	DECO	42×43	✓	109,521,915	+0.893
	DECO	42×64		109,038,390	+0.447
	Dense	—		108,552,960	0
0.24B	TopP	59×62		242,751,488	+0.637
	DeepSeek-V3	59×41	✓	241,604,705	+0.162
	DeepSeek-V3	59×62		242,752,609	+0.638
	ReMoE	59×41	✓	241,603,584	+0.161
	ReMoE	59×62		242,751,488	+0.637
	BlockFFN	57×64		242,324,539	+0.460
	DECO	57×43	✓	243,434,348	+0.920
	DECO	57×64		242,325,755	+0.461
	Dense	—		241,214,464	0
0.53B	TopP	79×63		536,597,760	+2.135
	DeepSeek-V3	79×42	✓	536,599,814	+2.135
	DeepSeek-V3	79×63		536,599,814	+2.135
	ReMoE	79×42	✓	536,597,760	+2.135
	ReMoE	79×64		541,856,000	+3.136
	BlockFFN	77×64		533,271,762	+1.502
	DECO	77×43	✓	535,835,440	+1.990
	DECO	77×64		533,273,426	+1.502
	Dense	—		525,382,400	0
1.18B	TopP	102×64		1,187,527,936	+0.479
	DeepSeek-V3	101×43	✓	1,185,975,611	+0.348
	ReMoE	102×64		1,187,527,936	+0.479
	BlockFFN	102×64		1,187,531,098	+0.480
	DECO	102×64		1,187,533,082	+0.480
	Dense	—		1,181,861,632	0
2.4B	TopP	147×64		2,423,557,120	+0.484
	DeepSeek-V3	146×43	✓	2,424,910,766	+0.540
	ReMoE	147×64		2,423,557,120	+0.484
	BlockFFN	147×64		2,423,561,677	+0.484
	DECO	147×64		2,423,563,661	+0.484
	Dense	—		2,411,891,200	0
7B	DECO	207×64		6,977,232,562	+0.492
	Dense	—		6,943,093,248	0

is -0.038%, -0.040%, -0.128%, -0.041%, -0.042% and -0.042% at the six scales. DECO's measured BPB advantage over dense is -0.033%, -0.119%, -0.089%, -0.056%, -0.027% and -0.007%. At 0.11B, 0.53B, 2.4B and 7B the parameter excess alone could account for DECO's entire edge. This is exactly why we claim that DECO *matches* dense rather than beats it: both quantities are an order of magnitude below the 0.333% relative-BPB standard deviation of a single training run under seed noise (Appendix F), so neither supports a claim of superiority in either direction.

Against the MoE baselines: the excess does not drive the ordering. The parameter excess cannot explain DECO's margin over the MoE baselines, for three reasons. (i) Every baseline also exceeds dense, and several by far more than DECO does, yet all of them lose to dense: at 0.53B, TopP carries $+2.135\%$ and ReMoE (non-gated) $+3.136\%$ excess parameters while trailing dense by 6.26% and 0.82% relative BPB. (ii) At 0.11B and 0.53B, DECO is *smaller* than the strongest baseline (by 0.40%

and 0.62%) and still leads it by 0.586% and 0.471% relative BPB. (iii) The DECO/BlockFFN pair is matched to within 0.0009% of its parameter count, and DECO leads by 0.289% to 1.620% relative BPB; four of those five margins exceed the 0.333% seed-noise band and the fifth, at 2.4B, sits just below it. Under the scaling exponent above, closing the DECO–BlockFFN gaps by size alone would require BlockFFN to be 3.4% to 21% larger, roughly four orders of magnitude more than the 960 to $1{,}984$ parameters that actually separate the two.

Activated parameters. For completeness, the last two rows of Table 8 report the activated non-embedding parameter count of DECO at the 20% routed activation ratio, which is 50.2% to 54.8% of the dense count; the dense first-layer FFN, the shared expert and the router are always active, so the per-token parameter budget does not fall to 20% of dense. The DeepSeek-V3 baselines realize the target ratio deterministically through TopK, at k/N_e of $9/44$, $12/59$, $16/79$, $20/101$ and $29/146$, i.e. 20.45%, 20.34%, 20.25%, 19.80% and 19.86%.

E Impact of the Scaling Factor Initialization Value

Table 10: The performance of DECO (0.11B) when the learnable expert-wise scaling factors are initialized with different values.

Init Value	0.01	0.025	0.05	0.1	0.25	0.5	1.0
PPL$_\downarrow$	34.71	35.66	34.61	34.36	**34.26**	35.20	34.74
Task$_\uparrow$	36.86	36.57	36.64	**36.90**	36.72	36.57	36.54

In this section, we conduct experiments on DECO when its learnable expert-wise scaling factors are initialized with different values. The results are shown in Table 10. The performance of DECO does not change monotonically. Empirically, the best performance is achieved when the scaling factors are initialized around 0.1–0.25. In our experiments in Section 4, we choose the initialization value of 0.1 for all DECO settings.

F Repeated Experiments

Table 11: The repeated experimental results on the 0.11B and 0.24B scales.

Setting	5-run PPL$_\downarrow$	Average PPL$_\downarrow$
Dense (0.11B)	34.40, 34.28, 34.49, 34.59, 34.38	**34.43±0.105**
DECO (0.11B)	34.36, 34.51, 34.90, 34.91, 34.75	34.69±0.218
Dense (0.24B)	27.85, 27.77, 27.15, 27.79, 28.07	27.73±0.307
DECO (0.24B)	27.74, 27.17, 27.71, 27.22, 27.81	**27.53±0.276**
Setting	5-run Task$_\uparrow$	Average Task$_\uparrow$
Dense (0.11B)	36.80, 36.93, 36.71, 36.67, 36.81	36.78±0.090
DECO (0.11B)	36.90, 36.78, 36.73, 37.15, 36.65	**36.84±0.174**
Dense (0.24B)	39.01, 38.85, 39.14, 38.96, 38.93	38.98±0.096
DECO (0.24B)	39.18, 39.35, 38.91, 39.37, 38.85	**39.13±0.217**

Large-scale pre-training is costly enough that experiments in this domain are typically run only once. To rigorously validate the reliability of our findings, we conducted repeated experiments at the 0.11B and 0.24B scales, performing five independent runs with different random seeds for each setting. As detailed in Table 11, DECO demonstrates highly stable and competitive performance. All four settings have population standard deviations below 0.31 PPL and 0.22 points, so the Dense-DECO differences reported above are of the same order as seed noise. Table 12 and the per-benchmark tables report the first seed of every setting, the same seed index throughout; at 0.11B that seed happens to be DECO's best of five, and averaging over seeds reverses the sign, leaving DECO 0.26 PPL behind dense there, while at 0.24B DECO stays ahead, by 0.20 PPL. Both gaps are inside the run-to-run spread, which is exactly why we claim parity rather than superiority. The task-accuracy averages do not reverse at either scale, so the reversal is confined to perplexity.

Furthermore, DECO not only closely approaches the Dense baseline on the 0.11B scale, but it actually slightly outperforms it on the 0.24B scale, achieving a lower average perplexity (27.53 ± 0.276) and a higher average task score (39.13 ± 0.217). Together, these results confirm that DECO remains competitive with the Dense setting across seeds. We take the largest single-run standard deviation across these four settings as the seed-noise scale used in Figure 3: 0.333% in relative BPB (Dense, 0.24B) and 0.217 points in task accuracy (DECO, 0.24B).

G Raw Perplexity and Bits-per-Byte at All Six Scales

Here we list the raw numbers behind Figure 3 and state the training configuration each group of runs was produced under.

Table 12: C4 English validation perplexity and bits-per-byte at all six total-parameter scales. **Perplexity is comparable within a column but not across the vertical rule**, because the two column groups use different tokenizers ($\beta_1 = 4.295$ and $\beta_2 = 4.739$ bytes per token); this is why PPL rises from 1.18B to 2.4B while BPB, which does not depend on tokenization, keeps falling. MoE baselines at 7B are not yet available due to cost considerations.

	Tokenizer 1 ($\beta = 4.295$)				Tokenizer 2 ($\beta = 4.739$)	
Method	0.11B	0.24B	0.53B	1.18B	2.4B	7B
Perplexity (↓)						
TopP	40.98	33.55	26.60	22.22	20.15	–
DeepSeek-V3	35.09	28.19	22.19	18.99	19.32	–
ReMoE	35.13	28.99	22.49	18.69	19.26	–
BlockFFN	35.27	28.16	22.94	19.28	19.24	–
Dense	34.40	27.85	21.93	18.41	19.09	15.693
DECO (ours)	**34.36**	**27.74**	**21.87**	**18.38**	**19.08**	**15.690**
Bits-per-byte (↓)						
TopP	1.2471	1.1799	1.1020	1.0416	0.9144	–
DeepSeek-V3	1.1950	1.1215	1.0411	0.9888	0.9014	–
ReMoE	1.1954	1.1309	1.0456	0.9834	0.9006	–
BlockFFN	1.1967	1.1211	1.0523	0.9939	0.9003	–
Dense	1.1884	1.1174	1.0371	0.9784	0.8979	0.8382
DECO (ours)	**1.1880**	**1.1161**	**1.0362**	**0.9778**	**0.8977**	**0.8381**

Two groups of runs. The 0.11B, 0.24B, 0.53B, and 1.18B models were trained with one tokenizer, while the 2.4B and 7B models were trained with a second tokenizer whose tokens are longer on average ($\beta_2 > \beta_1$ in Table 12). Within each scale, every setting we compare — Dense, DECO, and the MoE baselines — shares the same tokenizer, the same token budget, and the same schedule. Every comparison drawn anywhere in this paper is a comparison inside one scale, so the quantity we report, the gap to the equally-budgeted dense run, is always measured under a controlled setting.

Why we do not compare perplexity across the two groups. Perplexity is a per-token quantity, so its value depends on how the tokenizer segments the text: a tokenizer that produces longer tokens carries more information per token and therefore yields a larger perplexity on the same corpus, independently of model performance. The two tokenizers used here average $\beta_1 = 4.295$ and $\beta_2 = 4.739$ UTF-8 bytes per token on the C4 English validation set. The consequence is visible in Table 12: the perplexity of every setting *rises* between 1.18B and 2.4B — for the dense model from 18.41 to 19.09 — even though the larger models are the better ones. Bits-per-byte removes this artifact by normalizing the loss by the byte content of a token, Eq. (4); on the same runs it decreases monotonically with scale, from 1.1884 to 0.8382 for dense. Cross-scale statements in this paper are therefore made in BPB only, and perplexity is reported here for completeness and for within-scale comparison.

H Storage Cost of Matching DECO

Figure 1 asks for three properties at once: dense-level quality, low per-token computation, and a small stored-parameter footprint. The first two are measured directly (Figure 3, Table 5). This section

Table 13: Storage a sparse model would need to match DECO's quality. δ is the relative BPB deficit to DECO of the strongest MoE baseline at that scale; the budget ratio is $(1+\delta)^{1/0.0861}$, applied to the matched parameter budget of the scale. Footprints are weight bytes for the whole model, embeddings included, in GB at 4 bits and, in parentheses, 16 bits per parameter. The 7B row is an extrapolation, as no baseline was trained at that scale: it applies the δ range of the five measured scales.

Scale	Strongest baseline	δ	Budget Ratio	Weight footprint, INT4 (FP16) DECO	matching baseline
0.11B	DeepSeek-V3	0.59%	1.071	0.10 (0.41)	0.11 (0.43)
0.24B	BlockFFN	0.45%	1.053	0.18 (0.73)	0.19 (0.75)
0.53B	DeepSeek-V3	0.47%	1.056	0.34 (1.34)	0.35 (1.38)
1.18B	ReMoE	0.57%	1.069	0.68 (2.70)	0.71 (2.84)
2.4B	BlockFFN	0.29%	1.034	1.44 (5.76)	1.47 (5.89)
7B	*extrapolated*	0.29–0.59%	1.03–1.07	3.68 (14.74)	3.78–3.90 (15.1–15.6)

quantifies the third, and answers how much more an existing sparse design would have to store to catch up with DECO.

How much storage does DECO cost? Against the dense model it matches, DECO stores 0.45%–0.49% more non-embedding parameters at five of the six scales and 1.50% at 0.53B (Table 8), while reading only 50.2%–54.8% of the dense non-embedding parameters per token. Every MoE baseline carries a comparable or larger overhead, $+0.16\%$ to $+3.14\%$, so the comparisons in Section 4.1 are storage-matched to within 1.0% at five of the six scales, and to within 3.2% at 0.53B.

How much storage would an existing MoE design need to match DECO? At equal storage the existing designs are not equally good: the strongest baseline at each scale trails DECO by 0.29%–0.59% relative BPB. We convert such a deficit into a parameter budget using the dense scaling law fitted on our own six dense runs, $\mathrm{BPB} = 5.854\,N^{-0.0861}$ with N the non-embedding parameter count ($R^2 = 0.995$, that is 5.8% lower BPB per doubling), under which erasing a relative BPB deficit δ takes $(1+\delta)^{1/0.0861}$ times the parameters. These baselines therefore need 3.4%–7.1% more stored parameters to reach DECO, and the threshold-based TopP baseline, whose deficit is 1.9%–6.5%, needs $1.24\times$–$2.08\times$. Table 13 states this in bytes.

Why conventional MoE is not simply "equally sized". The comparison above holds the stored parameter count fixed, which is the regime an edge device imposes. Deployed MoE models are instead built the other way round: they fix the per-token computation and raise the stored parameter count until the quality target is met, giving stored-to-activated ratios of $3.6\times$ for Mixtral 8x7B (Jiang et al., 2024) and $18.1\times$ for DeepSeek-V3 (Liu et al., 2024). DECO's ratio is $1.83\times$–$2.00\times$. At the per-token computation of our 7B model, 3.48B activated non-embedding parameters, a design at DeepSeek-V3's ratio would store 63B parameters, 29 GB at 4 bits, against DECO's 6.98B and 3.25 GB. Both figures are non-embedding; the 3.68 GB in Table 13 includes embeddings. Those models are of course far stronger than a 7B one; the point is that their quality is bought on the one axis an edge device cannot pay on.

How to read these numbers. The budget ratios are an extrapolation from a curve fitted to six points, and carry three caveats. First, they assume that the MoE and dense curves share the exponent -0.0861; taking instead the steepest and shallowest slopes between adjacent scales, -0.120 and -0.065, widens the 3.4%–7.1% range to 2.4%–9.5%. Second, they convert a deficit into parameters alone, whereas the dense runs they are fitted on scale tokens with parameters (Table 7); adding parameters at a fixed token budget buys less than the fit predicts, so the stated budgets are lower bounds. Third, one of the five δ values, 0.29% at 2.4B, is within the 0.333% seed noise of a single run (Appendix F), so the ratios should be treated as approximate ranges rather than precise estimates; what is not within noise is that all 20 baseline-scale cells fall on the same side (Section 4.1).

I THEORETICAL MOTIVATION OF NORMSILU

Algorithm 1 Pseudocode of NormSiLU.

```
# Nt: num_tokens; De: dim_expert; Dh: dim_hidden
# Ne: num_experts; Na: num_active_experts
# learnable RMSNorm layer, weight shape is [De]
rms_norm = RMSNorm(dim=dim_expert)

def NormSiLU(input_hidden, up_proj_weight, intermediate_state):
    # input_hidden: [Nt, Dh], the input hidden state of MoE
    # up_proj_weight: [Ne, De, Dh], the up-projection weights of experts
    # intermediate_state: [Nt, Na, De], the expert intermediate state
    produced by a sparse linear operation between "input_hidden" and "
    up_proj_weight"

    # inter-expert mean normalization: conducted at the "num_experts"
    dimension
    up_proj_avg = mean(up_proj_weight, dim=0)  # [De, Dh], fixed during
    inference
    intermediate_avg = matmul(input_hidden, up_proj_avg.T)  # [Nt, De]
    intermediate_state -= intermediate_avg.unsqueeze(1)

    # intra-expert RMS normalization: conducted at the "dim_expert"
    dimension
    intermediate_state = rms_norm(intermediate_state)
    return SiLU(intermediate_state)  # standard SiLU activation
```

In this section, we first provide the complete pseudocode in Algorithm 1. Next, to motivate NormSiLU theoretically, we consider the post-activation output of all experts. Let $\mathbf{W}_{up} \in \mathbb{R}^{d_h \times (N_e d_e)}$ denote the flattened concatenated up-projection weights of N_e experts, and let $\mathbf{x} \in \mathbb{R}^{d_h}$ be the input hidden state. The post-activation intermediate state is:

$$\mathbf{y} = \mathrm{SiLU}(\mathrm{Norm}(\mathbf{z})),\ \mathbf{z} = \mathbf{W}_{up}^T \mathbf{x} \in \mathbb{R}^{N_e d_e}. \tag{6}$$

For simplicity, we first assume that the operator "Norm" is implemented as a vanilla layer normalization (Ba et al., 2016) across the entire concatenated expert dimension. Let $\mathbf{g} = \nabla_{\mathbf{y}} \mathcal{L}$ be the upstream gradient from the language modeling loss $\mathcal{L}$. The gradient with respect to the weights $\mathbf{W}_{up}$ is formulated as:

$$\begin{aligned} \nabla_{\mathbf{W}_{up}} \mathcal{L} &= \mathbf{x}(\nabla_{\mathbf{z}} \mathcal{L})^T = \mathbf{x}(\mathbf{J}_{norm}^T \mathbf{D}_{silu} \mathbf{g})^T, \\ \mathbf{u} &= \mathrm{Norm}(\mathbf{z}) = \frac{\mathbf{z}_0}{||\mathbf{z}_0||_{rms}},\ \mathbf{z}_0 = \mathbf{z} - \bar{\mathbf{z}}, \end{aligned} \tag{7}$$

where $\mathbf{D}_{silu} = \mathrm{diag}(\sigma(\mathbf{u}) + \mathbf{u} \odot \sigma(\mathbf{u}) \odot (1 - \sigma(\mathbf{u})))$ is the diagonal Jacobian matrix of SiLU, σ is the sigmoid function, and $\bar{\mathbf{z}}$ denotes the mean of $\mathbf{z}$. Letting $n = N_e d_e$ be the total dimension of $\mathbf{z}$ and $\mathbf{1}$ be the all-ones vector, the Jacobian matrix of the layer normalization $\mathbf{J}_{norm}$ is:

$$\begin{aligned} \mathbf{J}_{norm} &= \frac{\partial \mathbf{u}}{\partial \mathbf{z}} \\ &= \frac{1}{||\mathbf{z}_0||_{rms}} \left(\mathbf{I}_n - \frac{1}{n} \mathbf{1}\mathbf{1}^T - \frac{\mathbf{z}_0 \mathbf{z}_0^T}{n||\mathbf{z}_0||_{rms}^2} \right). \end{aligned} \tag{8}$$

Since $\mathbf{1}^T \mathbf{z}_0 = 0$, the matrix inside the parentheses is an orthogonal projection of rank $n - 2$, whose spectral norm is exactly 1; hence $||\mathbf{J}_{norm}||_2 = 1/||\mathbf{z}_0||_{rms}$. Assuming the elements of $\mathbf{W}_{up}$ follow a zero-centered i.i.d. normal distribution, we can statistically approximate the RMS norm as:

$$||\mathbf{z}_0||_{rms} \approx \frac{1}{\sqrt{d_h n}} ||\mathbf{W}_{up}||_F \cdot ||\mathbf{x}||_2. \tag{9}$$

Applying the sub-multiplicative property of matrix norms, the Frobenius norm of the gradient is bounded by:

$$\begin{aligned} &||\nabla_{\mathbf{W}_{up}} \mathcal{L}||_F \\ &\leq ||\mathbf{x}||_2 \cdot ||\mathbf{J}_{norm}||_2 \cdot ||\mathbf{D}_{silu}||_2 \cdot ||\mathbf{g}||_2 \\ &\leq ||\mathbf{g}||_2 \cdot \mathcal{O}(\sqrt{d_h n} \cdot ||\mathbf{W}_{up}||_F^{-1}). \end{aligned} \tag{10}$$

Therefore, as long as $\mathbf{W}_{up}$ is initialized with a proper Frobenius norm, this normalization keeps the gradient scale bounded and invariant to the input magnitude, preventing gradient explosion.

However, the above paradigm possesses a critical systems-level flaw: computing a global layer normalization requires the explicit materialization of $\mathbf{z}$ across *all* experts. This inherently violates the core sparsity principle of MoE, as it forces the computation of inactive experts.

To resolve this bottleneck, our final implementation of NormSiLU decouples the operation into a dual-stage mechanism: an inter-expert mean normalization and an intra-expert RMS normalization. During inference, the global inter-expert averaging operator mathematically reduces to: $\mathrm{Avg}(\mathbf{W}_{up}^T\mathbf{x}) = \bar{\mathbf{w}}_{up}^T\mathbf{x}$, where $\bar{\mathbf{w}}_{up} \in \mathbb{R}^{d_h \times d_e}$ is the fixed average up-projection weight. By contrast, the RMS normalization is strategically restricted to the intra-expert dimension (d_e) and applied *only* to activated experts. This dual-stage design preserves the theoretical gradient-bounding stability while strictly adhering to the computational constraints of sparse inference.

J Evaluation Results on Individual Benchmarks

In this section, we provide the evaluation results of DECO and baselines on each individual benchmark. The results of the 0.11B, 0.24B, 0.53B, 1.18B, 2.4B, and 7B settings are shown in Table 14, Table 15, Table 16, Table 17, Table 18, and Table 19, respectively.

Table 14: The evaluation scores of 0.11B settings on individual benchmarks.

	PIQA	SIQA	HellaSwag	ARC-E	ARC-C	WinoGrande	LAMBADA	**Avg.**
TopP	60.17	35.21	27.67	42.38	18.86	50.91	7.74	34.71
DeepSeek-V3 (GA)	60.12	35.93	27.85	44.15	19.45	50.75	15.89	36.31
DeepSeek-V3 (NG)	60.66	36.63	27.76	45.76	20.05	50.43	15.85	36.73
ReMoE (GA)	60.34	36.59	27.72	46.34	18.43	51.22	15.60	36.61
ReMoE (NG)	59.74	35.16	27.96	46.09	19.54	49.96	17.76	36.60
BlockFFN	60.83	36.28	27.62	44.87	19.45	50.12	16.55	36.53
DECO (GA)	58.22	35.93	27.65	42.51	19.03	51.30	15.84	35.78
Dense	60.77	35.57	28.03	45.88	20.22	50.59	16.51	**36.80**
DECO (NG)	60.94	36.85	27.96	45.84	20.22	50.67	15.80	**36.90**

Table 15: The evaluation scores of 0.24B settings on individual benchmarks.

	PIQA	SIQA	HellaSwag	ARC-E	ARC-C	WinoGrande	LAMBADA	**Avg.**
TopP	61.37	36.49	28.57	45.75	19.03	50.51	12.63	36.34
DeepSeek-V3 (GA)	62.79	37.87	29.61	48.53	20.90	52.17	19.84	38.82
DeepSeek-V3 (NG)	63.28	37.67	29.64	49.07	20.65	49.25	19.76	38.47
ReMoE (GA)	62.62	37.67	29.10	48.65	20.05	50.67	18.28	38.15
ReMoE (NG)	63.22	36.64	29.40	49.71	19.71	50.83	20.45	38.57
BlockFFN	63.38	35.82	29.61	48.82	20.99	50.51	19.95	38.44
DECO (GA)	61.86	37.10	28.44	47.18	19.37	49.64	17.43	37.29
Dense	63.00	36.90	29.57	51.47	22.01	50.51	19.62	**39.01**
DECO (NG)	64.36	36.18	29.47	51.98	20.73	52.09	19.48	**39.18**

Table 16: The evaluation scores of 0.53B settings on individual benchmarks.

	PIQA	SIQA	HellaSwag	ARC-E	ARC-C	WinoGrande	LAMBADA	**Avg.**
TopP	64.53	36.44	31.42	50.55	22.53	51.07	14.13	38.67
DeepSeek-V3 (GA)	66.32	38.38	32.71	55.18	24.91	51.07	28.06	42.38
DeepSeek-V3 (NG)	65.94	37.87	32.75	54.67	23.38	52.49	27.03	42.02
ReMoE (GA)	66.05	38.69	32.08	55.26	24.32	52.33	27.19	42.27
ReMoE (NG)	66.92	38.79	32.66	55.93	24.57	51.41	27.10	42.48
BlockFFN	66.10	38.59	32.51	56.36	24.15	51.38	27.30	42.34
DECO (GA)	66.14	38.74	32.69	56.41	23.72	51.78	26.57	42.29
Dense	66.87	37.77	32.92	56.27	24.74	51.70	29.34	**42.80**
DECO (NG)	65.94	39.61	32.78	57.11	25.43	53.28	25.50	**42.81**

Table 17: The evaluation scores of 1.18B settings on individual benchmarks.

	PIQA	SIQA	HellaSwag	ARC-E	ARC-C	WinoGrande	LAMBADA	**Avg.**
TopP	66.16	37.36	34.10	57.32	22.87	49.41	20.49	41.10
DeepSeek-V3 (GA)	69.53	39.92	36.65	62.21	28.41	53.28	34.52	46.36
ReMoE (NG)	68.93	40.07	37.43	61.20	28.07	53.43	35.46	46.37
BlockFFN	70.02	40.28	36.81	61.91	28.50	55.80	34.60	46.85
Dense	70.46	40.69	37.33	63.38	28.58	52.72	35.73	**46.98**
DECO (NG)	70.24	40.89	37.42	62.96	29.69	54.93	35.53	**47.38**

Table 18: The evaluation scores of the 2.4B settings on individual benchmarks.

	PIQA	SIQA	HellaSwag	ARC-E	ARC-C	WinoGrande	LAMBADA	**Avg.**
TopP	71.93	41.30	40.92	67.63	33.19	56.67	42.09	50.53
DeepSeek-V3 (GA)	73.72	41.56	41.61	69.36	35.32	58.33	43.53	51.92
ReMoE (NG)	72.96	41.56	42.64	68.16	36.43	56.51	45.35	51.94
BlockFFN	73.12	41.81	42.15	69.28	34.30	56.59	41.84	51.30
Dense	73.34	42.48	42.21	68.90	36.09	57.38	45.10	**52.21**
DECO (NG)	73.23	42.30	42.52	68.64	36.69	57.54	44.93	**52.26**

Table 19: The evaluation scores of the 7B settings on individual benchmarks. Due to cost considerations, the 7B experiments do not involve MoE baselines.

	PIQA	SIQA	HellaSwag	ARC-E	ARC-C	WinoGrande	LAMBADA	**Avg.**
Dense	76.77	43.96	49.41	76.43	44.45	60.69	54.69	**58.06**
DECO (NG)	76.88	44.08	49.58	76.34	44.31	60.80	54.52	**58.07**