LongCat

Scaling Embeddings Outperforms Scaling Experts in Language Models

Hong Liu, Jiaqi Zhang*, Chao Wang, Xing Hu, Linkun Lyu,
Jiaqi Sun, Xurui Yang, Bo Wang, Fengcun Li, Yulei Qian, Lingtong Si,
Yerui Sun, Rumei Li, Peng Pei*, Yuchen Xie, Xunliang Cai

Meituan LongCat Team

ABSTRACT

While Mixture-of-Experts (MoE) architectures have become the standard for sparsity scaling in large language models, they increasingly face diminishing returns and system-level bottlenecks. In this work, we explore embedding scaling as a potent, orthogonal dimension for scaling sparsity. Through a comprehensive analysis and experiments, we identify specific regimes where embedding scaling achieves a superior Pareto frontier compared to expert scaling. We systematically characterize the critical architectural factors governing this efficacy—ranging from parameter budgeting to the interplay with model width and depth. Moreover, by integrating tailored system optimizations and speculative decoding, we effectively convert this sparsity into tangible inference speedups. Guided by these insights, we introduce LongCat-Flash-Lite, a 68.5B parameter model with ∼3B activated trained from scratch. Despite allocating over 30B parameters to embeddings, LongCat-Flash-Lite not only surpasses parameter-equivalent MoE baselines but also exhibits exceptional competitiveness against existing models of comparable scale, particularly in agentic and coding domains.

Hugging Face: https://huggingface.co/meituan-longcat/LongCat-Flash-Lite

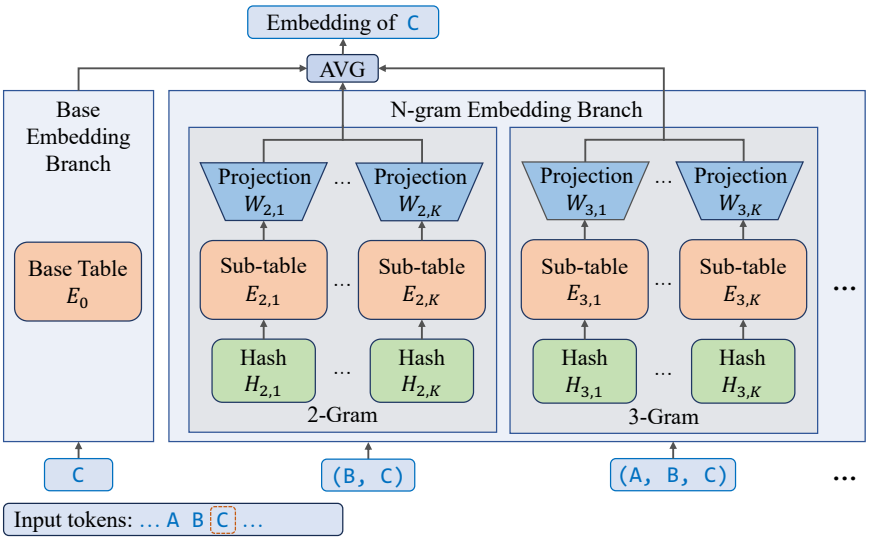


Figure 1: The architecture of a N-gram Embedding layer [Huang et al., 2025]. The embedding of each token is augmented by the N-gram Embedding branch.

[1]Corresponding authors: zhangjiaqi39@meituan.com, peipeng@meituan.com

Contents

1 Introduction

The Mixture-of-Experts (MoE) architecture has firmly established itself as the dominant paradigm for scaling Large Language Models (LLMs), enabling massive parameter counts while maintaining manageable computational costs Lepikhin et al. [2021]. By dynamically routing tokens to a subset of experts, models decouple parameter capacity from computational cost, allowing LLMs to scale to trillons of parameters while keeping modest inference latency. However, as the model size and sparsity level increase, the marginal gain in performance diminishes, eventually approaching an efficiency saturation point [Abnar et al., 2025]. Furthermore, the practical expansion of experts is constrained by system-level bottlenecks, particularly the escalating communication overhead and memory bandwidth pressure in distributed training. This necessitates the exploration of alternative, orthogonal dimensions for scaling sparse parameters beyond the Feed-Forward Networks (FFNs).

In contrast to MoE, the embedding layer offers an overlooked, inherently sparse dimension with $O(1)$ lookup complexity. This allows for massive parameter expansion without routing overheads—effectively achieving parameter extension without computation explosion. Theoretical foundations for this dimension have been established by scaling laws with vocabulary [Tao et al., 2024], which posit that larger models necessitate proportionally larger vocabularies to maximize computation efficiency. To exploit this potential, diverse strategies have been proposed. One prominent direction is structural expansion, exemplified by Per-Layer Embedding (PLE) [Google DeepMind, 2025, Sadhukhan et al., 2026, bcml labs, 2025], which allocates independent embedding parameters to each layer to scale capacity. Another key direction is vocabulary expansion via n-grams to densify information per token. This concept traces back to lookup-table language models [Huang et al., 2021] in the RNN era and has recently been advanced in LLMs [Clark et al., 2022, Huang et al., 2025, Pagnoni et al., 2025, Cheng et al., 2026]. These approaches collectively highlight the embedding layer as a fertile ground for scaling.

Despite the recent interest in expanding embedding parameters in LLMs, several key challenges remain underexplored. First, the comparative scaling efficiency between expert parameters and embedding parameters is not well understood, leaving the optimal allocation of capacity between these two sparse dimensions ambiguous. Second, the constraints of scaling embeddings are still not systematically characterized: it remains unclear how factors such as the total parameter budget, vocabulary size, initialization schemes, and the trade-offs between model width and depth jointly influence the effectiveness and stability of embedding scaling. Third, while some methods for scaling embeddings have been proposed, it is still unclear which scaling strategy is more effective and efficient under different regimes. Finally, scaling embeddings alters the input/output characteristics of the model during decoding, potentially impacting the overall I/O efficiency, yet its consequences for end-to-end inference performance remain insufficiently analyzed and optimized.

In this technical report, we present a study to address these challenges and establish a robust framework for embedding scaling. Our contributions are as follows:

- **Comparison of Embedding Scaling vs. Expert Scaling**: Through comprehensive scaling experiments across diverse scenarios, we identify specific regimes where embedding scaling achieves a superior Pareto frontier compared to increasing expert numbers, offering a high-efficiency alternative for model scaling.
- **Impact Analysis of Architectural Factors**: We establish the complete set of architectural factors determining embedding scaling efficacy, covering the integration timing, parameter budgeting, hash collisions, hyperparameter settings and initialization of embedding, together with the effects of model width and depth. Besides, we investigate different methods of scaling embedding and find that N-gram Embedding offers the most robust scalability.
- **Inference Efficiency and System Optimization**: We demonstrate that N-gram Embedding largely reduce I/O bottlenecks in MoE layers, particularly when paired with speculative decoding to maximize hardware utilization. Addressing the concomitant embedding overhead, we propose a specialized *N-gram Cache* and synchronized kernels, ensuring that the reduction in active parameters translates directly to lower latency and higher throughput.

Based on these findings, we introduce and open-source LongCat-Flash-Lite, a model trained from scratch with 68.5B total parameters and 2.9B$\sim$4.5B activated parameters depending on the context. Our evaluation demonstrates that LongCat-Flash-Lite not only surpasses a parameter-equivalent MoE baseline—validating the superior efficacy of allocating over 30B parameters to embeddings rather than experts—but also exhibits competitive performance against existing models of similar scale, particularly in agentic and coding tasks.

2 N-gram Embedding Layer

To scale the embedding parameters, we adopt the N-gram Embedding introduced in Clark et al. [2022], Huang et al. [2025], Pagnoni et al. [2025], which augments the representation of the embedding module by expanding a vocabulary-free n-gram embedding table. Specifically, for the i-th token t_i in a sequence, the augmented embedding e_i is calculated

as follows

$$e_i = \frac{1}{N}\big(E_0(t_i) + \sum_{n=2}^{N} E_n(\mathcal{H}_n(t_{i-n+1}, ..., t_i))\big), \quad \text{where } t_j = 0 \text{ if } j \leq 0, \tag{1}$$

where $E_0 \in \mathbb{R}^{V_0 \times D}$ is the original base embedding table with hidden size D, $E_n \in \mathbb{R}^{V_n \times D}$ is the expanded embedding table, N denotes the maximum n-gram order and $\mathcal{H}_n$ denotes the hash mapping function. We use the polynomial rolling hash funciton:

$$\mathcal{H}_n(t_{i-n+1}, ..., t_i) = (\sum_{j=0}^{n-1} t_{i-j} * V_0^j)\%V_n. \tag{2}$$

To further enhance the model's expressive ability and reduce hash collisions, Clark et al. [2022], Huang et al. [2025] decompose each n-gram embedding table into K sub-tables with different vocabulary size. Huang et al. [2025] further incorporate additional linear projection to map the outputs back to the original embedding space. The final version of N-gram Embedding (also referred to as Over-Encoding in Huang et al. [2025]) is shown in Figure 1 and can be written as

$$e_i = \frac{1}{(N-1)K+1}\Big(E_0(t_i) + \sum_{n=2}^{N}\sum_{k=1}^{K} W_{n,k} E_{n,k}(\mathcal{H}_{n,k}(t_{i-n+1}, ..., t_i))\Big), \tag{3}$$

where $E_{n,k} \in \mathbb{R}^{V_{n,k} \times D/((N-1)K)}$ is a sub-table and $W_{n,k} \in \mathbb{R}^{D \times D/((N-1)K)}$ is the linear projection matrix. By setting the hidden size of sub-tables to be inversely proportional to the number of sub-tables, this design ensures that the parameter count of N-gram Embedding remains invariant with respect to N and K.

3 Comparative Analysis of Expert and Embedding Scaling

This section presents our empirical findings regarding the comparison between scaling embeddings and scaling experts.

Experiment Settings We integrate N-gram Embedding into the Longcat-Flash architecture [Meituan, 2025] and conduct scaling experiments via from-scratch pre-training across varying activated parameter budgets (280M, 790M, and 1.3B). To rigorously compare scaling strategies, we establish a framework that contrasts scaling via N-gram Embedding against scaling experts. Specifically, for N-gram Embedding scaling, we first train MoE models with varying base sparsity levels ranging from 35% to 98% and incrementally incorporate N-gram Embedding from specific sparsity levels. Crucially, at each sparsity level, the N-gram Embedding model is paired with a parameter-equivalent MoE baseline, which attains the same total parameter count by increasing the number of experts. All models are pre-trained on a corpus of 300B tokens. We evaluate model performance by monitoring training loss and validation loss on two meticulously constructed datasets, covering both Chinese and English.

3.1 Optimal Timing for N-gram Embedding Integration

A pivotal finding is that the scaling dynamics of N-gram Embedding diverge markedly depending on the sparsity level of the base model. Figure 2 presents three distinct scaling trajectories[1]: the standard MoE baseline (blue), N-gram Embedding applied to a base model with a low parameter ratio (green), and N-gram Embedding applied to a base model with a high parameter ratio (red). The figure illustrates that the MoE scaling curve adheres to a strict log-linear relationship. This implies that in low-ratio regimes, a marginal increase in the number of experts yields a substantial reduction in loss. Conversely, at higher ratios, achieving an equivalent loss reduction necessitates a significantly larger increase in expert parameters. Consequently, when N-gram Embedding is introduced at low parameter ratios, its scaling advantage fails to surpass the gains obtained by simply increasing the number of experts. In contrast, at high sparsity levels, the benefits of N-gram Embedding become significantly more pronounced. This observation leads to the following design principle regarding the incorporation of N-gram Embedding.

Summary: N-gram Embedding should be introduced when the number of experts exceeds its "sweet spot".

This result indicates that embedding scaling could be a promising scaling dimension orthogonal to expert scaling.

[1]We utilize the ratio of total parameters to activated parameters on the x-axis as a proxy for sparsity.

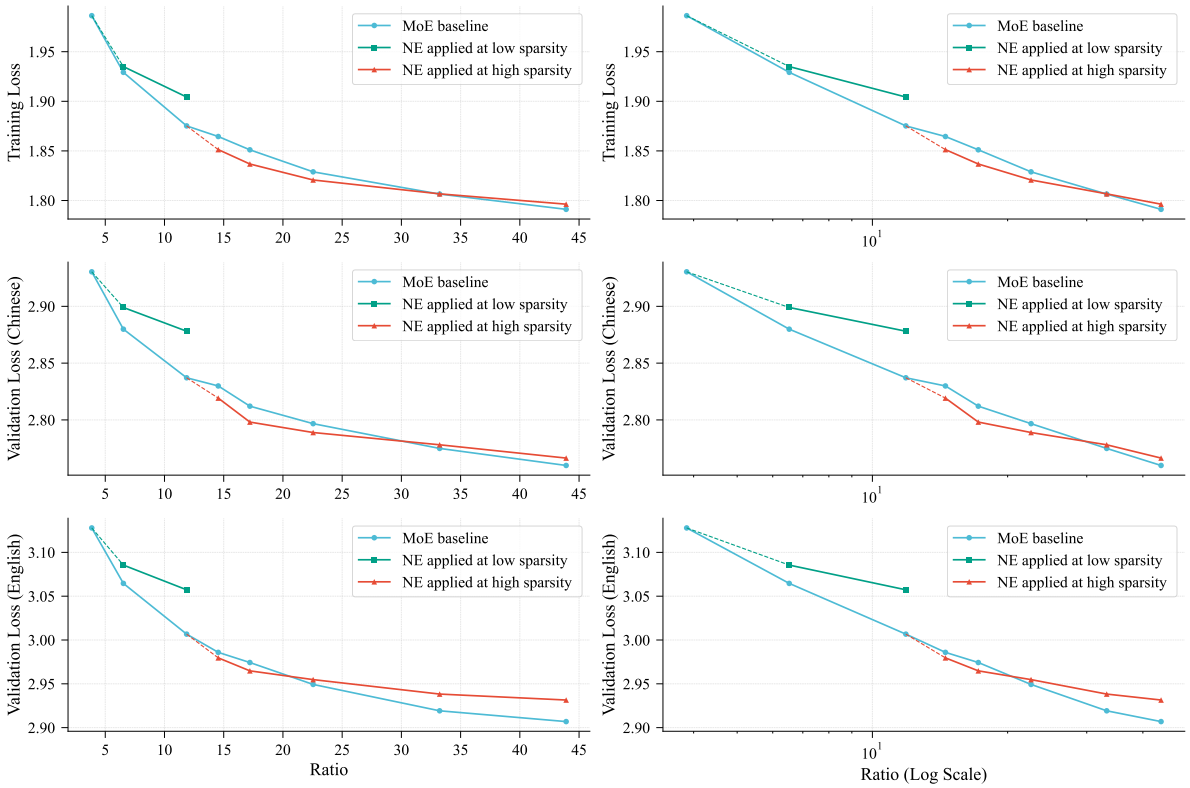


Figure 2: The scaling curve of MoE model and N-gram Embedding (NE) model. The horizontal axis is the ratio of total parameters to the activated parameters (280M). The axes of Figures on the right panel is converted to a logarithmic scale. For the two NE curves, we prepend a dashed line to connect the corresponding base MoE model without NE.

3.2 Integration Strategy

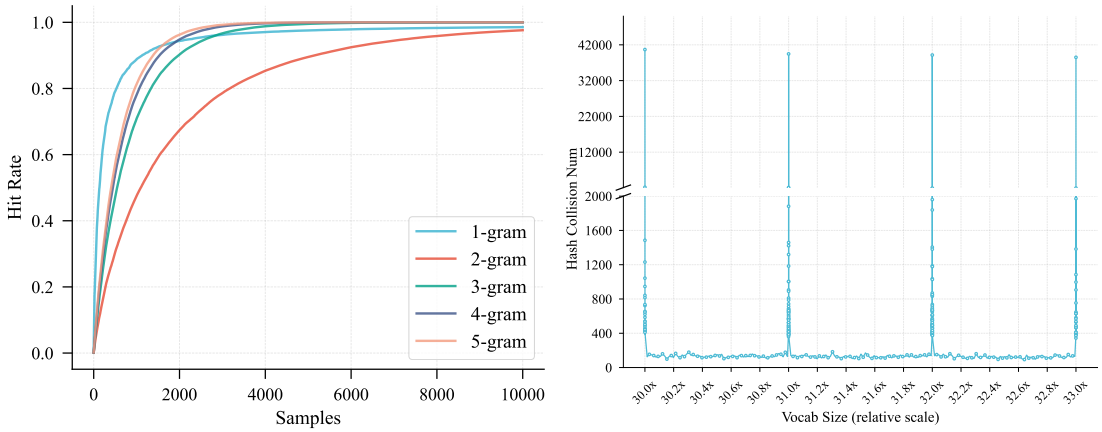
3.2.1 Parameter Budgeting for N-gram Embeddings

A closer inspection of Figure 2 reveals a distinct intersection between the blue and red curves: as the parameter ratio increases, the performance advantage of N-gram Embedding gradually diminishes and is eventually surpassed by the MoE baseline. This indicates that when a model allocates an excessive proportion of its parameter budget to N-gram Embedding, its performance becomes inferior to that of parameter-equivalent MoE baselines. This observation aligns with conclusions drawn in the concurrent work Engram [Cheng et al., 2026], which posits that the loss follows a U-shaped scaling curve as a function of the N-gram Embedding proportion. In Figure 2, the intersection point lies slightly above a ratio of 20. At this juncture, N-gram Embedding parameters constitute approximately 50% of the total parameter count (given that the base MoE model maintains a ratio of 12). Consequently, we derive a second principle from this phenomenon:

Summary: Allocate no more than 50% of the total parameter budget to N-gram Embedding.

3.2.2 Mitigating Hash Collisions via Vocabulary Sizing

In the context of N-gram Embedding, hash collisions force a single embedding vector to superimpose the semantics of multiple distinct n-grams. This collision-induced ambiguity impedes learning efficiency and consequently degrades model performance. We identify that selecting an appropriate vocabulary size is critical to mitigating high collision rates. During training, we observed that N-gram Embedding exhibits anomalously high hash collision rates at specific vocabulary sizes, particularly for 2-gram hashing. To investigate the underlying mechanics, we conduct a dual analysis focusing on: (1) **Vocabulary Hit Rate**, defined as the proportion of vocabulary entries activated at least once by the pre-training corpus; and (2) **Hash Collisions**, which quantifies the loss of unique token representation due to modulo-based indexing.



(a) (b)

Figure 3: **(a)** The vocabulary hit rate of different n-grams. **(b)** The collision number of 2-gram hashing at different vocabulary size. Sampling points are denser near integer multiples of vocabulary size and sparser elsewhere for clarity.

For the hit rate analysis, we use an n-gram vocabulary size set to $30\times$ the base vocabulary (128k). For the collision analysis, we sample a range of vocabulary sizes between $30\times$ and $33\times$ the base vocabulary size, computing n-gram collision counts over 100 training sequences for each configuration. The results are detailed in Figure 3.

Figure 3a illustrates that 2-gram hashing exhibits a gradual increase in hit rate, whereas higher-order n-gram hashing rapidly converges toward a hit rate of 1.0. Independent of the hit rate trends, we observe in Figure 3b that 2-gram hashing collision numbers display a strong, non-linear correlation with vocabulary size. A salient pattern emerges: collision counts spike noticeably when the vocabulary size approaches an integer multiple of the base vocabulary size. This phenomenon persists regardless of whether the n-gram vocabulary size is a prime number. Synthesizing these observations, Figure 3b motivates an additional design principle for configuring N-gram Embedding:

Summary: The vocabulary size of N-gram Embedding should significantly deviates from integer multiples of the base vocabulary size to prevent Hash collisions.

3.2.3 Sensitivity Analysis of Hyperparameters

We now examine the sensitivity of model performance to the internal configurations of N-gram Embedding, namely the n-gram order N and the number of sub-tables K.

Regarding the n-gram order N defined in Section 2, increasing N enables N-gram Embedding to capture richer contextual semantics, theoretically yielding embedding vectors with enhanced representational capacity. However, this also creates an extremely sparse distribution over the n-gram vocabulary, as high-order n-grams appear infrequently. This sparsity significantly exacerbates the challenge of learning effective embeddings.

Regarding the number of sub-tables K, this parameter governs the number of distinct hash functions applied to each n-gram, thereby substantially mitigating the probability of hash collisions. Nevertheless, empirical evidence suggests that increasing K beyond a certain threshold yields diminishing returns.

Utilizing the 790M activated-parameter model (corresponding to the initial data point on the red curve in Figure 6a), we conduct ablation studies across various combinations of N and K[2]. The results are summarized in Figure 4. It is evident that when both N and K are set to their minimal values ($N = 2$ and $K = 1$), the model exhibits notably inferior performance. Conversely, for $N \geq 3$ and $K \geq 2$, the performance variance across different configurations becomes relatively small, indicating that the model is robust to hyperparameter selection within this regime. Empirically, we observe that setting N in the range of 3 to 5 consistently yields near-optimal performance.

[2]For large N, numerical overflow during hash computation can be circumvented by applying the modulus operation prior to exponentiation.

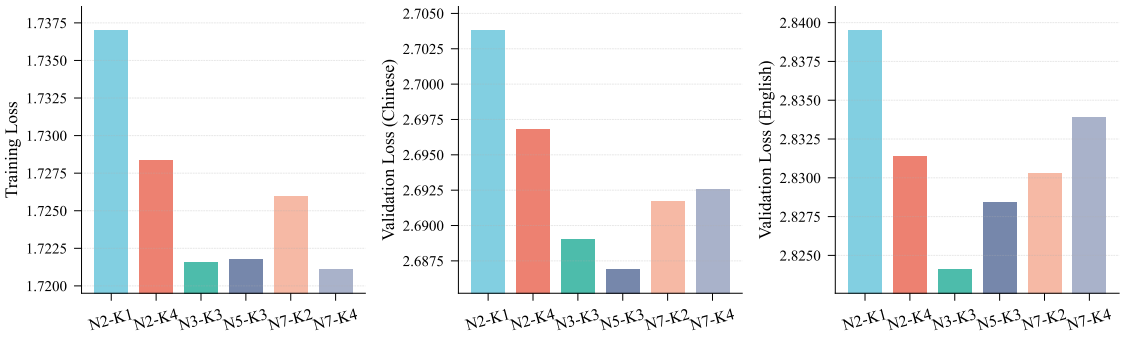


Figure 4: Comparison of training and validation loss under different combinations of N and K.

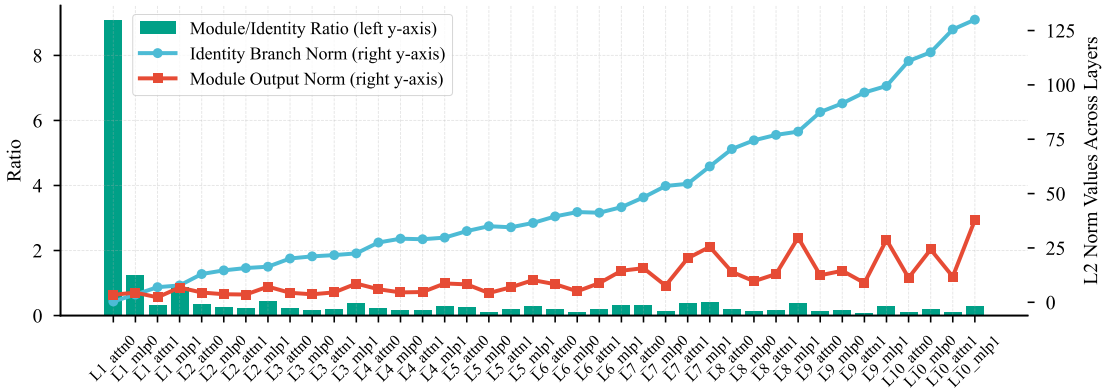


Figure 5: Layer-wise analysis of L2 norms for module outputs versus their corresponding identity branches, alongside the ratio of these norms. Each shortcut layer comprises two sub-layers, denoted by suffixes 0 and 1.

3.2.4 Embedding Amplification for Effective Training

In our preliminary experiments, we observed that suboptimal initialization of the embedding module can severely impede the efficacy of N-gram Embedding, preventing it from realizing its full potential. To validate this hypothesis, we revisited an early vanilla experiment configured as per Figure 2, but without any specific adjustments to the embedding module. After pre-training on 300B tokens, we compute the L2 norms of each module's output and its corresponding residual branch (identity path) across all layers, plotting the norms and their ratios in Figure 5.

In our early experiments, we find that suboptimal embedding initialization can significantly hinder the contribution of N-gram Embedding, preventing it from fully realizing its potential. To validate this hypothesis, we analyze an early vanilla experiment, which follows the configuration in Figure 2 but does not apply any adjustments to the embedding module. After training on 300B tokens, we plot, for all layers, the L2 norms of each module's output and its corresponding identity branch, as well as the ratio between these two norms, as shown in Figure 5.

Figure 5 exposes a critical disparity: the L2 norm of the first attention module's output is an order of magnitude larger (approximately $10\times$) than that of the corresponding identity branch, which essentially represents the output of the embedding module. This indicates that upon summation, the attention output dominates the residual stream, effectively "drowning out" the embedding signal. Although standard initialization of sub-tables and projection matrices in N-gram Embedding ensures that initial output norms match the baseline, this signal suppression phenomenon exacerbates significantly once training progress, leading to substantial performance degradation in N-gram Embedding models.

To mitigate this issue, we explore two strategies:

- **Scaling Factor:** Introducing a scaling factor (typically $\sqrt{D}$) to the embedding output to ensure a sufficient contribution to the forward pass.
- **Normalization:** Applying LayerNorm to the embedding output prior to merging with the residual branch. This similarly amplifies the embedding contribution, as LayerNorm enforces unit variance during the early stages of training.

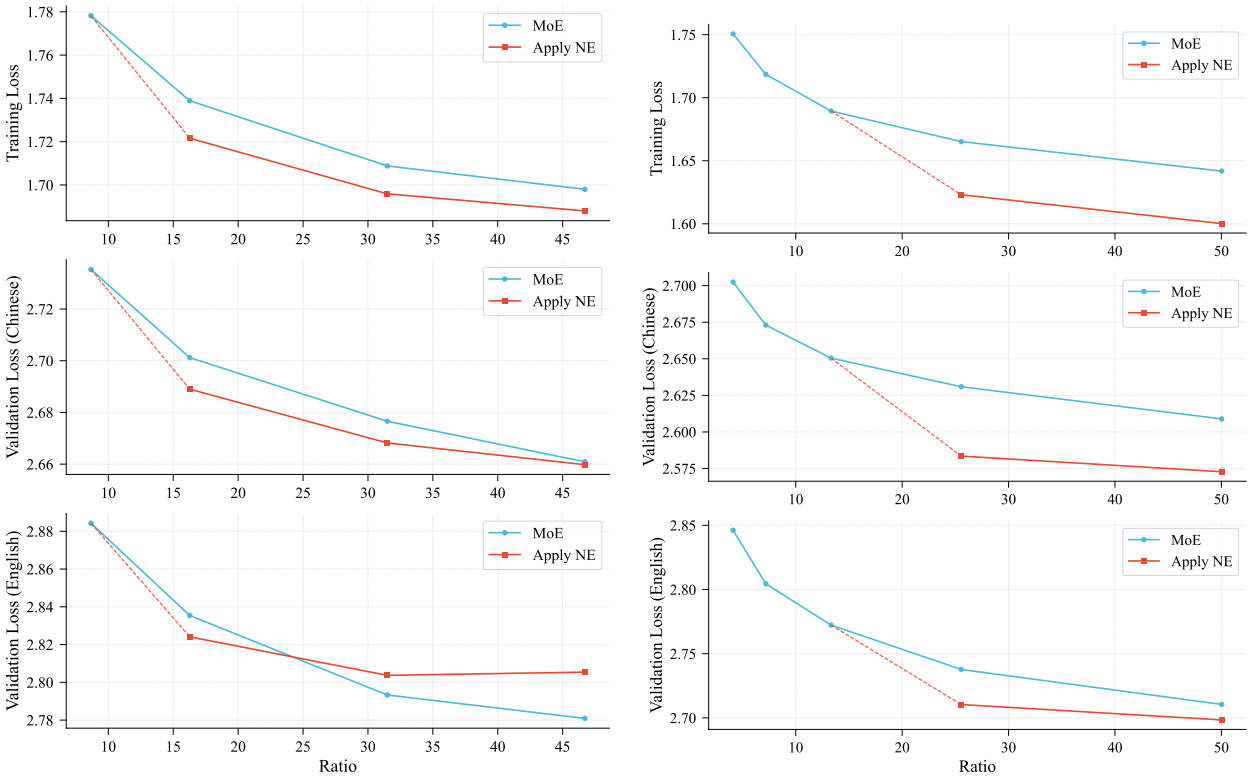


Figure 6: **(a)** Scaling curve at 790M activation size. **(b)** Scaling curve at 1.3B activation size.

Both techniques were originally proposed in Takase et al. [2025] with the primary objective of increasing residual branch variance to bound backward gradients and stabilize training. In our context, while we observed no significant impact on training stability, these methods—collectively termed **Embedding Amplification**—substantially enhance the performance of N-gram Embedding. In our experiments, applying Embedding Amplification yields superior performance compared to the vanilla baseline, with a consistent reduction of 0.02 in both the training loss and the two validation losses.

3.3 Scaling Properties across Model Width and Depth

3.3.1 Enhanced Advantage in Wider Models

This section investigates how the efficacy of N-gram embedding scaling evolves with increasing model width.

We conduct a series of scaling experiments at two larger activation scales (790M and 1.3B parameters), with model depth held constant (10 shortcut layers) while only the width (hidden size and module dimensions) varies. Figure 6 presents the resulting scaling curves. Our analysis reveals two key trends:

- When incorporated at an appropriate ratio, N-gram Embedding consistently yields a lower loss compared to a parameter-equivalent MoE baseline. This advantage gradually diminishes as the proportion of N-gram Embedding parameters increases, mirroring the behavior observed in Figure 2.
- Crucially, the intersection point between the N-gram Embedding curve and the MoE curve systematically shifts towards higher total-to-activated parameter ratios as the model width (activation size) increases. Specifically, for a 280M activation size, N-gram Embedding consistently underperforms its MoE counterpart once the ratio exceeds 30. At 790M, N-gram Embedding only underperforms on the English validation set at this ratio, while maintaining an advantage on all other metrics. Notably, at a 1.3B activation size, N-gram Embedding retains a clear advantage even at ratios as high as 50.

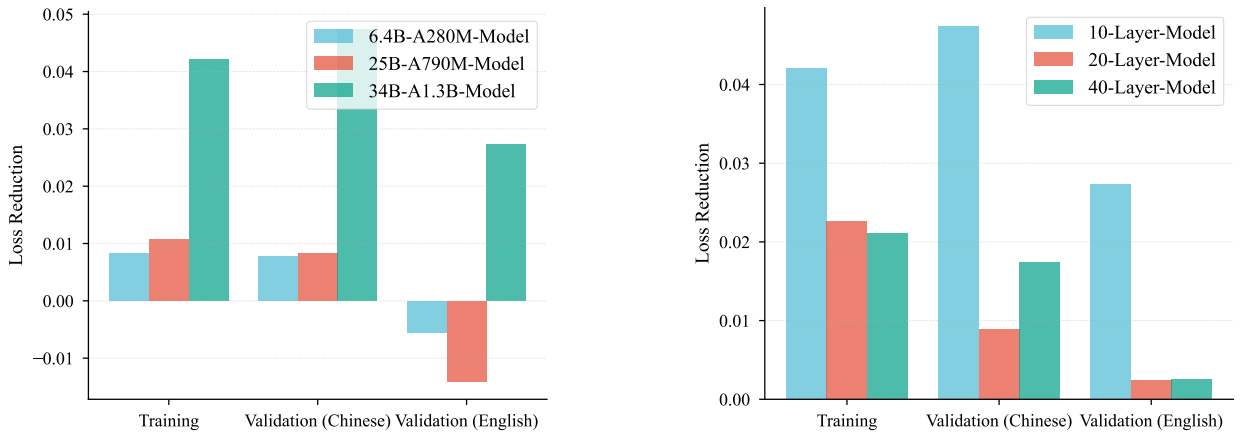


Figure 7: Loss reduction of models with N-gram Embedding compared to the baseline across different **(a)** model width and **(b)** model depth.

These findings demonstrate that wider models allow for a significantly expanded window of opportunity to leverage N-gram Embedding effectively. Consequently, Figure 6 leads to the following conclusion: for a fixed number of layers,

Summary: Increasing model width confers a greater advantage to N-gram Embedding.

3.3.2 Diminishing Returns in Deeper Models

We now investigate the impact of model depth on the efficacy of N-gram Embedding scaling. For pre-normalization architectures, the contribution of N-gram Embedding through the identity connection (residual branch) inherently diminishes as network depth increases, as the signal propagating through skip connections carries less direct information from earlier layers (also shown in Figure 5).

To probe this hypothesis, we conducted scaling experiments using deeper architectures, building upon our 1.3B activated parameter configuration. Specifically, we trained models with 20 and 40 layers while meticulously maintaining a consistent relative proportion of N-gram Embedding parameters (50% of the total parameters) across all tested depths.

Figure 7b presents a clear comparison of the performance gap between N-gram Embedding and the MoE baseline across these varying depths. A striking observation emerges: as model depth surpasses 20 layers, the performance advantage of N-gram Embedding over the baseline experiences a pronounced contraction. This trend stands in contrast to the effect of increasing model width, as illustrated in Figure 7a, where the performance gap demonstrably widens.

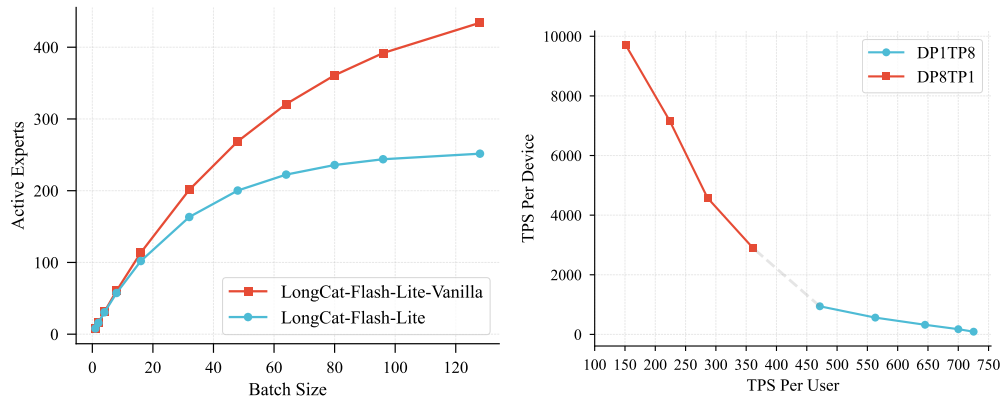
Summary: Increasing model depth diminishes the relative advantage of N-gram Embedding.

Note that the majority of current practical language models typically operate below 40 shortcut layers (equivalent to 80 conventional layers). Given our finding that increased width consistently amplifies N-gram Embedding's advantage, and its robust performance even at 40 layers, scaling n-gram embeddings up within these common architectural depths is likely to yield even greater performance gains.

4 Efficient Inference

4.1 Reduction of MoE Activation Parameters

The N-gram Embedding mechanism effectively redistributes parameters from the MoE layers to the embedding space. This architectural transformation maintains the total model parameters while reducing the number of activated parameters within MoE layers—particularly advantageous in memory I/O-bound decoding scenarios with large token



(a) (b)

Figure 8: **(a)** Number of activated experts in LongCat-Flash-Lite versus LongCat-Flash-Lite-Vanilla across varying batch sizes. **(b)** LongCat-Flash-Lite decoding performance on 8xH800-80G with ISL=4K and OSL=1K. The middle segment is for visual continuity. The model information of LongCat-Flash-Lite is described in Section 6.

counts. Moreover, the increased size of the embedding layer does not penalize latency, as the computational cost of embedding lookups scales with the number of input tokens rather than the total number of embedding parameters.

To fully capitalize on the efficiency gains from reduced active parameters, it is crucial to maximize hardware utilization through a large batch size (as shown in Figure 8a). This requirement creates a natural synergy with speculative decoding. Multi-step speculative decoding effectively expands the "effective batch size", thereby converting the theoretical advantage of parameter sparsity into tangible inference speedups.

4.2 Optimized Embedding Lookup

Although reallocating parameters from experts to N-gram Embedding effectively reduces memory I/O for MoE layers, it introduces additional overhead in terms of I/O, computation, and communication compared to a standard embedding layer. Minimizing the latency and resource consumption of N-gram Embedding is therefore critical for overall system efficiency. Furthermore, the dynamic and complex scheduling mechanisms inherent in modern inference frameworks make it difficult to pre-determine the exact token sequences for the forward pass, which complicates the optimization of N-gram embedding lookups.

To address these challenges, we introduce the *N-gram Cache*, a specialized caching mechanism inspired by the design principles of the KV cache. We implement custom CUDA kernels to manage N-gram IDs directly on the device, facilitating low-overhead synchronization with the intricate scheduling logic of various inference optimization techniques. This design significantly enhances the computational efficiency of N-gram embeddings.

In speculative decoding scenarios, where the draft model typically operates with fewer layers and substantially lower latency, the overhead of N-gram Embedding becomes relatively more pronounced. To mitigate this, we propose two complementary optimization strategies: (1) employing a conventional embedding layer for the draft model to bypass the more computationally expensive n-gram lookup; and (2) caching n-gram embeddings during the drafting phase to eliminate redundant computations during the subsequent verification step. These optimizations collectively reduce latency and improve throughput in speculative inference settings.

4.3 Rethinking N-gram Embedding Optimization: The Role of Speculative Decoding

Beyond hardware efficiency, we posit that the N-gram Embedding structure inherently encodes rich local context and token co-occurrence information, offering unexplored synergies with speculative decoding. We identify two promising directions where the semantic richness of N-gram Embedding could potentially be leveraged to further accelerate inference.

N-gram Embedding based drafting: Since the N-gram Embedding aggregates information from the preceding N-1 tokens, it implicitly captures short-range dependencies. We are currently exploring architectures to repurpose the N-gram embedding as an ultra-fast draft model. While a primary candidate involves attaching a lightweight linear

projection directly to the N-gram Embedding outputs, we are investigating a broader design space to fully exploit the captured local context for efficient token prediction.

Early rejection: The N-gram Embedding representation could also serve as a semantic consistency check (or confidence estimator) for tokens generated by external draft models. Draft tokens that result in low-probability match under N-gram Embedding might be "early-rejected" before entering the expensive verification phase of the target model. Theoretically, this pruning strategy would reduce the workload of the verification step, offering a pathway to further optimize end-to-end latency.

5 Integration with Per-Layer Embedding

As mentioned in Section 1, Per-Layer Embedding (PLE) is another way to scale parameters by allocating embedding parameters across layers. This section provides a direct comparison between N-gram Embedding and PLE, and introduces an attempt to integrate both approaches.

5.1 Per-Layer Embedding

PLE is applied in Google DeepMind [2024] and further studied in Sadhukhan et al. [2026]. PLE directly substitutes the output of up-projection matrix in the SwiGLU module with the embedding output, which is the most efficient method for injecting embedding information in our experiments. Let $x_i^{(l)}$ be the i-th input vector of the FFN module in layer l, the FFN output with PLE can be formalized as follows

$$\text{FFN}^{(l)}(x_i) = W_d^{(l)}(\text{SiLU}(W_g^{(l)}x_i^{(l)}) \odot E_0^{(l)}(t_i)) \tag{4}$$

where $W_d^{(l)}$ and $W_g^{(l)}$ denote the down-projection and gate-projection matrices of layer l respectively, and $E_0^{(l)}$ is the embedding table of layer l, with identical shape to the base embedding table in Eq. 1.

5.2 Per-Layer N-gram Embedding

Building upon PLE, we propose Per-Layer N-gram Embedding (PLNE), a novel extension that replaces the base embedding outputs with N-gram Embedding outputs at each layer, thereby enabling more flexible and targeted parameter scaling within the MoE framework. PLNE can be written as

$$\text{FFN}^{(l)}(x_i) = W_d^{(l)}(\text{SiLU}(W_g^{(l)}x_i^{(l)}) \odot e_i^{(l)}) \tag{5}$$

where $e_i^{(l)}$ is computed according to Eq. 3, with layer-specific embedding table and projection matrix.

5.3 Empirical Comparison

For both PLE and PLNE, embedding information is injected exclusively into the MLP within the dense sub-layer of each shortcut layer. Since each PLNE layer incorporates an n-gram vocabulary in addition to the base vocabulary, it introduces a larger number of parameters per layer compared to PLE. To avoid confounding factors related to layer positioning, we do not directly compare between PLE and PLNE under equivalent total parameter counts. Instead, we evaluate PLE and PLNE against their respective parameter-equivalent N-gram Embedding (NE) baselines, as illustrated in Figure 9.

Figure 9 reveals that PLE underperforms relative to N-gram Embedding, whereas PLNE yields marginal improvements over NE. We attribute the former to the superior learning efficiency of N-gram Embedding compared to standard embeddings. Consequently, we focused our scaling analysis on PLNE. However, in subsequent experiments involving increased model width or depth, PLNE failed to exhibit a consistent advantage, performing on par with NE in most scenarios. Given that PLNE inherently increases activated parameters (due to the addition of a substantial projection matrix in each layer), we opted not to adopt PLNE for our larger-scale experiments. Nonetheless, this approach merits further investigation—specifically regarding the optimal allocation of embedding parameters across layers, such as determining whether to concentrate them in a few specific layers or distribute them uniformly throughout the network.

6 LongCat-Flash-Lite

Leveraging the insights from our previous analysis, we introduce LongCat-Flash-Lite, a model trained from scratch with integrated N-gram Embedding. LongCat-Flash-Lite undergoes a complete pipeline of pre-training, mid-training, and supervised finetuning, and demonstrates highly competitive performance for its scale.

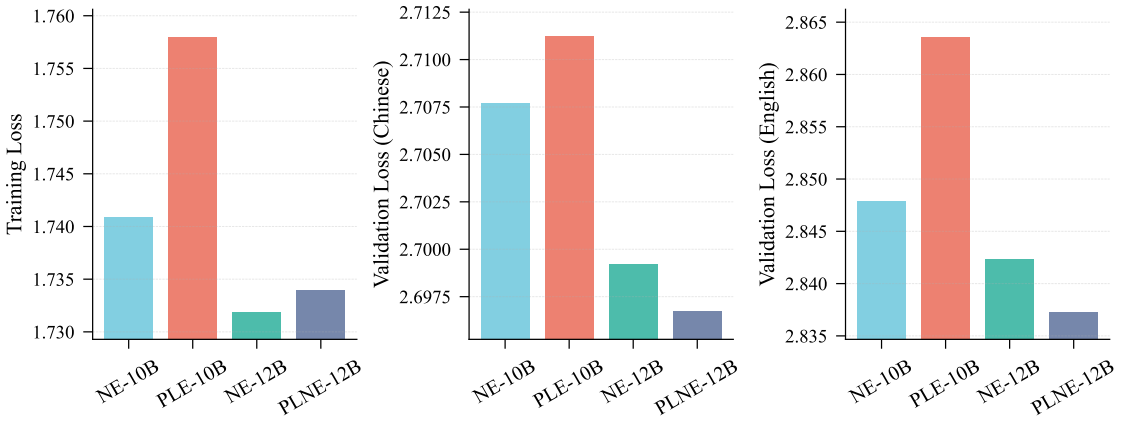


Figure 9: Loss comparison of N-gram Embedding (NE), PLE, and PLNE under the 790M activated-parameter setting. Note that PLE and PLNE are compared against NE at two distinct parameter scales.

6.1 Model Information

Architecture LongCat-Flash-Lite adopts the same architecture as Longcat-Flash [Meituan, 2025], with a total of 14 shortcut layers. It has 68.5 billion total parameters and dynamically activates between 2.9B and 4.5B parameters per token due to the zero-experts. In each shortcut layer, the MoE module consists of 256 FFN experts and 128 zero-experts, and each token selects 12 experts. For embedding module, LongCat-Flash-Lite includes 31.4B N-gram Embedding parameters, accounting for 46% of the total.

Training Data LongCat-Flash-Lite follows the same data recipe with LongCat-Flash-Chat [Meituan, 2025]. It is first pre-trained on 11T tokens with a sequence length of 8k, followed by 1.5T tokens of mid-training during which the sequence length is extended to 128k, and is finally trained on SFT data. To support extended context, we implement YARN [Peng et al., 2023] during the 32k sequence length training stage, enabling LongCat-Flash-Lite to handle sequences up to 256k tokens.

Baseline without N-gram Embedding We train an MoE baseline with exactly the same parameters as LongCat-Flash-Lite (referred to as LongCat-Flash-Lite-Vanilla) by converting all N-gram Embedding parameters into additional experts. Both models undergo identical training strategy and data recipe.

6.2 Base Model Evaluation

Throughout training, LongCat-Flash-Lite consistently achieves lower training loss compared to LongCat-Flash-Lite-Vanilla, as illustrated in Figure 10. To assess downstream performance, we evaluate both models on benchmarks spanning three core capability domains:

- **General Tasks:** MMLU [Hendrycks et al., 2021], MMLU-Pro [Wang et al., 2024], C-Eval [Huang et al., 2023], and CMMLU [Li et al., 2023].
- **Reasoning Tasks:** BBH [Suzgun et al., 2023], GPQA [M-A-P Team, ByteDance., 2025], DROP [Dua et al., 2019] and GSM8K [Cobbe et al., 2021].
- **Coding Tasks:** HumanEval+ [Liu et al., 2024], MultiPL-E [Cassano et al., 2022], and BigCodeBench [Zhuo et al., 2025].

As detailed in Table 1, LongCat-Flash-Lite demonstrates substantial performance improvements over LongCat-Flash-Lite-Vanilla across the majority of benchmarks in all three domains. These findings validate our earlier analysis: when sparsity reaches sufficient levels, strategically scaling total parameters through N-gram Embedding—while maintaining an optimal proportion of embedding parameters—consistently outperforms approaches that merely increase expert numbers.

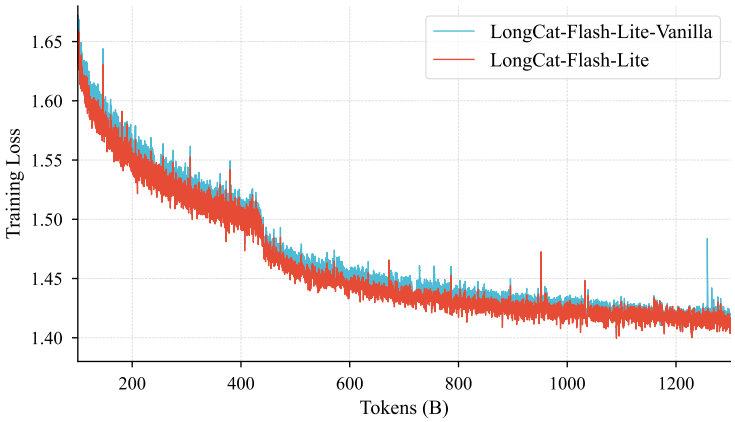


Figure 10: Smoothed training loss curves of LongCat-Flash-Lite and LongCat-Flash-Lite-Vanilla. The loss drop at 420B tokens coincides with the batch size increases.

Table 1: Comparison of base model between LongCat-Flash-Lite and LongCat-Flash-Lite-Vanilla.

	BenchMark	LongCat-Flash-Lite-Vanilla@1.3T	LongCat-Flash-Lite @1.3T
General	MMLU	**64.81**	64.01
	MMLU-pro	34.43	**35.89**
	CEval	64.09	**67.21**
	CMMLU	67.08	**69.55**
Reasoning	BBH	38.54	**43.67**
	GPQA	25.37	**29.66**
	DROP	47.92	**52.43**
	GSM8K	50.00	**50.50**
Coding	HumanEval+	28.66	**31.10**
	MultiPL-E	**30.20**	30.03
	BigCodeBench	33.42	**36.05**

6.3 Chat Model Evaluation

The evaluation of the chat model covers several core capabilities: agentic tool use tasks, agentic coding tasks, general domain tasks and mathematical reasoning tasks. The benchmarks used for assessment include:

- **Agentic Tool Use Tasks:** τ^2 Bench [Barres et al., 2025], Vita Bench [He et al., 2025]. For τ^2-Bench, we use a revised version[3] to perform our evaluation because the original version contains noisy data.
- **Agentic Coding Tasks:** SWE-Bench [Jimenez et al., 2023], TerminalBench [Merrill et al., 2026], SWE-Bench Multiligual [Yang et al., 2025], and PRDBench [Fu et al., 2025].
- **General Domain Tasks:** GPQA-Diamond [Rein et al., 2024], MMLU [Hendrycks et al., 2021], MMLU-Pro [Wang et al., 2024], C-Eval [Huang et al., 2023], and CMMLU [Li et al., 2023].
- **Mathematical Reasoning Tasks:** MATH500 [Lightman et al., 2023], AIME24 [MAA, 2024], AIME25 [MAA, 2025].

Table 2 presents the comprehensive evaluation results of LongCat-Flash-Lite across various benchmark categories, along with comparisons with Qwen3-Next-80B-A3B-Instruct, Gemini 2.5 Flash-Lite[4], and Kimi-Linear-48B-A3B. LongCat-Flash-Lite demonstrates exceptional parameter efficiency and competitive performance across core capability dimensions.

Agentic Tool Use. LongCat-Flash-Lite excels in agentic tool use tasks, establishing a clear lead over all comparison models. In the τ^2-Bench benchmark, it achieves the highest scores across all three sub-scenarios: Telecom (72.8),

[3] https://github.com/AGI-Eval-Official/tau2-bench-revised
[4] Gemini 2.5 Flash-Lite Preview 09-2025

Table 2: Comparison between LongCat-Flash-Lite and other models. Values marked with * are sourced from public reports.

Benchmark	Kimi-Linear-48B-A3B	Qwen3-Next-80B-A3B-Instruct	Gemini 2.5 Flash-Lite	LongCat-Flash-Lite
Architecture	MoE	MoE	-	MoE + NE
# Total Params	48B	80B	-	68.5B
# Activated Params	3B	3B	-	2.9B∼4.5B
Agentic Tool Use				
Tau2-Airline(avg@8)	44.00	45.5*	35.00	58.00
Tau2-Retail(avg@8)	18.86	57.3*	37.50	73.10
Tau2-Telecom(avg@8)	15.68	13.2*	21.93	72.80
VitaBench(avg@4)	-	5.80	4.50	7.00
Agentic Coding				
SWE-Bench(acc)	32.80	37.60	41.3*	54.40
TerminalBench(acc)	20.00	15.19	20.00	33.75
SWE-Bench Multiligual	37.20	31.30	-	38.10
PRDBench	-	15.36	-	39.63
General Domains				
GPQA-Diamond(avg@16)	69.89	74.33	70.20*	66.78
MMLU(acc)	79.91	89.28	84.68	85.52
MMLU-Pro(acc)	67.22	82.93	78.95	78.29
CEval(acc)	78.48	90.91	75.16	86.55
CMMLU(acc)	76.26	86.50	72.06	82.48
Mathematical Reasoning				
MATH500(acc)	94.20	98.00	95.20	96.80
AIME24(avg@32)	70.52	81.35	63.33	72.19
AIME25(avg@32)	59.58	68.44	50.1*	63.23

Retail (73.1), and Airline (58.0). Notably, in the Telecom scenario, its score significantly outperforms Gemini 2.5 Flash-Lite and Kimi-Linear-48B-A3B. This highlights its superior ability to handle complex dependencies on tools and domain-specific task execution. In VitaBench, it achieves a score of 7.00, outperforming Qwen3-Next-80B-A3B-Instruct (5.80), and Gemini 2.5 Flash-Lite (4.50). This leading score underscores LongCat-Flash-Lite's superior ability to handle complex, real-world task workflows via tool integration in practical business scenarios.

Agentic Coding. In coding-related tasks, LongCat-Flash-Lite demonstrates remarkable practical problem-solving capabilities. In SWE-Bench, it achieves an accuracy of 54.4, outperforming all baselines—surpassing Qwen3-Next-80B-A3B-Instruct (37.6), Gemini 2.5 Flash-Lite (41.3), and Kimi-Linear-48B-A3B (32.8) by a significant margin. This indicates its proficiency in solving real-world software engineering issues, including bug fixes and feature implementation. In TerminalBench, which evaluates terminal command execution competence, LongCat-Flash-Lite secures a leading score of 33.75, far exceeding Qwen3-Next-80B-A3B-Instruct (15.19), Gemini 2.5 Flash-Lite (20.0) and Kimi-Linear-48B-A3B (20.0), reflecting its robust ability to understand and execute terminal-related instructions critical for developer-centric agentic applications. Additionally, in SWE-Bench Multilingual—a benchmark designed to measure cross-language programming generalization across diverse software ecosystems—LongCat-Flash-Lite achieves a strong accuracy of 38.10, outperforming Qwen3-Next-80B-A3B-Instruct (31.3) and Kimi-Linear-48B-A3B (37.2), thus demonstrating its reliable adaptability to multi-language development scenarios. In PRDBench, LongCat-Flash-Lite achieves a score of 39.63, significantly outperforming Qwen3-Next-80B-A3B-Instruct (15.36). We observe that our model can autonomously write unit tests to verify its development, producing higher-quality code repositories.

General Domains. LongCat-Flash-Lite delivers balanced and competitive performance in general domain knowledge tasks. On MMLU, it scores 85.52, which is comparable to Gemini 2.5 Flash-Lite (84.68) and Kimi-Linear-48B-A3B (79.91), and only slightly lower than Qwen3-Next-80B-A3B-Instruct (89.28). In Chinese-specific benchmarks (CEval and CMMLU), it achieves 86.55 and 82.48 respectively, performing particularly well against Kimi-Linear-48B-A3B (78.48 and 76.26) and Gemini 2.5 Flash-Lite (75.16 and 72.06). On GPQA-Diamond, it scores 66.78, maintaining competitiveness within the benchmark's performance range. For MMLU-Pro, it achieves 78.29, demonstrating solid performance in handling more challenging multi-task language understanding questions.

Mathematical Reasoning. LongCat-Flash-Lite exhibits strong mathematical reasoning capabilities across both basic and advanced tasks. On MATH500, it achieves an accuracy of 96.80, which is close to Qwen3-Next-80B-A3B-Instruct (98.00), and outperforms Gemini 2.5 Flash-Lite (95.20). In advanced mathematical competition benchmarks, it delivers impressive results: AIME24 (72.19) and AIME25 (63.23). These scores surpass Kimi-Linear-48B-A3B (70.52 and

59.58) and Gemini 2.5 Flash-Lite (63.33 and 50.1), highlighting its ability to handle complex, multi-step mathematical deduction.

6.4 Fast Inference with Optimized Kernels

As discussed in Section 4, the extreme activation sparsity of this model necessitates a large effective batch size to fully saturate GPU memory bandwidth. To achieve this, we deploy the model using "Eagle3" [Li et al., 2025] with a "3-step speculative decoding strategy". Similarly to Qian et al. [2025] and Meituan [2025], we adopt wide EP (Expert Parallel) and SBO (Single Batch Overlap) to accelerate inference speed. While the above optimizations successfully expands the effective batch size, the model's lightweight nature shifts the bottleneck towards kernel launch overheads, making it challenging to maintain high GPU occupancy. To address this and minimize end-to-end latency, we implement the following system-level optimizations:

Kernel Optimization

- Kernel Fusion: We apply extensive kernel fusion to reduce execution overhead and memory traffic. Specifically, all intra-TP-group communication operations are fused with subsequent fine-grained kernels (e.g., `AllReduce` + `Residual Add` + `RMSNorm`, `AllGather` + `Q-Norm` + `KV-Norm`, and `ReduceScatter` + `RMSNorm` + `Hidden State Combine`). For the quantized model, we integrate every activation quantization step into existing operators, including the aforementioned communication-fusion kernels and the `SwiGLU` component. Additionally, the processing of router logits (`Softmax` + `TopK` + `router scaling`) and zero-expert selection is consolidated into a single unified kernel.
- Optimized Attention Combine: We employ a splitkv-and-combine strategy during decoding phase. When the number of KV splits is high, the combine operation can incur significant latency, sometimes comparable to the computation itself. By optimizing the combine kernel, we effectively reduce its latency by 50%.

PDL (Programmatic Dependent Launch) We utilize PDL [NVIDIA, 2026] to allow dependent kernels to overlap their execution by triggering early launches. This mechanism not only eliminates the gaps between consecutive kernels but also improves SM utilization.

Building upon these optimizations, we achieve the exceptional inference performance illustrated in Figure 8b.

7 Conclusions

In this technical report, we presented a comprehensive study on the scalability and efficiency of embedding scaling in LLMs. Through systematic analysis of architectural constraints and comparative scaling laws, we demonstrated that scaling embeddings yields a superior Pareto frontier compared to increasing expert numbers in specific regimes, while our proposed system optimizations, including the N-gram Cache and synchronized kernels, effectively resolve associated I/O bottlenecks. Validating these findings, we introduced LongCat-Flash-Lite, a 68.5B MoE model with over 30B N-gram Embeddingparameters, which not only outperforms parameter-equivalent MoE baselines but also exhibits competitive performance in agentic and coding tasks, thereby establishing a robust and efficient framework for future model scaling.

8 Acknowledgement

We extend our sincere gratitude to both the infrastructure team and evaluation team for their invaluable support and constructive feedback throughout this project. The primary contributors from these teams include:

Linsen Guo	Lin Qiu	Xiao Liu	Yaoming Zhu
Mengxia Shen	Zijian Zhang	Xiaoyu Li	Chao Zhang
Yunke Zhao	Dengchang Zhao	Yifan Lu	

References

Hongzhi Huang, Defa Zhu, Banggu Wu, Yutao Zeng, Ya Wang, Qiyang Min, and Xun Zhou. Over-tokenized transformer: Vocabulary is generally worth scaling, 2025. URL https://arxiv.org/abs/2501.16975.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=qrwe7XHTmYb.

Samira Abnar, Harshay Shah, Dan Busbridge, Alaaeldin El-Nouby, Joshua M. Susskind, and Vimal Thilak. Parameters vs flops: Scaling laws for optimal sparsity for mixture-of-experts language models. In *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=l9FVZ7NXmm.

Chaofan Tao, Qian Liu, Longxu Dou, Niklas Muennighoff, Zhongwei Wan, Ping Luo, Min Lin, and Ngai Wong. Scaling laws with vocabulary: Larger models deserve larger vocabularies. *Advances in Neural Information Processing Systems*, 37:114147–114179, 2024.

Google DeepMind. Gemma 3n. https://deepmind.google/models/gemma/gemma-3n/, 2025. Accessed: 2026-01-16.

Ranajoy Sadhukhan, Sheng Cao, Harry Dong, Changsheng Zhao, Attiano Purpura-Pontoniere, Yuandong Tian, Zechun Liu, and Beidi Chen. Stem: Scaling transformers with embedding modules, 2026. URL https://arxiv.org/abs/2601.10639.

bcml labs. ROSA+: RWKV's ROSA implementation with fallback statistical predictor. https://github.com/bcml-labs/rosa-plus, 2025. Accessed: 2026-01-23.

W. Ronny Huang, Tara N. Sainath, Cal Peyser, Shankar Kumar, David Rybach, and Trevor Strohman. Lookup-table recurrent language models for long tail speech recognition. In Hynek Hermansky, Honza Cernocký, Lukás Burget, Lori Lamel, Odette Scharenborg, and Petr Motlícek, editors, *22nd Annual Conference of the International Speech Communication Association, Interspeech 2021, Brno, Czechia, August 30 - September 3, 2021*, pages 2002–2006. ISCA, 2021. doi:10.21437/INTERSPEECH.2021-340. URL https://doi.org/10.21437/Interspeech.2021-340.

Jonathan H. Clark, Dan Garrette, Iulia Turc, and John Wieting. Canine: Pre-training an efficient tokenization-free encoder for language representation. *Transactions of the Association for Computational Linguistics*, 10:73–91, 2022. doi:10.1162/tacl_a_00448. URL https://aclanthology.org/2022.tacl-1.5/.

Artidoro Pagnoni, Ramakanth Pasunuru, Pedro Rodriguez, John Nguyen, Benjamin Muller, Margaret Li, Chunting Zhou, Lili Yu, Jason E Weston, Luke Zettlemoyer, Gargi Ghosh, Mike Lewis, Ari Holtzman, and Srini Iyer. Byte latent transformer: Patches scale better than tokens. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9238–9258, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi:10.18653/v1/2025.acl-long.453. URL https://aclanthology.org/2025.acl-long.453/.

Xin Cheng, Wangding Zeng, Damai Dai, Qinyu Chen, Bingxuan Wang, Zhenda Xie, Kezhao Huang, Xingkai Yu, Zhewen Hao, Yukun Li, et al. Conditional memory via scalable lookup: A new axis of sparsity for large language models. *arXiv preprint arXiv:2601.07372*, 2026.

Meituan. Longcat-flash technical report, 2025. URL https://arxiv.org/abs/2509.01322.

Sho Takase, Shun Kiyono, Sosuke Kobayashi, and Jun Suzuki. Spike no more: Stabilizing the pre-training of large language models, 2025. URL https://arxiv.org/abs/2312.16903.

Google DeepMind. Gemma 3n documentation. https://ai.google.dev/gemma/docs/gemma-3n, 2024. Accessed: 2025-09-04.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. Yarn: Efficient context window extension of large language models. *ArXiv*, abs/2309.00071, 2023. URL https://api.semanticscholar.org/CorpusID:261493986.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2021.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. *arXiv preprint arXiv:2406.01574*, 2024.

Yuzhen Huang, Yuzhuo Bai, Zhihao Zhu, Junlei Zhang, Jinghan Zhang, Tangjun Su, Junteng Liu, Chuancheng Lv, Yikai Zhang, Jiayi Lei, Yao Fu, Maosong Sun, and Junxian He. C-Eval: A multi-level multi-discipline chinese evaluation suite for foundation models. In *Advances in Neural Information Processing Systems*, 2023.

Haonan Li, Yixuan Zhang, Fajri Koto, Yifei Yang, Hai Zhao, Yeyun Gong, Nan Duan, and Timothy Baldwin. CMMLU: Measuring massive multitask language understanding in chinese. *arXiv preprint arXiv:2306.09212*, 2023.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, and Jason Wei. Challenging BIG-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, 2023.

M-A-P Team, ByteDance. SuperGPQA: Scaling LLM evaluation across 285 graduate disciplines. *arXiv preprint arXiv:2502.14739*, 2025.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. *arXiv preprint arXiv:1903.00161*, 2019.

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

Jiawei Liu, Songrun Xie, Junhao Wang, Yuxiang Wei, Yifeng Ding, and Lingming Zhang. Evaluating language models for efficient code generation. *arXiv preprint arXiv:2408.06450*, 2024.

Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q Feldman, Arjun Guha, Michael Greenberg, and Abhinav Jangda. MultiPL-E: A scalable and extensible approach to benchmarking neural code generation. *arXiv preprint arXiv:2208.08227*, 2022.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen Gong, Thong Hoang, Armel Randy Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, Prateek Yadav, Naman Jain, Alex Gu, Zhoujun Cheng, Jiawei Liu, Qian Liu, Zijian Wang, Binyuan Hui, Niklas Muennighoff, David Lo, Daniel Fried, Xiaoning Du, Harm de Vries, and Leandro Von Werra. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions, 2025. URL https://arxiv.org/abs/2406.15877.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. *CoRR*, abs/2506.07982, 2025.

Wei He, Yueqing Sun, Hongyan Hao, Xueyuan Hao, Zhikang Xia, Qi Gu, Chengcheng Han, Dengchang Zhao, Hui Su, Kefeng Zhang, Man Gao, Xi Su, Xiaodong Cai, Xunliang Cai, Yu Yang, and Yunke Zhao. Vitabench: Benchmarking llm agents with versatile interactive tasks in real-world applications. *arXiv preprint arXiv:2509.26490*, 2025.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Jenia Jitsev, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Jesse Hu, Christopher Michael Rytting, Ryan Marten, Yixin Wang, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces, 2026. URL https://arxiv.org/abs/2601.11868.

John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. Swe-smith: Scaling data for software engineering agents, 2025. URL https://arxiv.org/abs/2504.21798.

Lingyue Fu, Bolun Zhang, Hao Guan, Yaoming Zhu, Lin Qiu, Weiwen Liu, Xuezhi Cao, Xunliang Cai, Weinan Zhang, and Yong Yu. Automatically benchmarking llm code agents through agent-driven annotation and evaluation. *arXiv preprint arXiv:2510.24358*, 2025.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2023.

MAA. Aime 2024, 2024. URL https://maa.org/math-competitions/american-invitational-mathematics-examination-aime.

MAA. Aime 2025, 2025. URL https://artofproblemsolving.com/wiki/index.php/AIMEProblemsandSolutions.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-3: Scaling up inference acceleration of large language models via training-time test, 2025. URL https://arxiv.org/abs/2503.01840.

Yulei Qian, Fengcun Li, Xiangyang Ji, Xiaoyu Zhao, Jianchao Tan, Kefeng Zhang, and Xunliang Cai. Eps-moe: Expert pipeline scheduler for cost-efficient moe inference, 2025. URL https://arxiv.org/abs/2410.12247.

NVIDIA. Programmatic dependent launch and synchronization. https://docs.nvidia.com/cuda/cuda-programming-guide/04-special-topics/programmatic-dependent-launch.html, 2026. Accessed: 2026-01-27.