

Precision Under Pressure: Pushing the Boundaries of the Accuracy-Efficiency Frontier in Question Answering with Mixture-of-Depths

Stanford CS224N Custom Project

Soha Sultan, Haoyue Yang, Jan Kopanski
Department of Computer Science
Stanford University
{sohas, haoyuey, kopanski}@stanford.edu

Abstract

Standard transformer architectures encounter inference bottlenecks by applying uniform computational depth to tokens of varying complexity. While Mixture-of-Depths (MoD) addresses this inefficiency via prohibitive full-scale pre-training, post-hoc “MoDification” methods applied to pre-trained models disrupt residual streams and attention pipelines, causing representational collapse. We overcome these limitations with a novel joint LoRA-MoD architecture that co-optimizes routing thresholds and task-specific adapters, isolating sparsity exclusively to Feed-Forward (MLP) modules to preserve hardware-efficient attention. Evaluated on a latency-constrained *Jeopardy!* framework (SmolLM2-135M), our approach establishes a compute-accuracy Pareto frontier. Our optimal configuration reduces inference FLOPs by 49% with minimal accuracy degradation. Furthermore, we discovered that moderate sparsity acts as a regularizer: bypassing MLPs for simple syntax reduces FLOPs by 24% while actively outperforming the dense baseline’s factual accuracy.

1 Key Information to include

- Mentor: Julie Kallini
- External Collaborators: Chris Welty (Google Research)
- Sharing project: No
- Late Days (if you are using any):

Team member	Late days remaining
Soha Sultan	4
Haoyue Yang	1
Jan Kopanski	4
Total pooled	9
Team size	3
Late days applied for the report	3

2 Introduction

Real-time Large Language Model (LLM) applications, such as Google AI Overviews and Gemini’s fast mode, demand strict sub-second inference for an optimal user experience. However, achieving this efficiency is structurally challenging: traditional Transformers allocate the same computational budget to every token, inefficiently expending massive FLOPs on easy, predictable syntax rather than dynamically focusing compute on complex factual extraction. Optimizing computational efficiency while maintaining high accuracy remains a notoriously difficult architectural trade-off. To rigorously

evaluate performance under these dual pressures, we introduce a latency-constrained question-answering framework modeled on the *Jeopardy!*[1] game. In this environment, inference latency acts as a competitive "buzzer," while factual hallucinations incur strict penalties. To aggressively reduce latency without sacrificing accuracy, we implement a novel joint LoRA[2]-MoD [3] architecture that co-optimizes routing thresholds and task-specific adapters, isolating sparsity strictly to the massive feed forward blocks. On a 135M-parameter baseline, this framework yields a notable Pareto frontier between compute and accuracy. Our optimal configuration reduces total inference FLOPs by 48.62% with only 0.67% drop in RoBERTa[4] accuracy score. Furthermore, structural sparsity acts as a regularizer: skipping predictable syntax reduces FLOPs by 24.00% while actively outperforming the dense baseline’s accuracy, proving that intelligent routing surpasses static computation.

3 Related Work

Inference Acceleration and Depth Reduction

Traditional Transformers suffer from a uniform compute fallacy, allocating identical FLOPs to predictable syntax and complex factual extraction. While early exiting methods [5, 6] propose dynamic depth, they often prematurely bypass terminal layers essential for the precise vocabulary projection required in *Jeopardy!*-style retrieval. Similarly, static pruning [7] and input-agnostic sparsity, such as Gamma-MoD [8], permanently strip the representational depth needed to resolve nuanced clues. Acceleration strategies like speculative decoding [9, 10] and LayerSkip [11] struggle in fact-dense tasks, suffering severe rejection penalties because the high-entropy proper nouns and dates inherent to *Jeopardy!* answers frequently invalidate drafted tokens. Token-reduction methods like MrT5 [12] remain limited to bidirectional encoders, rendering them incompatible with modern causal decoders.

Dynamic Compute via Mixture-of-Depths

The Mixture-of-Depths (MoD) [3] framework offers a surgical alternative, utilizing dynamic token-level routing to allow easily resolved tokens to bypass computational blocks via residual connections. However, standard MoD relies on Top- k routing, which requires sorting token probabilities across an entire sequence. This introduces a hardware-inefficient sequential bottleneck that severely hinders autoregressive wall-clock inference. To resolve this, MoDification [13] introduces a highly parallelizable Threshold- p mechanism, evaluating tokens independently against a fixed probability threshold rather than relying on a global sequence-sort. While MoDification successfully resolves these structural routing bottlenecks, it remains a purely architectural optimization lacking a mechanism for task-specific factual alignment. Naively fine-tuning this sparse architecture for rigorous QA formats risks catastrophic forgetting of the broad, pre-trained world knowledge that is essential for *Jeopardy!* retrieval. To bridge this gap, we propose a joint LoRA-MoD architecture, detailed below, to simultaneously achieve dynamic inference acceleration and robust factual retention.

4 Approach

4.1 Architecture: Joint LoRA-MoD

We transform the dense SmoLLM2-135M[14] into a compute-efficient retriever using a joint Mixture-of-Depths (MoD) and Low-Rank Adaptation (LoRA) framework. While our routing mechanism employs the parallelizable Threshold- p strategy from MoDification[13], we fundamentally restructure its layer-skipping logic (Table 1). MoDification primarily focuses on mitigating the sequence-level sorting bottlenecks that constrain Top- k routing during long-context generation. Conversely,

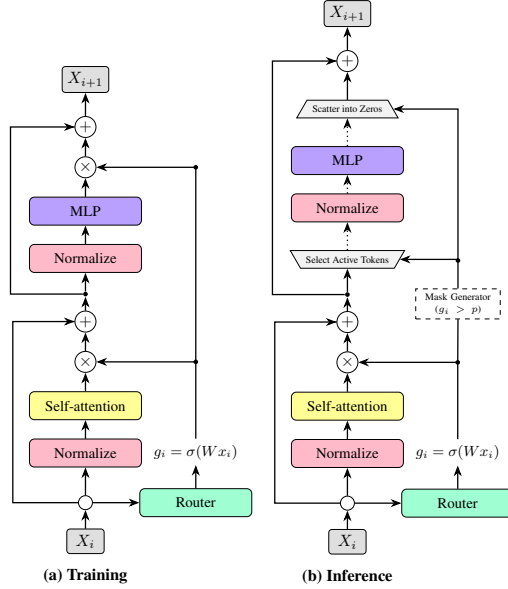


Figure 1: Hybrid Mixture-of-Depths Architecture. Soft gating in training enables differentiable gradient flow. Threshold- p masking at inference enables FLOP savings via dynamic token skipping.

Table 1: Architectural Comparison of Mixture of Depths (MoD) Variants

Feature	Original MoD	MoDification	Our Implementation
Routing	Top- k selection (Fixed compute budget)	Threshold- p (Dynamic routing)	Threshold- p (Dynamic routing)
Scaling	MLP output only	Attention & MLP outputs	Attention & MLP outputs
Training	Sparse; tokens dropped based on k	Sparse; tokens dropped via threshold	Dense MLP with Soft-gated Attention
Inference (Attn)	Sparse (Fixed-capacity)	Sparse (Custom MoE kernels)	Dense (HuggingFace / RoPE constraints)
Inference (MLP)	Sparse (Fixed-capacity)	Sparse (Custom MoE kernels)	Sparse (Slicing & Scattering)

Jeopardy! relies on inherently short clue-response pairs where sequence-length penalties and attention costs are relatively negligible. Instead, the massive Multi-Layer Perceptron (MLP) blocks account for the vast majority of the inference FLOPs, imposing a severe, constant computational penalty. Consequently, as presented in Figure 1, our computational graph explicitly bifurcates: we preserve fully dense Self-Attention to guarantee rich contextual grounding, isolating our strict computational savings entirely to the MLPs via hard physical token slicing.

To prevent catastrophic forgetting of pre-trained world knowledge essential for trivia retrieval, we freeze the base model and inject trainable LoRA rank-decomposition matrices exclusively into these active MLP paths. The routing logic and joint forward pass are governed by a lightweight linear projection over the hidden states. For a given token i , the router predicts a continuous probability $g_i \in [0, 1]$, which determines the discrete routing decision $f_i \in \{0, 1\}$ via the threshold p . During training continuous relaxation with soft-gating by g_i is applied to both Attention and MLP. At inference we maintain soft-gating for Attention but, whereas the MLP undergoes hard thresholding:

$$\begin{aligned}
g_i &= \sigma(W_{router}\mathbf{x}_i + b_{router}), & f_i &= \mathbb{I}(g_i > p) \\
\mathbf{y}_i &= g_i \cdot \text{Attention}(\text{Norm}(\mathbf{x}_i)) + \mathbf{x}_i \\
\mathbf{z}_i &= \begin{cases} g_i \cdot \text{MLP}_{LoRA}(\text{Norm}(\mathbf{y}_i)) + \mathbf{y}_i, & f_i = 1 \\ \mathbf{y}_i, & f_i = 0 \end{cases}
\end{aligned} \tag{1}$$

Here, σ denotes the sigmoid activation, $\mathbb{I}$ is the indicator function, and MLP_{LoRA} represents the feed-forward network augmented with task-specific LoRA adapters. This joint optimization forces the routers to learn a synergistic gating policy, activating the expensive MLPs and their associated LoRA weights only for complex, high-entropy factual tokens.

4.2 Sparsity-Constrained Optimization

Standard cross-entropy optimization naturally causes the routing mechanism to collapse toward maximum compute ($g_i \rightarrow 1$) to minimize perplexity. To enforce genuine sparsity during joint fine-tuning, we adopt the auxiliary penalty from MoDification[13]:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{LM}} + \alpha \sum_{j \in \mathcal{M}} (F_j \cdot \bar{G}_j) \tag{2}$$

where $\mathcal{L}_{LM}$ is the causal language modeling loss, $\mathcal{M}$ represents the set of MoD-enabled layers, F_j is the empirical fraction of routed tokens, and $\bar{G}_j$ is their mean probability. The hyperparameter α modulates the regularization strength. Finally, to prevent premature router collapse, we initialize the router bias with a continuous relaxation ($b_{router} = 2.0$), ensuring most tokens initially route through the LoRA adapters before the auxiliary penalty gracefully induces sparsity.

4.3 Baselines

To evaluate our LoRA-MoD architecture, we benchmark against a SmoLLM2-135M baseline ($r = 32, \alpha = 64$) post-trained using **Standard LoRA** on the *Jeopardy!* dataset. This single-control comparison strictly isolates the impact of our threshold-based compute dynamicism. By measuring both accuracy and latency against this dense LoRA variant, we quantify the efficiency-accuracy trade-off introduced by the MoDification framework. This ensures our evaluation specifically verifies whether the routing mechanism can effectively skip redundant computation while maintaining high-fidelity representational capacity.

4.4 Research Novelty

We present two primary contributions to the field of compute-efficient inference. First, we propose the novel joint LoRA-MoD architecture. This framework advances the MoD paradigm through a **joint calibration regime** that co-optimizes routers and LoRA adapters simultaneously, avoiding the representational collapse typical of post-hoc routing. Additionally, we employ **selective sublayer routing**, isolating sparsity strictly to the massive MLP modules. This architectural choice maximizes FLOP savings without sacrificing critical syntactic information captured by self-attention, nor disrupting the dense adjacency matrices required for stable, hardware-efficient pipelines.

Second, we establish a **Pareto frontier** between compute efficiency and factual accuracy for knowledge-intensive tasks. In mapping this frontier, we demonstrate **scale portability** by successfully adapting MoDification, originally validated for models exceeding 1.5 billion parameters, to the ultra-compact SmoLLM2-135M. Furthermore, we validate **task-specific routing** along this frontier, correlating factual entropy in *Jeopardy!* with dynamic compute-skipping thresholds to establish a reliable heuristic for token-level routing.

5 Experiments

5.1 Data

The *Jeopardy!* archive [1] serves as a rigorous benchmark, challenging models with diverse factual domains and complex wordplay (Table 2). We partitioned 548,364 clues across 42 seasons. To mitigate recency bias and ensure temporal out-of-distribution testing, we used seasons 1–20 and 22–40 for training (93.6%), and seasons 21 and 41 for validation (5.1%). The unseen season 42 (1.3%) was reserved for testing, crucially post-dating the baseline model’s pre-training cutoffs.

Table 2: Dataset Samples showcasing linguistic diversity.

Category	Clue	Response
Geography	Himalayan "land of the thunder dragon."	Bhutan
Rhyme Time	A heavy, well-fed feline.	Fat Cat
Before & After	Screen icon who represents the US legislative body.	M. Monroe House of Reps.

5.2 Evaluation Methods

We evaluate our models across three primary axes: factual retrieval accuracy, theoretical compute efficiency, and empirical inference latency.

Answer Match Accuracy Our evaluation prioritizes factual retrieval accuracy. We employ an ensemble of four metrics: (1) Exact Match (EM) with string-normalized equality, (2) PEDANTS semantic matching [15], (3) Transformer-based semantic answer equivalence via a fine-tuned RoBERTa-Large model [4], and (4) LLM-as-a-Judge with Gemini 3 Flash. While Gemini aligns best with human judgment, prohibitive API costs make local RoBERTa our primary evaluation metric. We maintain this multi-metric evaluation to counteract specific vulnerabilities observed during qualitative analysis; for example, PEDANTS heavily penalizes valid synonyms, while RoBERTa and Gemini occasionally produce anomalous accuracy spikes when grading malformed outputs.

Compute Efficiency To provide hardware-agnostic benchmarks, we calculate token-level FLOP reductions. For our MoD-based architecture, the compute cost per token relies on the hidden dimension (d_m) and intermediate expansion size (d_{ff}), yielding $6 \cdot d_m \cdot d_{ff}$ for the MLP and $4 \cdot d_m^2$ for attention. By aggregating skipped tokens relative to the maximum possible token-layer evaluations, we report three metrics: (1) Skipped Token Rate, fraction of tokens bypassing the MLP in MoD-router equipped layers; (2) MLP FLOPs Rate, savings relative to the total potential MLP computations; and (3) Total FLOPs Rate, comprehensive savings including the dense attention overhead.

Inference Latency We rigorously profile empirical wall-clock generation speed in milliseconds. To eliminate framework initialization overhead and stabilize CUDA graphs, the model executes a 2-iteration batch generation warmup prior to measurement. During evaluation, we synchronize CPU-GPU execution to ensure accurate wall time measurement. We report the average generation latency and standard deviation across 10 independent runs, applying the non-parametric Mann-Whitney U test to validate the statistical significance (p -value) of the hardware acceleration against the dense baseline.

5.3 Experimental Details

We selected initial dense baselines using Google Colab GPU (T4, L4) and prototyped our architectures on Modal utilizing NVIDIA T4 instances. For large-scale exploration, we provisioned an 8-node GCP cluster equipped with NVIDIA L4 GPUs, orchestrating a distributed training sweep via Weights & Biases. Our hyperparameter optimization proceeded in two sequential phases:

Phase 1 (Grid Search) We evaluated 44 models across varying learning rates ($10^{-3}, 10^{-4}, 10^{-5}$), sparsity weights ($\alpha \in \{0.1, 0.2, 0.3, 0.5, 0.6, 0.7, 1.0\}$), and layer-skipping frequencies (1, 2) using a 2000-step linear sparsity ramp. From this sweep, we selected four finalist candidates for further analysis, as detailed in the Results section.

Phase 2 (Threshold Optimization) We swept the dynamic routing probability threshold (p) from 0.1 to 1.0 (in 0.1 increments) on our most balanced candidate to precisely map the latency-accuracy cliff.

To eliminate framework-level tensor-allocation overhead during inference, we optimized our evaluation pipelines using `torch.compile()` for dynamic graph fusion. This bypassed tracer delays, enabling accurate, hardware-saturated latency benchmarking up to a batch size of 1024.

5.4 Results

We evaluate our joint LoRA-MoD architecture against the dense SmoLLM2-135M baseline across three axes: accuracy, compute efficiency, and latency. Figure 2 maps the efficiency-accuracy trade-off across our hyperparameter sweep, while Table 3 details the specific performance metrics of our selected representative configurations. In these configurations, `sw` denotes the sparsity weight (α in our loss formulation), and `freq` indicates the layer-skipping frequency (e.g., 1 for every layer, 2 for alternating layers).

Hyperparameter Consistency and Fair Baselines As illustrated in Figure 2, our model sweep reveals that a learning rate of $1e-3$ universally dominates the performance frontier for our sparse models. Crucially, our pure LoRA dense baseline also achieves its peak accuracy at this learning rate. This alignment confirms that our baseline is fully optimized, ensuring the observed MoD improvements reflect genuine architectural trade-offs rather than artifacts of sub-optimal hyperparameter tuning.

Sparsity as a Regularizer The upper-left quadrant of our ablation space proves that moderate sparsity acts as a functional enhancement. As detailed in Table 3, `mod_sw0.2_freq2` represents our most accurate configuration, actively outperforming the dense baseline in RoBERTa match (35.75% vs. 35.55%) while eliminating 24.00% of total inference FLOPs. This confirms our hypothesis: by successfully bypassing the MLP for predictable syntactic framing, the model filters out noise, focusing its active LoRA-augmented pathways strictly on complex factual extraction.

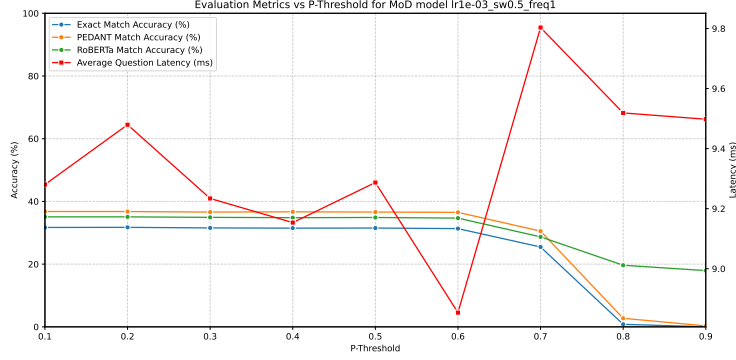


Figure 3: Accuracy and latency vs P-Threshold. The optimal inference-time P-Threshold (0.6) for latency minimization without the loss of accuracy is above the training-time P-Threshold (0.5).

6 Analysis

Bimodal Distribution and Depth Allocation. Aggregated skip profiles reveal a distinct **bimodal distribution** in budget utilization (see Figure 4). To preserve representational stability, a baseline compute floor is maintained with approximately **11%** of tokens skipping zero layers. Conversely, the router acts as a high-confidence classifier, aggressively pushing roughly **47%** of tokens to bypass 28–29 layers entirely, thereby offloading lightweight structural triggers early.

Furthermore, skip probability scales with depth, rising from **36% at L01** to a saturation point of **82% by L29** (Figure 5). This depth-bound decay indicates rapid representational maturation; once stable contextual attractors form in early layers, the model safely bypasses the vast majority of late-stage compute operations with negligible impact on accuracy.

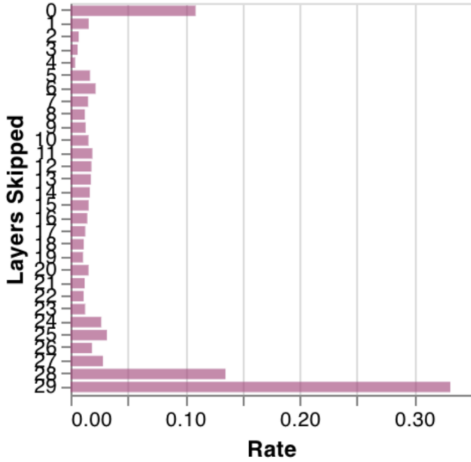


Figure 4: Percentage of tokens per layers skipped

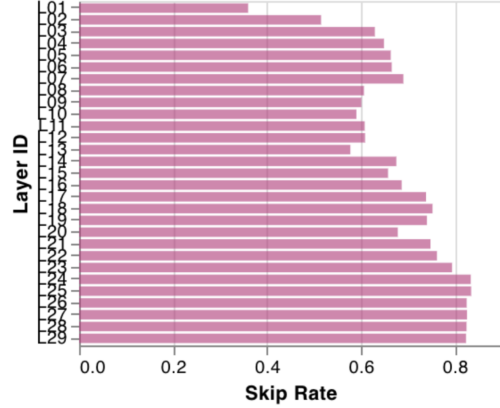


Figure 5: Skip Rate per Layer ID

Categorical Variance and Difficulty Stability. The MoD router exhibits high sensitivity to linguistic domains, with skip frequencies dispersing significantly across categories (Figure 6). Direct factual domains like **Science** and **World Geography** achieve the highest efficiency (skip rates ≈ 0.80). Conversely, categories like **Nonfiction** and **Television** require significantly less skipping (≈ 0.65), as they frequently require mapping multiple intersecting entities—such as connecting specific actors, fictional character names, and episodic plots. Crucially, this compute dynamicism is invariant to game-specific difficulty proxies, anchoring decisions solely on internal linguistic complexity. This stability is confirmed by remarkable consistency across game rounds (Jeopardy, Double Jeopardy, and Final Jeopardy) and a strictly flat skip rate (≈ 0.69) across all clue values from \$200 to \$3000+ (Figure 7).

Qualitative Token Analysis. Aggregated skip counts reveal a clear **information-entropy gradient** that explicitly decouples semantic content from structural redundancy (Table 4). At one extreme,

Semantic Anchors (Bucket 0; e.g., 'Truman', 'TV') possess high surprisal and undergo fully dense computation; the router identifies these as non-compressible constituents requiring full-rank processing.

Conversely, high-frequency **Function Words** and **Secondary Quantifiers** (Buckets 1–2) trigger moderate skipping. At the opposite pole, **Prompt Template Frames** (Bucket 3; e.g., ':', 'Question') bypass 21–29 layers. Because these boilerplate tokens are highly predictable, the router treats them as lightweight positional pointers. This demonstrates the MoD router’s ability to successfully distinguish the **problem space** from the **formatting space**, aggressively pruning the latter to preserve compute.

Table 4: Qualitative Token Frequency Modes across Equal 10-Layer Skip Buckets

Bucket	Skip Count	Top Sampled Tokens τ	Conceptual Role
Bucket 0	0 Layers	'Mary', 'Truman', 'Martha', 'TV', 'pet'	<i>Semantic Anchor</i>
Bucket 1	1–10 Layers	'of', 'his', 'as', 'to', 'AN'	<i>Function Word</i>
Bucket 2	11–20 Layers	'"', '&', '0', '9', 'IN'	<i>Secondary Quantifier</i>
Bucket 3	21–29 Layers	':', 'Category', 'Question', 'this', 'In'	<i>Prompt Template Frame</i>

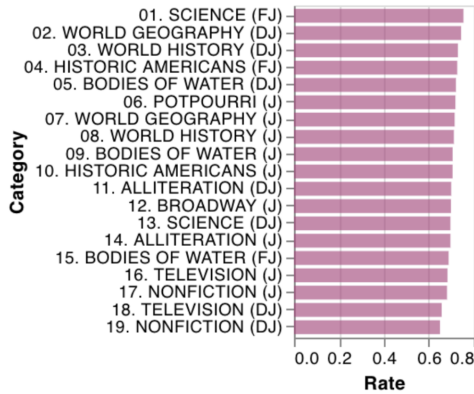


Figure 6: Skip Rate per Category

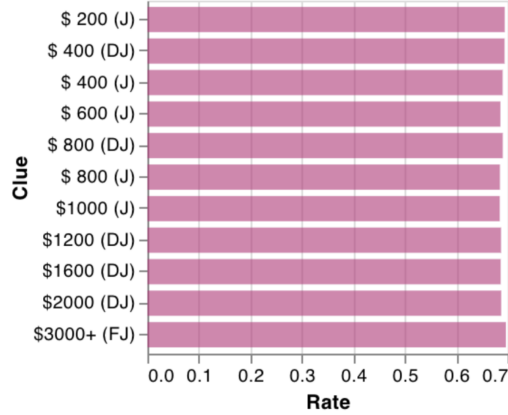


Figure 7: Skip Rate per Clue Value

*Jeopardy rounds: J=Jeopardy, DJ=Double Jeopardy, FJ=Final Jeopardy

7 Conclusion

In this work, we established a rigorous Pareto frontier for inference efficiency in knowledge-intensive question answering. Our joint LoRA-MoD architecture demonstrates that dynamic, adaptive thresholding can reduce total inference FLOPs by up to 48.6% with negligible accuracy degradation, while moderate sparsity actively regularizes the network to outperform dense baselines. Ultimately, co-optimizing these routing thresholds alongside task-specific adapters preserves critical representational stability, proving that intelligent, token-level compute allocation is a highly effective strategy for accelerating real-time inference.

Team contributions

Soha Sultan: LoRA training loop and hyper-parameter tuning for LoRA. Initial Mixture-of-Depths implementation version of joint training loop using Threshold- p routing logic. Visualization charts on Weights and Biases for training and evaluation loops, FLOPs calculations

Haoyue Yang: Qwen LoRA baseline, Modal Setup, MoD skipping logic correction and MLP optimization, sparsity loss improvement, Final Joint LoRA-MoD implementation, CUDA-synchronized benchmarking and inference warm-up for latency tracking. Significance test and results analysis.

Jan Kopanski: Project structure and tooling setup, Dataset preparation, Initial training pipeline, Evaluation pipeline and methodology (PEDANTS, RoBERTa, LLM-as-a-Judge with Gemini), Mixture-of-Depths architecture refinements and bugfixes, MoD hyperparameter sweep, Jeopardy game simulator, Data visualization, Qualitative analysis.

References

- [1] J! Archive. J! archive: The fan-created archive of Jeopardy! games and clues. <https://j-archive.com/>, 2026. Accessed: 2026-02-10.
- [2] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models, 2021.
- [3] David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models, 2024.
- [4] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, Veselin Stoyanov. RoBERTa: A Robustly Optimized BERT Pretraining Approach, 2019.
- [5] Tal Schuster, Adam Fisch, Jai Gupta, et al. Confident adaptive language modeling, 2022.
- [6] Yanxi Chen, Xuchen Pan, Yaliang Li, Bolin Ding, and Jingren Zhou. EE-LLM: Large-scale training and inference of early-exit large language models with 3d parallelism, 2024.
- [7] Andrey Gromov, Kushal Tirumala, Hassan Shapourian, Paolo Glorioso, and Daniel A. Roberts. The unreasonable ineffectiveness of the deeper layers, 2025.
- [8] Yaxin Luo, Gen Luo, Jiayi Ji, et al. γ -MoD: Exploring mixture-of-depth adaptation for multimodal large language models, 2024.
- [9] Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling, 2023.
- [10] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding, 2023.
- [11] Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, et al. Layerskip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024.
- [12] Julie Kallini, Shikhar Murty, Christopher D. Manning, Christopher Potts, and Róbert Csordás. MrT5: Dynamic token merging for efficient byte-level language models, 2025.
- [13] Chen Zhang, Meizhi Zhong, Qimeng Wang, et al. MoDification: Mixture of depths made easy. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics*, 2025.
- [14] Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Agustín Piqueres Lajarín, Vaibhav Srivastav, Joshua Lochner, Caleb Fahlgren, Xuan-Son Nguyen, Clémentine Fourier, Ben Burtenshaw, Hugo Larcher, Haojun Zhao, Cyril Zakka, Mathieu Morlon, Colin Raffel, Leandro von Werra, Thomas Wolf. SmolLM2: When Smol Goes Big – Data-Centric Training of a Small Language Model, 2025.
- [15] Zongxia Li, Ishani Mondal, Yijun Liang, Huy Nghiem, and Jordan Lee Boyd-Graber. Pedants: Cheap but effective and interpretable answer equivalence, 2024.
- [16] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, Zihan Qiu. Qwen3 Technical Report, 2025.

8 Appendix

8.1 Evaluation Framework: Jeopardy Simulation

To evaluate the trade-off between inference latency and response accuracy in a competitive environment, we implemented a discrete-event simulation of the Jeopardy! gameplay mechanics across a given broadcast season. For each clue c , models are sorted by their generation latency $L(c)$ such that $L_1 < L_2 < \dots < L_n$. The model possessing the minimum latency (L_1) is granted the primary opportunity to answer ("buzz-in"). If the model's prediction matches the ground-truth answer according to our Gemini-based evaluation judge, the model is awarded the static point value of the clue $V(c)$, and gameplay for that clue terminates. Conversely, an incorrect response incurs a deduction of $V(c)$, mimicking the game's penalty structure for sub-optimal retrieval, and the turn sequentially passes to the next-fastest model (L_2). Standard clues use their recorded base dataset values, while Final Jeopardy clues are fixed at a constant valuation of \$5,000. This reward-penalty framework strictly incentivizes not only accuracy but also fast response dispatch, punishing models with high answer hallucinations which would otherwise thrive in non-sequential scoring pipelines.

Evaluation Configuration Due to framework constraints regarding the execution of our custom MoD architecture on local hardware pipelines, we approximate comparative baselines using three distinct configurations of the Gemini 3 LLM ecosystem invoked via API endpoints. Specifically, we test: 1) **Gemini 3 Flash (Minimal Thinking)**: Structured to dispatch answers immediately with negligible thought-buffer overhead, 2) **Gemini 3 Flash (High Thinking)**: Preconditioned with instructions allowing a high allocation of sequential reasoning steps, generating increased latency before dispatching answers, 3) **Gemini 3 Pro (Low Thinking)**: A larger-parameter architecture constrained to low-frequency thinking allocations. Table 5 summarizes the aggregated earnings, buzz attempts, and turn-order breakout metrics calculated over all broadcast episodes of Season 42. We also record the per-episode scores in Figure 8.

Table 5: Jeopardy! Game Simulation Results (Season 42). "Attempted" metrics reflect actual gameplay turns triggered by sequential breakouts. "First Buzz" counts the number of times the model was the fastest to provide an answer. "Second Buzz" counts the number of times the model was the second fastest to answer after the fastest model provided an incorrect answer.

Model	Total Earnings	First Buzz	Second Buzz	Attempted	Accuracy When Buzzed
Gemini 3 Flash (Minimal)	\$6,084,600	6,807	1	6,808	96.31%
Gemini 3 Flash (High)	\$202,800	174	71	324	82.41%
Gemini 3 Pro (Low)	\$44,000	26	179	221	60.18%

Static Benchmark Evaluation To decouple inference throughput from the sequential mechanics of the buzzer simulation, we measure the absolute classification and retrieval capabilities of each configuration over the full $N = 7,007$ test questions. As expected, scaling the thinking allocation improves static reasoning accuracy: Gemini 3 Flash (High Thinking) achieves the peak correctness rate of 98.63% with Gemini-Match, outperforming its minimal-thought counterpart by roughly +2.25% absolute. However, this accuracy manifests with an almost 9-fold increase in average end-to-end response latency (9.51s vs. 1.04s), establishing the fundamental premise that static scoring systems tend to favor excessive computation budgeting that would prove uncompetitive in live-buzzer game state rollout.

Table 6: Static Benchmarks (All 7,007 Questions). Numbers inside the scoring metrics reflect percent correct (%) except average F1. Latency is measured in seconds per query (sec/q).

Model	Gemini	RoBERTa	PEDANT	Exact Match	Avg F1	Avg Latency
Gemini 3 Flash (Minimal)	96.38%	95.76%	95.05%	87.87%	0.900	1.04s
Gemini 3 Flash (High)	98.63%	97.37%	96.35%	89.40%	0.916	9.51s
Gemini 3 Pro (Low)	97.60%	96.57%	95.70%	89.05%	0.908	2.97s

Sequential Selection Bias in Competitive Inference A notable discrepancy appears for Gemini 3 Pro (Low Thinking), which achieves high static baseline accuracy of 97.60% Table 5, yet drops to only 60.18% "Accuracy when Buzzed" (Table 6) within the live execution framework. This contrast illustrates a strong sequential selection bias (or "survivorship bias") induced by the simulation mechanics. Because Gemini 3 Flash (Minimal Thinking) possesses extremely low latency, it almost invariably claims the first buzzer allocation (6, 807 times). Consequently, Gemini 3 Pro only receives a buzzer turn if Flash (Minimal Thinking) answers incorrectly. Because the pool of remaining available questions is highly filtered to only contain data that prior models failed, it acts as a hard-bracket filter isolating rare topics, obscure phrasing, or high-ambiguity completions. Therefore, even an absolute larger parameter model suffers depressed accuracy rates because it is exclusively forced to answer the most difficult questions in the test dataset to gain points, effectively lowering its live competitive performance to only items that its faster competitors found insolvable.

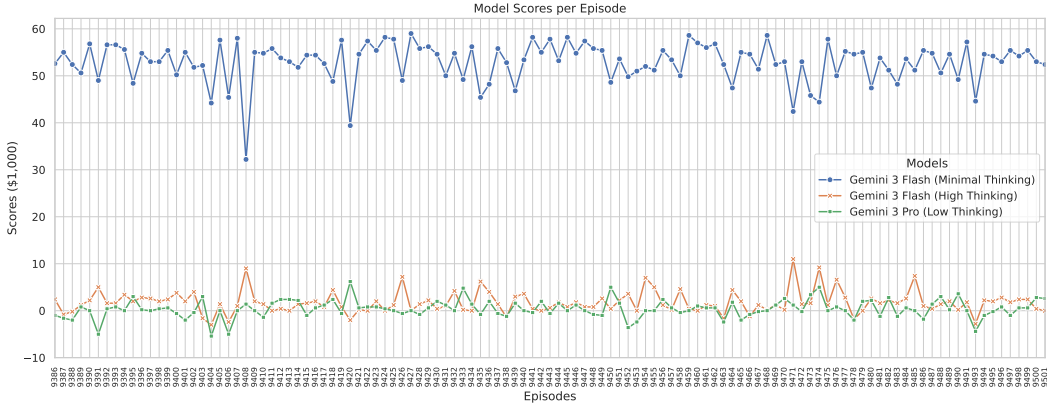


Figure 8: Jeopardy! Game Simulation scores per episode.

8.2 Baseline Calibration & Model Selection

8.2.1 Experimental Details

Using Gemini 3 Flash/Pro as zero-shot benchmarks, we evaluated the Base and Instruct variants of SmoLLM2-135M and Qwen3-0.6B[16]. Finding comparable performance, we prioritized SmoLLM2-135M for its superior iteration velocity.

A 7-iteration hyperparameter sweep on a 3-season subset established optimal parameters (BF16 precision, 1×10^{-3} learning rate, batch size 32, one epoch), which were then applied across all candidate models for a final 39-season comparative trial (510K samples). Structural adaptation via LoRA ($r = 32, \alpha = 64$) was applied to all modules except for `lm_head` and `embedding`. Following evaluation, SmoLLM2-135M was finalized as our implementation baseline to establish a stable, high-fidelity dense benchmark list for speedups comparison.

8.2.2 Results

Our evaluations (Table 7) establish Gemini 3 Flash as the accuracy ceiling. Within the edge tier, SmoLLM2-135M leads in efficiency, achieving 35.55% RoBERTa Match and 37.56% PEDANTS, outperforming the larger Qwen3-0.6B. We note that the Qwen3 baseline utilized a shared hyperparameter regime ($r = 32, \alpha = 64, lr = 10^{-3}$) that likely limited its scale-specific potential. Separately, the Base architecture consistently outperformed its Instruct counterpart, confirming our hypothesis that raw base-models facilitate superior domain adaptation. These gains (10.45% to 35.55% RoBERTa Match; 2.29% to 32.24% Exact Match) validate that fine-tuning narrows the factuality gap, transforming knowledge-blind baselines into domain experts.

Table 7: Model Evaluation Results (Test set: $N = 7007$)

Model	Gemini (%)	RoBERTa Match (%)	PEDANTS (%)	Exact Match (%)	Avg F1-Score	Avg Non-Batch Latency (s)
Gemini 3 Flash (minimal)	96.38	95.76	95.05	87.87	0.9000	1.04
Gemini 3 Flash (high)	98.63	97.37	96.35	89.40	0.9158	9.51
Gemini 3 Pro (low)	97.60	96.57	95.70	89.05	0.9079	2.97
Base Models						
SmolLM2-135M (Base)	1.96*	10.45	6.31	2.29	0.0204	0.24
SmolLM2-135M-Instruct	0.44*	10.96	3.84	0.53	0.0123	0.23
Qwen3-0.6B-Base	9.79	20.28	13.89	9.98	0.0431	0.35
Qwen3-0.6B (Instruct)	71.69*	3.60	1.64	0.13	0.0056	0.40
Fine-tuned Models						
SmolLM2-135M	34.87	35.55	37.56	32.24	0.3517	0.29
SmolLM2-135M-Instruct	26.43	28.81	30.97	25.96	0.2872	0.23
Qwen3-0.6B-Base	29.34	29.97	31.85	26.89	0.2952	0.29
Qwen3-0.6B (Instruct)	23.01	28.14	28.16	22.89	0.2351	0.42

* Indicates LLM-as-a-Judge parse failures due to non-canonical outputs.

8.3 MoD-Layer Token Skip Rate Pareto Frontier

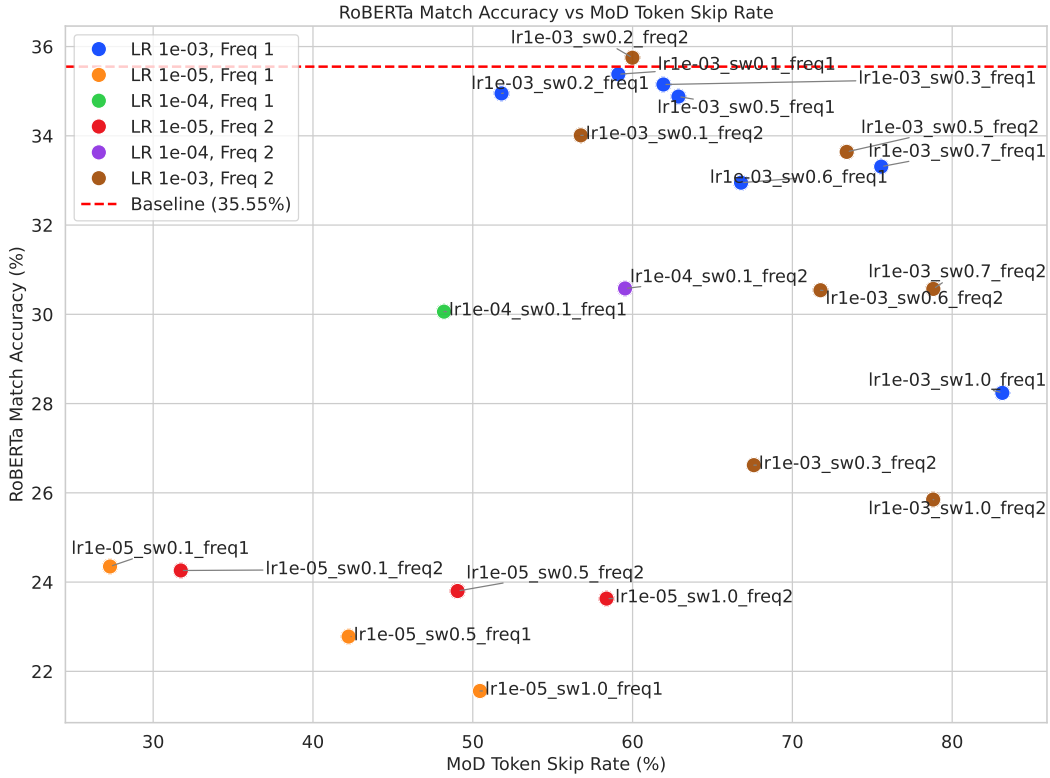


Figure 9: RoBERTa Match Accuracy vs. MoD Token Skip Rate. This figure shows the average percentage of tokens skipped at each MoD-equipped layer. Layers without MoD routers are not included in this analysis. The figure demonstrates the impact of token pruning aggressiveness on factual retrieval performance across different threshold and frequency configurations. The red dashed line denotes the baseline dense accuracy (35.55%).