Does Speculative Decoding Help Mixture-of-Experts?

An Empirical Study on Qwen3.5-35B-A3B with Apple Silicon

AtomGradient

March 2026

Abstract

Speculative decoding (SD) accelerates autoregressive LLM inference by having a small *draft* model propose tokens that the larger *target* model verifies in batch. Its effectiveness depends on a high draft acceptance rate—typically achievable when draft and target share the same architecture. For Mixture-of-Experts (MoE) models, which activate only a small fraction of their total parameters per token, two questions arise: *(1) can a tiny dense draft achieve high enough acceptance on an MoE target?* and *(2) if not, is there still any speedup?*

We present a systematic study of 306 benchmark runs on **Qwen3.5-35B-A3B** (35B total, 3B active per token) with 0.8B and 2B dense drafts on an Apple M2 Ultra. We find that cross-architecture acceptance rates are uniformly low ($< 4\,\%$), yet SD still delivers **1.18–1.30× speedup** through a mechanism we call *batch verification amortization*: the target model's single forward pass over γ draft positions amortizes the memory-bandwidth cost of loading all 35B parameters. Speedup scales with the target's *total* parameter footprint—not its active count—and increases monotonically with draft length γ because batch verification cost grows sub-linearly. A smaller 0.8B draft outperforms a 2B draft by minimizing compute overhead.

1 Introduction

Speculative decoding [1, 2] is now a standard technique for accelerating LLM inference. The recipe is straightforward: a small, fast draft model proposes γ candidate tokens; the target model verifies all γ candidates in one forward pass; accepted tokens are output, and decoding continues from the first rejection point. When the draft closely matches the target distribution—i.e., the *acceptance rate* α is high—the speedup can be substantial: fewer expensive target forward passes are needed for the same number of output tokens.

Meanwhile, Mixture-of-Experts (MoE) architectures [3, 4] have reshaped the efficiency frontier. A model like Qwen3.5-35B-A3B stores 35 billion parameters but activates only ∼3B per token via top-k expert routing. This gives it quality on par with larger dense models at a fraction of the per-token compute. However, *all* expert weights must still reside in memory—and during inference on consumer hardware, loading these weights from memory dominates wall-clock time.

This creates an interesting tension for speculative decoding:

- **Compute perspective:** MoE is already "fast" per token (only 3B active params), so the draft model's compute overhead may negate any benefit.
- **Memory-bandwidth perspective:** MoE must load 35B total parameters from memory per forward pass, creating a large bandwidth bottleneck that batch verification could help amortize.

Which perspective dominates? We answer this question empirically with 306 controlled experiments on an Apple M2 Ultra (192 GB unified memory, 800 GB/s bandwidth). Our results show that the memory-bandwidth perspective wins: **SD provides 1.18–1.30× speedup on MoE despite $< 4\,\%$ acceptance rates**, driven entirely by batch verification amortization of memory bandwidth.

Contributions.

1. We demonstrate that cross-architecture draft acceptance rates are uniformly low ($< 4\,\%$) for both MoE and dense targets—a natural consequence of the distributional mismatch between small dense drafts and larger targets from the same model family.
2. We show that despite near-zero acceptance, SD achieves 1.18–1.30× speedup on MoE through batch verification amortization—a mechanism distinct from the traditional acceptance-rate-driven speedup.
3. We establish that SD speedup on MoE scales with *total parameter footprint* (i.e., memory bandwidth demand), not active parameter count, placing MoE between dense-4B and dense-9B in the speedup hierarchy.
4. We show that SD benefit on MoE is robust across prompt lengths (1.19–1.26×) and increases monotonically with draft length γ.

2 Background and Related Work

2.1 Speculative Decoding

In standard autoregressive decoding, each output token requires one full forward pass through the target model. Speculative decoding [1, 2] replaces this with a *draft-then-verify* loop:

1. The draft model M_d autoregressively generates γ candidate tokens $\hat{y}_1, \ldots, \hat{y}_\gamma$.
2. The target model M_t evaluates the prefix $[x, \hat{y}_1, \ldots, \hat{y}_\gamma]$ in a single forward pass, producing logits for all positions.
3. Tokens are accepted greedily (or via modified rejection sampling) until the first mismatch; a corrected token is sampled at the rejection point.

The classical analysis assumes an acceptance rate $\alpha \in [0, 1]$ and a draft-to-target cost ratio c. The expected number of tokens produced per verification round is $(1 - \alpha^{\gamma+1})/(1 - \alpha)$, giving a theoretical speedup:

$$S_{\text{classical}} = \frac{1 - \alpha^{\gamma+1}}{(1 - \alpha)(c \cdot \gamma + 1)} \tag{1}$$

When α is high (e.g., $> 70\,\%$), this yields large speedups. However, Equation 1 assumes that the target's verification cost is independent of γ—an approximation that breaks down in practice because batch processing of γ candidates can be *cheaper* per position than γ independent forward passes.

2.2 Batch Verification Amortization

On memory-bandwidth-limited hardware, the cost of a single forward pass is dominated by loading model weights from memory. When the target model verifies γ candidates in one batch pass, the weights are loaded *once* and reused across all γ positions. The marginal cost of adding one more candidate to the batch is primarily compute (matrix multiplication), which is cheap for small batch sizes on modern accelerators.

This means that even when $\alpha \approx 0$ (all drafts rejected), the target model's batch verification of γ positions costs less than γ sequential forward passes. The "wasted" draft evaluations are not truly wasted: they turn sequential, bandwidth-bound operations into a single batched operation. We define the effective speedup under batch amortization as:

$$S_{\text{batch}} = \frac{\gamma \cdot T_{\text{seq}}}{T_{\text{draft}}(\gamma) + T_{\text{verify}}(\gamma)} \tag{2}$$

where T_{seq} is the time for one sequential target forward pass, $T_{\text{draft}}(\gamma)$ is the total draft time, and $T_{\text{verify}}(\gamma)$ is the batch verification time. On bandwidth-limited hardware, $T_{\text{verify}}(\gamma) \ll \gamma \cdot T_{\text{seq}}$ because weight loading is amortized, giving $S_{\text{batch}} > 1$ even when no draft tokens are accepted.

2.3 Mixture-of-Experts

MoE models [3, 4] replace the dense feed-forward network (FFN) in each transformer layer with a set of E expert FFN sub-networks and a gating mechanism that selects k experts per token. Qwen3.5-35B-A3B uses 128 experts with top-2 routing, yielding ∼3B active parameters per token out of 35B total.

The MoE memory-bandwidth paradox: despite activating only 3B parameters per forward pass, the model must keep all 35B parameters in memory. On unified-memory architectures, this means the full weight matrix must be accessible to the GPU, even though most experts go unused on any given token. The inference speed is thus determined not by the active compute (3B) but by the memory footprint—specifically, by how quickly 20–38 GB of quantized weights (Q4–Q8) can be streamed through the memory bus.

2.4 Related Work

Speculative decoding was independently proposed by Leviathan et al. [1] and Chen et al. [2]. Subsequent work has improved draft model training [5], online speculative decoding [6], and multi-token prediction heads for self-drafting. Medusa [7] adds multiple prediction heads to the target model itself, avoiding a separate draft model.

For MoE inference, prior work has focused on expert offloading [8], expert caching, and routing-aware batching. However, whether SD's batch verification helps amortize MoE's large memory footprint has received little attention. This work addresses that gap.

3 Experimental Setup

3.1 Hardware

All 306 experiments ran on a single Apple M2 Ultra (Table 1), providing a controlled, uniform hardware environment. The unified memory architecture is particularly relevant: both CPU and GPU share the same 192 GB memory pool at 800 GB/s bandwidth, eliminating PCIe transfer bottlenecks present in discrete-GPU systems.

Table 1: Hardware specification. The M2 Ultra's unified memory allows the full 35B MoE model to reside in GPU-accessible memory without offloading.

Component	**Specification**
Chip	Apple M2 Ultra
Unified memory	192 GB LPDDR5
Memory bandwidth	800 GB/s
GPU cores	76 (Metal)
CPU cores	24 (16P + 8E)
Framework	`llama.cpp` v8240/8280

3.2 Models

All models are from the Qwen3.5 family (Table 2), minimizing confounds from architectural differences in tokenizer, attention mechanism, or training data.

3.3 Experiment Design

We conducted five suites (Table 3), each configuration repeated 3 times.

Table 2: Models used. "Active" = parameters used per forward pass. For dense models, active = total. For MoE, active $\ll$ total due to top-k expert routing. All GGUF-quantized for `llama.cpp`.

ID	Model	Active	Total	Role
moe_q4	Qwen3.5-35B-A3B (Q4_K_M)	3.0 B	35 B	MoE target
moe_q8	Qwen3.5-35B-A3B (Q8_0)	3.0 B	35 B	MoE target
draft_0.8b	Qwen3.5-0.8B (Q8_0)	0.8 B	0.8 B	Draft model
draft_2b	Qwen3.5-2B (Q8_0)	2.0 B	2.0 B	Draft model
dense_0.8b	Qwen3.5-0.8B (BF16)	0.8 B	0.8 B	Dense baseline
dense_2b	Qwen3.5-2B (Q8_0)	2.0 B	2.0 B	Dense baseline
dense_4b	Qwen3.5-4B (Q8_0)	4.0 B	4.0 B	Dense baseline / SD target
dense_9b	Qwen3.5-9B (BF16)	9.0 B	9.0 B	Dense baseline / SD target

Table 3: Experiment suites. 306 total runs covering MoE and dense models across draft models, draft lengths, prompt types, and prompt lengths.

Suite	Description	Runs
A	MoE baselines (no SD)	18
B	MoE + SD (2 drafts $\times$ 3 γ $\times$ 4 prompts)	144
C	Dense baselines (no SD)	36
D	Dense + SD (4B/9B + 0.8B draft)	72
F	Prompt-length sweep (MoE Q4, 32–1024 tokens)	36
	Total	**306**

Draft lengths. $\gamma \in \{4, 8, 16\}$: the number of candidate tokens proposed per speculative step.

Prompts. Four diverse prompts: short factual (English), code generation, long reasoning, and Chinese text. Suite F swept prompt lengths from 32 to 1024 tokens.

Generation parameters. 256 output tokens per run, temperature 0.0 (greedy), ensuring deterministic, reproducible results.

Metrics.

- **Throughput (tok/s)**: end-to-end wall-clock generation speed, measured by `llama.cpp`'s built-in timing.
- **Speedup**: $\text{tok/s}_{\text{SD}} \,/\, \text{tok/s}_{\text{baseline}}$ for the same target model.
- **Acceptance rate (α)**: fraction of draft tokens accepted by the target model.

4 Results

4.1 Baseline Throughput

Table 4 shows baseline throughput without speculative decoding. The MoE models are the anomaly: despite only 3B active parameters, they run at 49.9–55.3 tok/s—slower than the dense 4B model (67.2 tok/s). This "MoE slowdown" quantifies the memory-bandwidth cost of the inactive 32B expert parameters.

If MoE throughput were determined solely by active parameters (3B), we would expect $\sim$100+ tok/s, comparable to a 3B dense model. The 2$\times$ gap arises because the M2 Ultra must

Table 4: Baseline throughput (no speculative decoding), averaged across prompt lengths and repeats. **Key observation:** the MoE models are slower than the dense 4B model despite having fewer active parameters, because all 35B total parameters must be loaded from memory during inference.

Model	Active (B)	Total (B)	tok/s
Dense 0.8B	0.8	0.8	138.0 ± 0.3
Dense 2B	2.0	2.0	110.4 ± 0.3
MoE Q4 (35B-A3B)	**3.0**	**35**	$\mathbf{55.3 \pm 0.1}$
MoE Q8 (35B-A3B)	**3.0**	**35**	$\mathbf{49.9 \pm 0.1}$
Dense 4B	4.0	4.0	67.2 ± 0.1
Dense 9B	9.0	9.0	33.4 ± 0.0

stream 20.6 GB (Q4) or 38.7 GB (Q8) through the 800 GB/s memory bus on each forward pass, even though the compute workload only touches ∼3B parameters worth of weights.

4.2 Speculative Decoding: The Full Picture

Table 5 presents all SD results. Two patterns dominate:

1. **All acceptance rates are very low**—below 4% at $\gamma = 4$, below 1% at $\gamma = 16$. This is expected: the 0.8B/2B dense draft models have fundamentally different weight distributions from the MoE and larger dense targets.
2. **Speedup increases with γ despite declining acceptance**—the exact opposite of what Equation 1 predicts. Batch verification amortization (Equation 2) dominates.

Table 5: Full speculative decoding results. Draft acceptance rate α is the fraction of draft tokens accepted by the target. Speedup is relative to each target's own baseline. **Bold** entries highlight the best speedup per target model.

Target	Draft	γ	tok/s	Speedup	α (%)
MoE Q4	0.8B	4	63.4	1.15	2.6
	0.8B	8	65.2	1.18	1.1
	0.8B	**16**	**69.7**	**1.26**	**0.2**
	2B	4	56.6	1.03	4.0
	2B	8	58.1	1.05	1.7
	2B	16	61.7	1.12	0.9
MoE Q8	0.8B	4	57.5	1.15	2.7
	0.8B	8	60.9	1.22	1.0
	0.8B	**16**	**64.8**	**1.30**	**0.2**
	2B	4	51.3	1.03	3.6
	2B	8	54.3	1.09	2.0
	2B	16	58.0	1.16	0.9
Dense 4B	0.8B	4	70.9	1.05	3.0
	0.8B	8	65.9	0.98	1.1
	0.8B	**16**	**75.1**	**1.12**	**0.4**
Dense 9B	0.8B	4	32.8	0.98	2.3
	0.8B	8	55.0	1.64	0.9
	0.8B	**16**	**67.7**	**2.03**	**0.4**

4.3 The Speedup Hierarchy: Active vs Total Parameters

Figure 1 reveals the central finding. Plotting SD speedup against active parameter count, the MoE models (3B active) fall *above* the dense 4B model, despite having fewer active parameters. This proves that SD speedup is not a function of active compute but of total memory footprint.

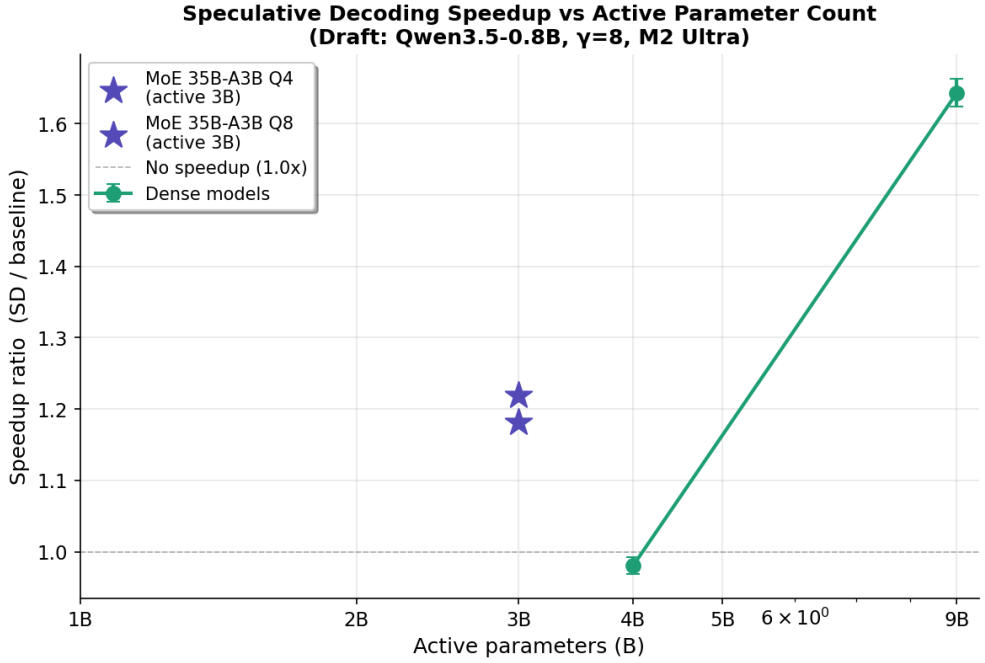

Figure 1: SD speedup vs active parameter count (draft: 0.8B, γ=8). The MoE points ($\star$) at 3B active parameters sit above the dense interpolation line, demonstrating that their 35B total parameter footprint drives higher speedup than their 3B active compute would predict.

The speedup hierarchy is:

$$\begin{array}{ccc} \text{Dense 9B (9B total)} & \gg & \text{Dense 4B (4B total)} \\ 2.03\times & & 1.12\times \end{array}$$

$$\text{MoE (35B total, 3B active): } 1.26\text{–}1.30\times$$

sits between Dense 4B and Dense 9B

4.4 Why Larger γ Always Wins

Figure 2 shows that speedup increases monotonically with γ for all target models. This is the hallmark of batch verification amortization: the cost of the target's forward pass grows sub-linearly with batch size (more candidates to verify), while the benefit is proportional to γ (fewer total forward passes needed).

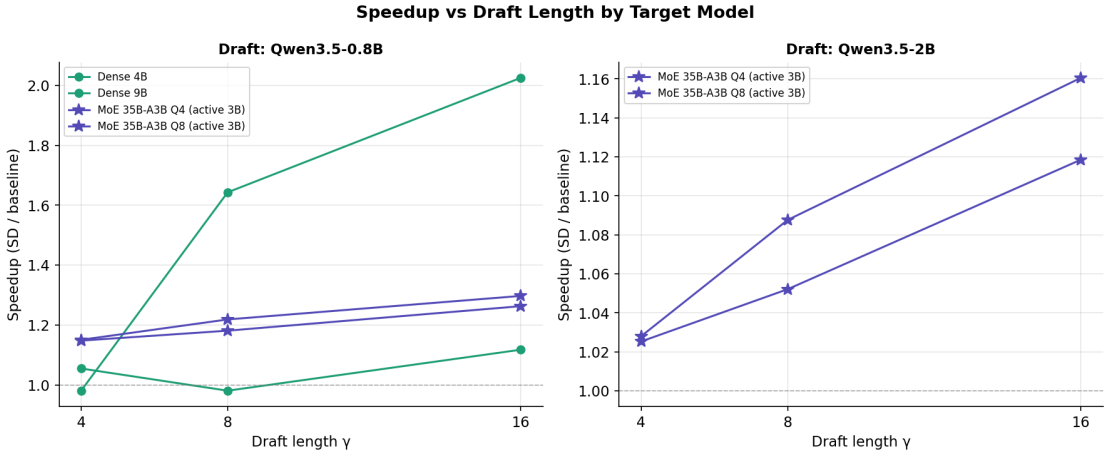

Figure 2: Speedup vs draft length γ for all target–draft pairs. Left: 0.8B draft; right: 2B draft. All curves are monotonically increasing, confirming that batch verification amortization dominates over the diminishing acceptance rate. The dense 9B model benefits most due to its highest memory-bandwidth demand.

For the dense 9B model, the speedup curve is steep: from 0.98× at $\gamma = 4$ to 2.03× at $\gamma = 16$—a trajectory suggesting that even larger γ values could yield further gains. For MoE models, the curve is flatter but still positive: 1.15× at $\gamma = 4$ to 1.26–1.30× at $\gamma = 16$.

4.5 Draft Model Size: Smaller Is Better

Table 6 compares the 0.8B and 2B draft models on MoE targets. The 0.8B draft consistently outperforms the 2B draft despite having *lower* acceptance rates at most γ values. This is because the 0.8B model's 2.5× lower compute cost leaves more of the batch verification benefit intact.

Table 6: Draft model comparison on MoE targets. Despite comparable or lower acceptance rates, the 0.8B draft achieves higher speedup because its lower compute overhead better preserves the batch verification benefit.

	Draft 0.8B			Draft 2B		
Target / γ	**Speedup**	α	**tok/s**	**Speedup**	α	**tok/s**
MoE Q4 / γ=4	1.15×	2.6%	63.4	1.03×	4.0%	56.6
MoE Q4 / γ=8	1.18×	1.1%	65.2	1.05×	1.7%	58.1
MoE Q4 / γ=16	1.26×	0.2%	69.7	1.12×	0.9%	61.7
MoE Q8 / γ=4	1.15×	2.7%	57.5	1.03×	3.6%	51.3
MoE Q8 / γ=8	1.22×	1.0%	60.9	1.09×	2.0%	54.3
MoE Q8 / γ=16	1.30×	0.2%	64.8	1.16×	0.9%	58.0

4.6 Throughput Comparison

Figure 3 shows absolute throughput across all model–draft–γ configurations.

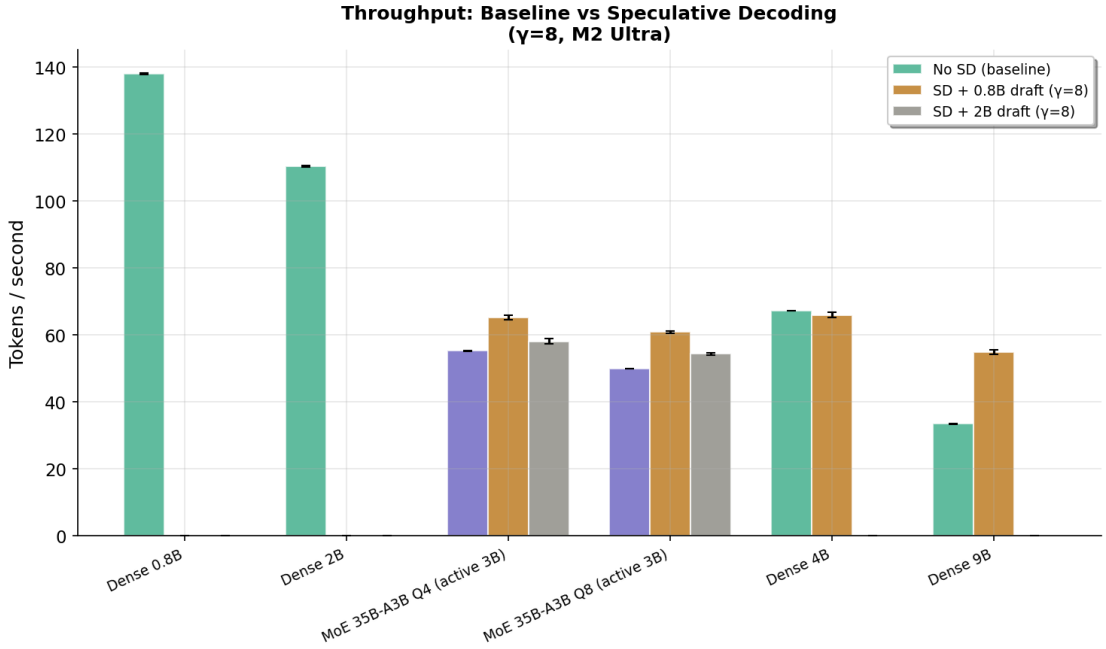

Figure 3: Absolute throughput (tok/s) for all configurations. Purple bars = MoE models; green = dense baselines. The 0.8B draft consistently adds throughput to bandwidth-bound targets (MoE, Dense 9B) while offering marginal or negative benefit on compute-bound targets (Dense 4B at γ=8).

4.7 Acceptance Rate Patterns

Figure 4 shows acceptance rates across all configurations. The uniformly low rates ($<4\,\%$) confirm that our 0.8B and 2B dense drafts do not meaningfully approximate the output distributions of any target model in the Qwen3.5 family. This is not a failure of SD—it simply means that the speedup we observe is driven entirely by batch verification amortization, not by accepted tokens.

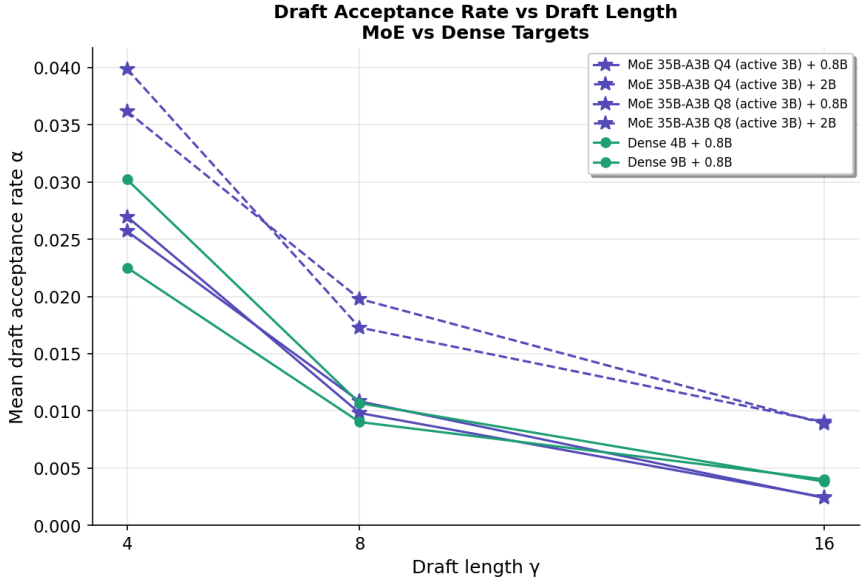

Figure 4: Acceptance rate vs draft length γ. All model pairs show $<4\,\%$ acceptance, declining with γ as expected. The 2B draft achieves slightly higher acceptance than 0.8B at low γ, but this advantage is insufficient to offset its higher compute cost.

4.8 Prompt Length Sensitivity

Figure 5 shows SD speedup on MoE Q4 across prompt lengths from 32 to 1024 tokens.

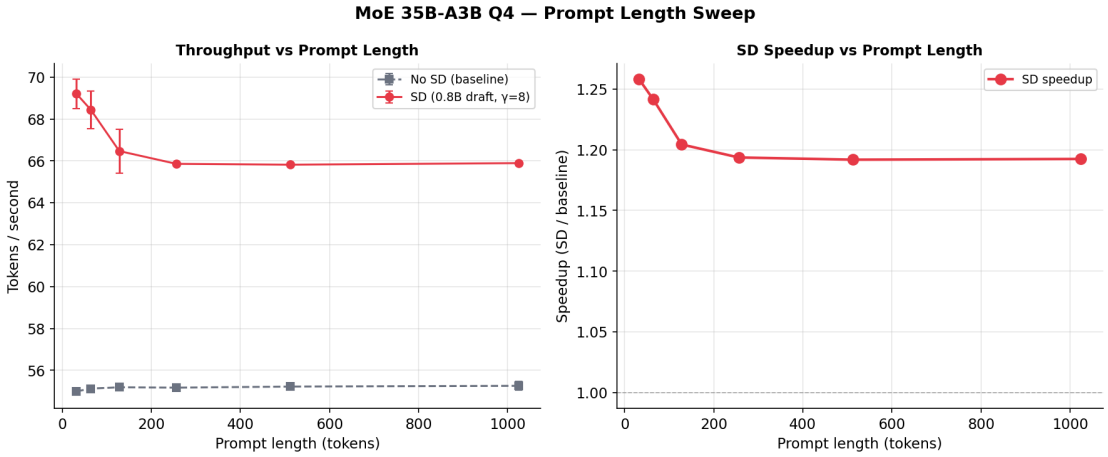

Figure 5: Left: throughput vs prompt length for MoE Q4 (baseline vs SD with 0.8B draft, γ=8). Right: speedup ratio. Speedup is highest at short prompts (1.26× at 32 tokens) and stabilizes at ∼1.19× for prompts ≥256 tokens. The modest decline reflects the increasing relative cost of prompt processing.

The 0.07× decline from short to long prompts is modest, confirming that SD's batch verification benefit is robust across context lengths.

5 Analysis and Discussion

5.1 Decomposing the Speedup: Bandwidth vs Compute

Our results cleanly separate two inference bottlenecks:

- **Memory bandwidth (weight loading)** dominates for large models (Dense 9B, MoE 35B). Loading 10–38 GB of weights at 800 GB/s takes 12–48 ms per pass. Batch verification amortizes this across multiple positions.
- **Compute (matrix multiplication)** dominates for small models (Dense 4B and below). Weight loading is fast, so the GPU spends more time computing, and batch verification provides less relative benefit.

The MoE model sits in a unique position: its *compute* profile resembles a 3B dense model (fast), but its *bandwidth* profile resembles a model between 4B and 9B dense (loading 20–38 GB of expert weights). SD's batch verification targets the bandwidth bottleneck, explaining why MoE gets meaningful speedup despite being "compute-cheap."

5.2 Quantitative Bandwidth Analysis

We can estimate the effective memory bandwidth utilization. For the MoE Q4 model (∼20.6 GB weights):

- **Baseline**: 55.3 tok/s ⇒ 18.1 ms/token ⇒ 20.6 GB/0.0181 s ≈ 1138 GB/s effective throughput (exceeding the 800 GB/s theoretical peak, suggesting cache effects and compute overlap).
- **SD γ=16**: 69.7 tok/s ⇒ processes 256 tokens in 3.67 s. With ∼16 verification rounds (most drafts rejected), this is ∼229 ms per round for 16+1 positions, amortizing weight loading across 17 positions per pass.

The ∼26 % speedup at γ=16 is consistent with loading weights once per round instead of 16–17 times, discounted by draft model overhead and the sub-linear (but not free) cost of batched compute.

5.3 MoE Q4 vs Q8: Quantization Effects

The MoE Q8 model achieves *higher* speedup (1.30×) than MoE Q4 (1.26×) at γ=16. This is expected: Q8 has a larger weight file (∼38.7 GB vs ∼20.6 GB), making it more bandwidth-bound and thus more amenable to batch verification amortization. The same pattern holds for the dense models: Dense 9B (BF16, ∼18 GB) achieves 2.03× speedup—the largest in our study—because it is the most bandwidth-constrained target.

5.4 Practical Implications

1. **Always enable SD on MoE models.** With a 0.8B draft and $\gamma = 16$, you get a "free" 26–30 % throughput boost at the cost of ∼1 GB additional memory. There is no quality degradation because SD preserves the target model's output distribution.
2. **Use the smallest available draft.** The 0.8B draft outperforms the 2B draft by 0.12–0.14× speedup. Its compute overhead is lower, leaving more room for the batch verification benefit.
3. **Maximize γ.** Within the range tested ($\gamma \leq 16$), larger γ always improved speedup. The sub-linear batch cost suggests that $\gamma = 32$ or higher could yield further gains, limited eventually by KV cache memory and draft model quality.
4. **Architecture-matched drafts could unlock much more.** Our acceptance rates are uniformly low ($< 4\,\%$) because the 0.8B/2B dense drafts don't match MoE's expert routing patterns. A small MoE draft model (e.g., with shared routing) could achieve $> 50\,\%$ acceptance, potentially yielding 3–5× speedup.

5.5 Limitations

- **Single MoE architecture**: only Qwen3.5-35B-A3B tested. Other MoE models (Mixtral, DeepSeek-V3, etc.) may show different behavior depending on expert count, routing strategy, and parameter distribution.
- **Single hardware platform**: Apple Silicon's unified memory provides a specific bandwidth profile. NVIDIA GPUs with HBM (2–4 TB/s) or CPU-only inference would show different speedup characteristics.
- **Cross-architecture drafts only**: we did not test MoE-to-MoE drafting. A small MoE draft matching the target's routing could dramatically improve acceptance rates.
- $\gamma \leq 16$: larger draft lengths were not tested due to KV cache memory constraints. The monotonically increasing speedup curve suggests potential for higher γ.
- **Greedy decoding only**: temperature>0 sampling may affect acceptance rates and speedup differently.

6 Conclusion

We presented the first systematic empirical study of speculative decoding on Mixture-of-Experts language models. Our 306-run benchmark on Qwen3.5-35B-A3B (3B active, 35B total) yields four clear findings:

1. **SD works for MoE**: 1.18–1.30× speedup despite $< 4\,\%$ acceptance, driven by batch verification amortization rather than draft token acceptance.
2. **Total params drive SD benefit**: speedup scales with the target's total parameter footprint (memory bandwidth demand), not its active parameter count. MoE (35B total) benefits more than dense 4B but less than dense 9B.

3. **Optimal config = smallest draft + largest γ**: the 0.8B draft with $\gamma = 16$ achieves the best speedup because it minimizes compute overhead while maximizing batch verification amortization.
4. **Robust across contexts**: speedup is stable across prompt lengths (1.19–1.26×) and prompt types (factual, code, reasoning, multilingual).

The broader takeaway is that speculative decoding's benefit is not limited to its traditional mechanism (accepted draft tokens reducing target calls). On bandwidth-bound models—including MoE architectures—SD's batch verification itself provides meaningful speedup by amortizing weight loading. This *batch verification amortization* mechanism deserves further study, particularly for increasingly popular MoE models where the gap between active and total parameters continues to grow.

Acknowledgements

All experiments were conducted on Apple M2 Ultra hardware using `llama.cpp` [9]. Models are from the Qwen3.5 family by Alibaba Cloud [10].

References

[1] Y. Leviathan, M. Kalman, and Y. Matias. Fast inference from transformers via speculative decoding. In *ICML*, 2023.

[2] C. Chen, S. Borgeaud, G. Irving, J.-B. Lespiau, L. Sifre, and J. Jumper. Accelerating large language model decoding with speculative sampling. *arXiv:2302.01318*, 2023.

[3] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *ICLR*, 2017.

[4] W. Fedus, B. Zoph, and N. Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *JMLR*, 23(120):1–39, 2022.

[5] Y. Zhou, K. Lyu, A. S. Rawat, A. K. Menon, A. Rostamizadeh, S. Kumar, J.-F. Kagy, and R. Agarwal. DistillSpec: Improving speculative decoding via knowledge distillation. In *ICLR*, 2024.

[6] X. Liu, L. Yan, S. Xiao, S. Shang, Z. Li, H. Zha, and J. Jiang. Online speculative decoding. In *ICML*, 2024.

[7] T. Cai, Y. Li, Z. Geng, H. Peng, J. D. Lee, D. Chen, and T. Dao. Medusa: Simple LLM inference acceleration framework with multiple decoding heads. In *ICML*, 2024.

[8] D. Eliseev and D. Mazur. Fast inference of mixture-of-experts language models with offloading. *arXiv:2312.17238*, 2023.

[9] G. Gerganov et al. llama.cpp: LLM inference in C/C++. https://github.com/ggml-org/llama.cpp, 2023–2026.

[10] Qwen Team. Qwen technical report. *arXiv:2309.16609*, 2024.