Accelerating Large Language Models using Speculative Decoding

Leying Li
EURECOM
leying.li@eurecom.fr

Alexis Balestra
EURECOM
Alexis.Balestra@eurecom.fr

Gabriel Lafont–Millet
EURECOM
Gabriel.Lafont-Millet@eurecom.fr

Abstract

Speculative decoding is a promising technique for accelerating inference in large language models (LLMs) by leveraging a lightweight draft model to propose multiple tokens per step, which are then selectively verified by a larger, more accurate target model. In this project, we implement speculative decoding for the open-source Qwen2.5 model family and evaluate its performance through a series of systematic experiments.

We benchmark speculative decoding against classical sampling using the HumanEval code generation dataset and explore the effects of parameters such as draft length and temperature on decoding speed and output quality. Our results show that speculative decoding achieves up to **3.7× lower latency per token** and **nearly 5× higher throughput**. However, these speed gains come with **a modest reduction in generation accuracy**, particularly in pass@1 and pass@5 metrics when the draft model is significantly weaker than the target.

Through detailed analysis of draft token acceptance rates, we observe that accuracy degradation is more pronounced under high-temperature or long draft-length configurations. These findings highlight the need to balance speculation aggressiveness and verification strength. We also evaluate the feasibility of dynamic draft length adaptation as a future optimization.

This work provides practical insights into the trade-offs of deploying speculative decoding for efficient yet accurate LLM inference.

1 Introduction

Large Language Models (LLMs) such as GPT-4 [1], PaLM [2], and Qwen2 [7] have achieved impressive performance in tasks ranging from open-domain dialogue to program synthesis. However, their high inference cost—particularly during autoregressive decoding—poses a major challenge for latency-sensitive and resource-constrained applications. Each output token requires a full forward pass, which limits throughput and leads to significant latency [12], especially in applications like live coding assistants, real-time agents, and edge deployments.

To alleviate these costs, a variety of acceleration strategies have been proposed. Compression techniques such as quantization [6], pruning, and knowledge distillation [10] reduce model size but often introduce additional training burdens or accuracy loss. Caching-based optimizations [9] exploit autoregressive dependencies to reuse computation, while early-exit methods [11] reduce depth

Preprint. Under review.

dynamically. However, these methods typically require architectural changes or access to internal model parameters.

Speculative decoding offers an attractive alternative that accelerates generation without modifying model structure. It combines a small draft model with a larger target model: the draft predicts multiple next tokens in parallel, and the target selectively verifies or corrects them. This reduces the number of sequential decoding steps while retaining the original model's predictive quality [3, 8]. Though adopted in proprietary systems [5], speculative decoding remains underexplored in open-source LLMs.

In this work, we implement speculative decoding for the Qwen2.5 model family [7], a suite of open-source decoder-only LLMs, and evaluate its performance on real-world code generation tasks. We focus on measuring both speed and accuracy trade-offs, analyzing the effects of key hyperparameters such as draft length and temperature.

Our key contributions are:

- We implement speculative decoding for Qwen2.5 using HuggingFace APIs, supporting configurable draft-token generation and temperature scaling.
- We evaluate performance on the HumanEval benchmark and observe up to **3.75× lower latency** and **4.23× higher throughput** for the 0.5B→3B configuration, and up to **2.78× lower latency** and **4.17× throughput** for 0.5B→1.5B.
- Our accuracy results show that speculative decoding maintains reasonable generation quality but can slightly degrade pass@k metrics, particularly when the draft model is small. For example, we observe up to **2–5% loss in pass@1/5** in both target scales.
- We analyze how temperature and draft length influence token acceptance and decoding efficiency, providing practical guidance for balancing speed and accuracy.

The remainder of this paper is structured as follows. Section 2 presents the implementation of speculative decoding and its algorithmic components. Section 3 describes the evaluation setup and dataset. Section 4 discusses our results on speed, accuracy, and acceptance rate. Section 5 concludes with insights and future directions.

2 Method

2.1 Speculative Decoding Overview

Speculative decoding [3, 8] is an inference-time acceleration technique for autoregressive language models. It operates by combining two models: a smaller, faster *draft model*, and a larger, more accurate *target model*. The draft model proposes multiple tokens in a single step, and the target model selectively verifies or corrects them. This reduces the number of sequential decoding steps required, improving overall efficiency.

Formally, given a prompt $x_{1:t}$, the draft model p_ϕ generates k candidate tokens $\tilde{x}_{t+1:t+k}$ using autoregressive sampling. The target model p_θ then evaluates the conditional probabilities $p_\theta(\tilde{x}_{t+i} \mid x_{1:t+i-1})$ for each token. Accepted tokens are appended to the output. If all k tokens are accepted, the target model generates an additional token (an *extra step*) to ensure generation continues. If any token is rejected, the target model resamples from that point onward. This method preserves the target model's output distribution while amortizing the cost of verification over multiple tokens, and does not require architecture changes or retraining.

The acceptance criterion for each draft token is calculated as:

$$\alpha_i = \frac{p_\theta(\tilde{x}_{t+i} \mid x_{1:t+i-1})}{p_\phi(\tilde{x}_{t+i} \mid x_{1:t+i-1})} \tag{1}$$

A token $\tilde{x}_{t+i}$ is accepted if $\alpha_i \geq 1$, or with probability α_i if $\alpha_i < 1$.

Algorithm 1 Speculative Decoding

```
Input: Prompt $x_{1:t}$, draft model $p_\phi$, target model $p_\theta$, draft length $k$
Output: Generated output sequence $x$
1 Initialize $x \leftarrow x_{1:t}$ while not finished do
2     Sample $k$ draft tokens $\tilde{x}_{t+1:t+k} \sim p_\phi$ for $i = 1$ to $k$ do
3         Compute $\alpha_i$ using Eq. (1) if $\alpha_i \geq 1$ or accepted with probability $\alpha_i$ then
4             Append $\tilde{x}_{t+i}$ to $x$
5         else
6             Resample token $x_{t+i} \sim p_\theta$ and append to $x$;
                break
7     if all $k$ tokens accepted then
8         Sample and append one extra token from $p_\theta$
9 return $x$
```

2.2 Model Selection

We select the Qwen2.5 model family [7] for implementing and evaluating speculative decoding, motivated by the following factors:

- **Open-source accessibility**: All Qwen2.5 models are openly available via Hugging Face, making them ideal for reproducible experimentation.
- **Decoder-only architecture**: The models follow an autoregressive, decoder-only design, which naturally aligns with speculative decoding methods.
- **Scalable model sizes**: Qwen2.5 offers models at multiple scales (e.g., 0.5B, 1.5B, 3B), enabling flexible draft–target configurations for performance analysis.
- **Strong baseline performance**: Despite being relatively compact, Qwen2.5 models perform competitively on benchmarks such as MMLU and HumanEval.

These properties make Qwen2.5 a practical and representative choice for studying decoding-time acceleration strategies.

2.3 Our Implementation

We implement speculative decoding using `Qwen2.5-0.5B` as the draft model, and experiment with both `Qwen2.5-1.5B` and `Qwen2.5-3B` as target models to explore different draft-to-target size ratios. All models are integrated via the Hugging Face Transformers interface and executed in PyTorch.

The implementation includes configurable temperature sampling, adaptive stopping based on EOS token or double newline, and context window truncation to 1024 tokens to ensure GPU memory stability. We additionally log key statistics such as decoding latency, throughput, and token acceptance rates. For qualitative insights, we provide a real-time terminal visualization that highlights accepted, rejected, and corrected tokens at each decoding step.

3 Acceptance Rate and Speed Comparison

3.1 Setup and Metrics

To systematically study decoding behavior, we evaluate how acceptance rate and speed vary with `max_new_tokens` and sampling temperature, visualized via a 2D heatmap. In-depth tests are conducted with `Qwen2.5-0.5B` as draft and `Qwen2.5-3B`, or `Qwen2.5-7B` as target to assess performance under high draft-to-target ratios.

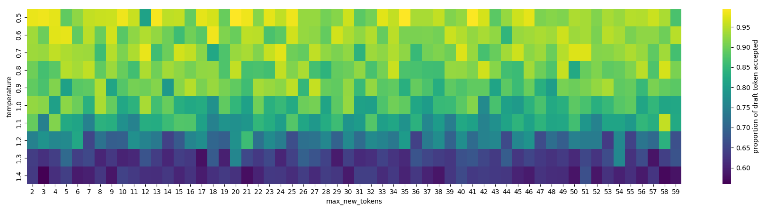

Figure 1: Proportion of draft token accepted per max new tokens and temperature

3.2 Acceptance rate comparison

We defined the acceptance rate as the number of draft tokens that pass the acceptance criterion (1). If a token isn't accepted because a previous token didn't pass the acceptance criterion, but he himself passes it, he will count towards the acceptance rate.

We can see the result of our tests in Figures 1 and 2.

In Figure 1, we used Qwen2.5-0.5B as the draft model and Qwen2.5-3B as the target model. We can see, as expected, that the proportion of draft tokens accepted goes down as we increase the temperature. This is the expected behavior as higher temperatures mean more random output that is more likely not to pass the acceptance criterion. We have a hard time really taking note of any trends regarding the max new tokens axis. That is why we chose to get more precise results that lead to Figure 2.

For Figure 2, we used Qwen2.5-0.5B as the draft model and Qwen2.5-7B as the target model, with a temperature set to 1. Each measurement was repeated 30 times to compute the mean and standard deviation. We observe that the average proportion of draft tokens slightly decreases over time. This trend likely reflects the tendency of smaller models to hallucinate, leading to increased divergence from larger models. However, these findings should be interpreted with caution due to the high standard deviation, indicating substantial variability. While a downward trend is present, it remains weak and uncertain.

3.3 Speed Comparison

For Figure 3, we used qwen2.5-0.5b as the draft model and qwen2.5-3b as the target model. We used a lower target model so as to be able to run a much higher number of experiments in an acceptable time. We tried to see the effect on very long generations from the draft model, as well as the effects of temperature. As the number of max new draft tokens increases, the performances worsen quickly, especially at high temperatures. A low temperature, which gives more deterministic models, performs better. Although it is less pronounced, a very low number of max new tokens also performs worse. It seems like the best and most consistent results are around a max new tokens of around 10 with a temperature of less than 1. It is also possible to have a good speed with a much higher number of max new tokens, but the results are way less consistent, and will also often perform worse.

A very low amount of max new tokens (around 2-3) don't perform as well as they aren't speculating enough for a big difference to be made, while a very high amount of max new tokens generates a large amount of wrong tokens which will often results in a large amount of unnecessary inference. A low temperature makes the models less deterministic, which makes the draft model and the target model less likely to agree on the same tokens.

One thing we didn't take into account in Figure 3 is the amount of tokens generated for each point. If more tokens are generated, it is normal that the model takes more time to generate them. We rerun the same experiment as in Figure 1, for a number of max tokens between 2 and 20, but this time while keeping track of the number of tokens generated. This way we could have the time taken per max new tokens and temperature divided by the amount of generated tokens. The results are in Figure 4. The results are very similar to the ones we got in Figure 3, but we have a bit less noise. This experiment confirms that the amount of tokens generated doesn't change the trends that we saw in Figure 3.

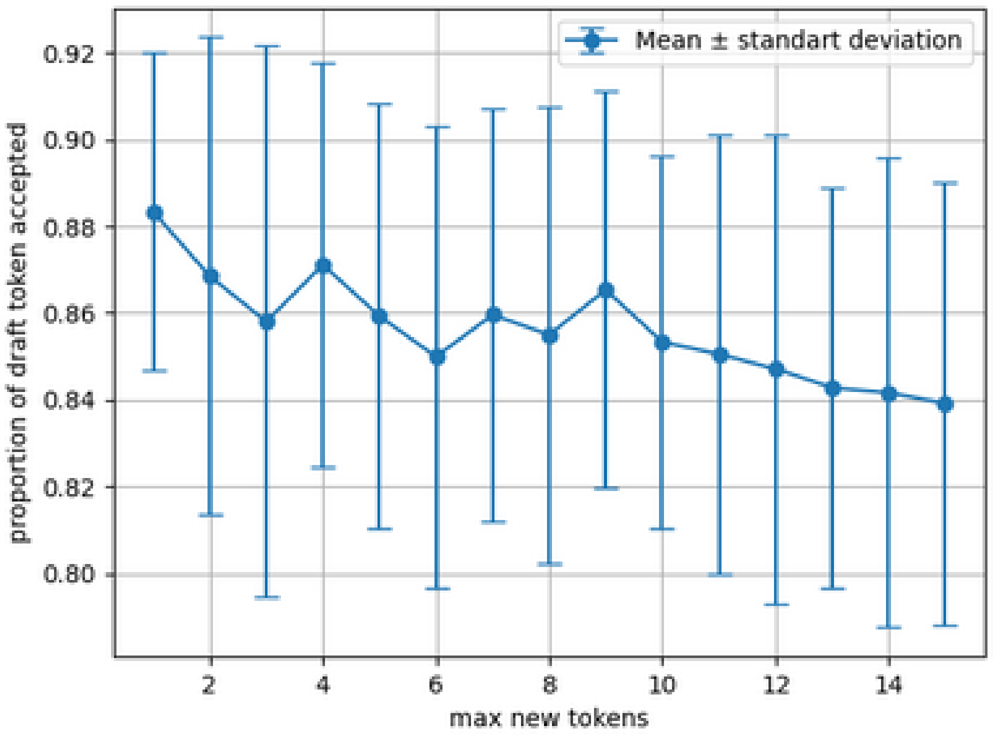

Figure 2: Proportion of draft token accepted per max new tokens

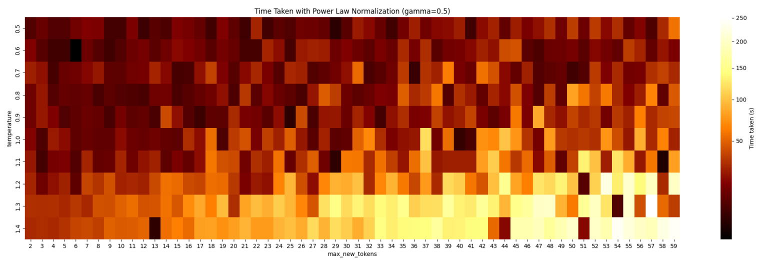

Figure 3: Time taken per max new tokens and temperature

4 Speculative and Classic Sampling Comparison

4.1 Experiments

4.1.1 Setup

To evaluate generation quality, we compute pass@k metrics on the HumanEval benchmark. For each prompt, we generate 10 completions using both classical sampling and speculative decoding. A solution is considered correct if it passes all unit tests. We report pass@1, pass@5, and pass@10 to assess both top-1 accuracy and diversity. All decoding runs share identical parameters for temperature, token limit, and stopping criteria to ensure a fair comparison. All models are executed using PyTorch with CUDA acceleration enabled. We perform two sets of experiments:

- **Experiment A:** Draft model is `Qwen2.5-0.5B`, target model is `Qwen2.5-1.5B`.
- **Experiment B:** Draft model is `Qwen2.5-0.5B`, target model is `Qwen2.5-3B`.

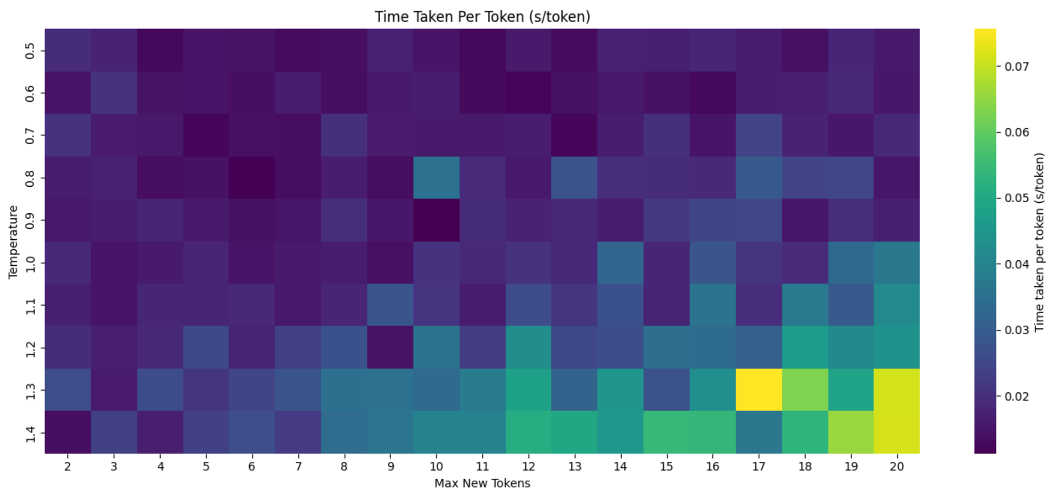

Figure 4: Time taken per max new tokens and temperature divided by the amount of generated tokens

For each prompt, we generate 10 completions using both classical and speculative decoding. All runs share identical parameters: maximum new tokens = 128, temperature = 0.7, top-p = 0.95.

4.1.2 Metrics

We evaluate both decoding efficiency and generation quality using the following metrics:

Latency is defined as the average time taken to generate one sample:

$$\text{Latency} = \frac{\text{Total time}}{\text{Number of samples}} \tag{2}$$

Throughput (TPS) measures the number of generated tokens per second:

$$\text{TPS} = \frac{\text{Total generated tokens}}{\text{Total time}} \tag{3}$$

pass@k measures the accuracy of generations using:

$$\text{pass@k} = 1 - \frac{\binom{n-c}{k}}{\binom{n}{k}} \quad \text{if } c \leq n - k \tag{4}$$

and $\text{pass@k} = 1$ if $c > n - k$, where n is the number of completions and c is the number of correct ones.

We report pass@1, pass@5, and pass@10 to assess both top-1 accuracy and diversity.

4.1.3 Benchmark: HumanEval

We evaluate the performance of both classical and speculative decoding using the **HumanEval** benchmark [4], a widely used dataset for code generation. It consists of 164 Python programming problems, each with a natural language prompt, a reference implementation, and corresponding unit tests.

We select HumanEval for the following reasons:

- It serves as a standard and reproducible benchmark for code generation tasks.
- Each sample includes unit tests, enabling automatic and objective evaluation of correctness.
- Its wide adoption in prior work allows meaningful comparisons with existing results.

For each problem, we generate **10 completions** using both decoding methods under the same settings (e.g., temperature, maximum tokens, stopping criteria). We compute **pass@k** metrics ($k = 1, 5, 10$), which estimate the likelihood that at least one of the k completions passes all unit tests.

This evaluation setup captures both top-1 accuracy and output diversity, making it particularly suitable for assessing functional correctness in code generation.

4.1.4 Evaluation Procedure

The evaluation is conducted as follows:

- **Classical sampling:** The target model generates tokens autoregressively using top-p sampling.
- **Speculative decoding:** The draft model generates $K = 3$ tokens per step, which are then verified or corrected by the target model using rejection sampling.

For each prompt:

- Both methods generate 10 completions.
- We record generation time and number of tokens for throughput and latency analysis.
- Results are averaged over all prompts and across 4 independent runs to ensure robustness.

4.1.5 Visualization

We visualize the results from both experiment settings using:

- A **line chart** showing pass@k (1, 5, 10) across runs.
- A **bar chart** comparing average latency and throughput between classical and speculative decoding.

These comparisons allow us to quantify the performance trade-offs under different draft-to-target scaling ratios and provide evidence of the robustness and effectiveness of speculative decoding.

4.2 Results and Discussion

We evaluate speculative decoding under two settings: **Experiment A**, where the draft–target pair is `Qwen2.5-0.5B` $\rightarrow$ `Qwen2.5-1.5B`, and **Experiment B**, where the target model is scaled up to `Qwen2.5-3B`. We analyze both decoding efficiency (latency and throughput) and generation quality (pass@k metrics), and further discuss the effects of draft–target size gap on performance.

4.2.1 Experiment A: 0.5B Draft → 1.5B Target

In this setting, speculative decoding demonstrates significant speed advantages over classical sampling. As shown in Figure 5, it achieves up to **2.8× lower latency** per token and **3.5× higher throughput**. These results indicate effective amortization of target model cost with modest model size gap.

In terms of accuracy, speculative decoding shows a moderate trade-off when compared to classical sampling at the 1.5B target scale. Figure 6 illustrates a **slight decrease of approximately 2% in pass@1** when using speculative decoding. Similarly, pass@5 and pass@10 also show marginal drops of roughly 4% and 6%, respectively. These results suggest that while speculative decoding maintains reasonable output quality at this scale, it may introduce slight degradation in correctness due to early-stage token rejection or over-speculation.

4.2.2 Experiment B: 0.5B Draft → 3B Target

With a larger target model, the benefits of speculative decoding become even more pronounced. As seen in Figure 7, latency is reduced by up to **3.75×**, and throughput increases by **4.23×** over classical decoding. This performance boost highlights the scalability of speculative decoding as the target model grows, since the overhead of verification is amortized more effectively. Accuracy results in this setting show a moderate decline when using speculative decoding. As illustrated in

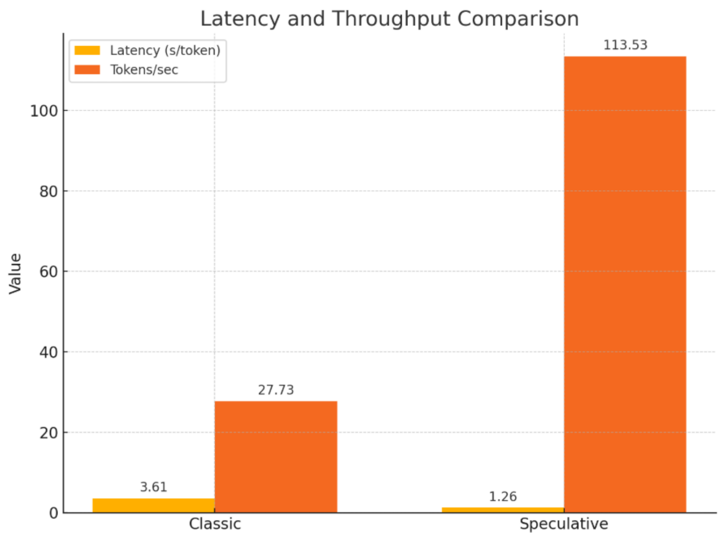

Figure 5: Latency and throughput comparison for Experiment A (0.5B $\rightarrow$ 1.5B).

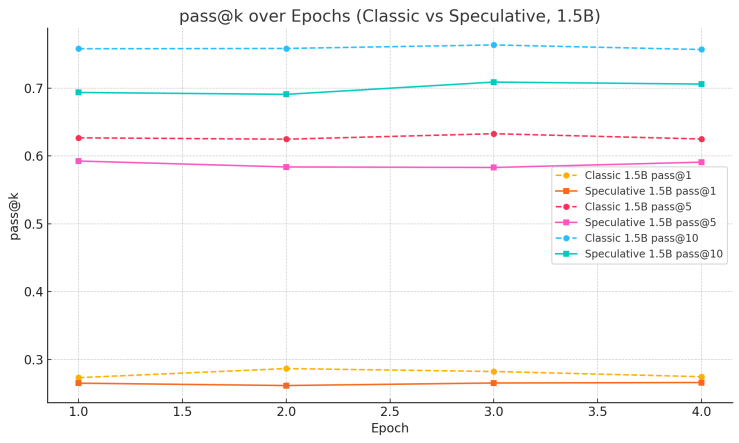

Figure 6: Pass@k comparison for Experiment A.

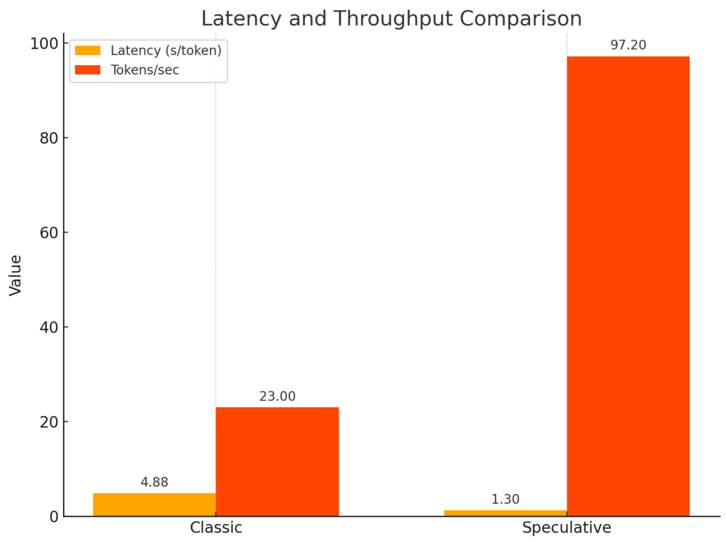

Figure 7: Latency and throughput comparison for Experiment B (0.5B $\rightarrow$ 3B).

Figure 8, classical sampling outperforms speculative decoding across all pass@k metrics. Specifically, pass@1 decreases by approximately **1.7% absolute**, pass@5 drops by around **5.9%**, and pass@10 declines by nearly **10%**. These results suggest that while speculative decoding still benefits from improved latency and throughput, it introduces a quality trade-off at the 3B scale, possibly due to over-aggressive token acceptance from the smaller draft model.

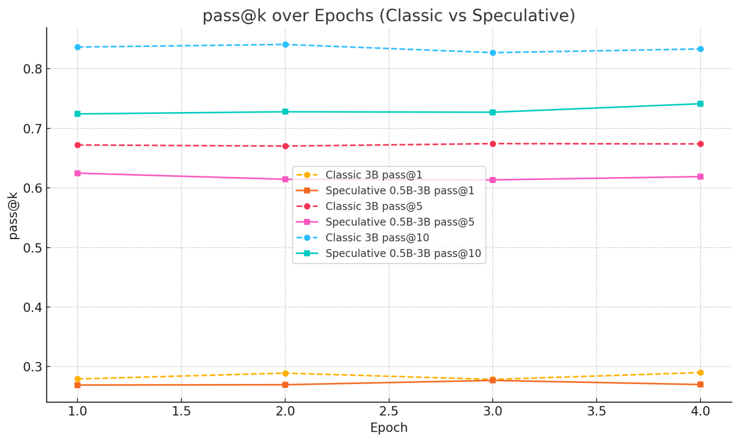

Figure 8: Pass@k comparison for Experiment B.

4.2.3 Summary of Findings

Overall, speculative decoding provides substantial gains in decoding speed, with up to 3.75× lower latency and 4.2× higher throughput observed in our experiments. These improvements are consistent across both model scales (1.5B and 3B targets), and become more pronounced when the target model is larger, as seen in Experiment B. This supports the theoretical premise that speculative decoding better amortizes verification cost when the target model is slower but more accurate.

In terms of accuracy, our results show that speculative decoding incurs a modest trade-off. While generation quality remains acceptable, we observe consistent declines of 2–6% in pass@1 and pass@5 scores compared to classical sampling, particularly when the draft model is weak relative to the target. This suggests that early-stage draft errors are not always fully corrected by the target model, especially in precise tasks such as code generation.

Nonetheless, speculative decoding remains valuable as a high-efficiency inference strategy. It allows deployment of smaller draft models with strong targets while significantly reducing response time. For practitioners, these results highlight the importance of tuning parameters like draft length and temperature, and balancing quality against throughput based on application needs.

5 3-model Speculative Sampling

We wanted to test if it was possible to improve on the current speculative sampling method. For that, we thought of adding a third model to speculative sampling. A draft model creates a first draft for a certain number of tokens, and an intermediary model then checks the quality of the draft. After a certain amount of verification by the intermediary model, the target model, which is the largest of the three, verifies the tokens generated so far.

Algorithm 2 3-model Speculative Decoding

```
Input  : Prompt $x_{1:t}$, draft model $p_\phi$, intermediary model $p_\psi$, target model $p_\theta$, draft length $k_d$,
         intermediary cycles $k_i$
Output : Generated output sequence $x$
10 Initialize $x \leftarrow x_{1:t}$ while not finished do
11     Initialize empty batch sequence $x_{\text{batch}}$ /*Intermediary Verification Cycles */
12     for $j = 1$ to $k_i$ do
13         Let all_drafts_accepted ← true  Sample $k_d$ draft tokens $\tilde{x} \sim p_\phi$ (conditioned on context)
             /*Draft Generation */
14         for $l = 1$ to $k_d$ do
15             Compute acceptance probability $\alpha_l = \frac{p_\psi(\tilde{x}_l|x,\tilde{x}_{1:l-1})}{p_\phi(\tilde{x}_l|x,\tilde{x}_{1:l-1})}$ if accepted with probability
                 $\min(1, \alpha_l)$ then
16                 Append $\tilde{x}_l$ to $x_{\text{batch}}$
17             else
18                 all_drafts_accepted ← false  Resample token $x'_l \sim p_\psi$ and append to $x_{\text{batch}}$ break
                     /*End draft generation for this cycle */
19         if all_drafts_accepted then
20             Sample and append one extra token from $p_\psi$ to $x_{\text{batch}}$
       /*Target Model Verification */
21     Let $N$ be the number of tokens in $x_{\text{batch}}$  Let all_batch_accepted ← true  for $m = 1$ to $N$ do
22         Let $y_m$ be the $m^{th}$ token in $x_{\text{batch}}$  Compute acceptance probability $\beta_m = \frac{p_\theta(y_m|x,y_{1:m-1})}{p_\psi(y_m|x,y_{1:m-1})}$ if
             accepted with probability $\min(1, \beta_m)$ then
23             Append $y_m$ to final sequence $x$
24         else
25             all_batch_accepted ← false  Resample token $x''_m \sim p_\theta$ and append to $x$  break /*End
                 target verification and start new main loop */
26     if all_batch_accepted then
27         Sample and append one extra token from $p_\theta$ to $x$
28 return $x$
```

We tested its performance on the HumanEval benchmark to see if it has the same performance as classical sampling. As we can see in Figure 9, the performances are slightly worse.

We tested the 3-model speculative sampling's speed per token, with $k_d = 5$ and $k_i = 3$, against speculative sampling with a max token of 5, and classical sampling. We repeated the measurements 10 times to get the mean and a standard deviation. The results can be seen in Figure 10. As expected, speculative sampling outperforms classic sampling. However, the 3-model speculative sampling is performing worse. This could be due to the high standard deviation of the results. It is also possible that if we tried different combinations of parameters, some would make the 3-model speculative sampling worth it. There might be also However, with our current results, we can't affirm that 3-model speculative sampling is a good alternative to current speculative techniques.

6 Conclusion and Future Work

This study evaluates speculative decoding on the Qwen2.5 model family as a method for accelerating autoregressive generation in large language models. Our experiments on the HumanEval benchmark confirm that speculative decoding significantly improves decoding efficiency, achieving up to **3.75**× lower latency and **4.23**× higher throughput relative to classical sampling. These speedups are particularly evident when using a lightweight draft model paired with a larger target.

In terms of accuracy, speculative decoding shows a trade-off. Contrary to prior assumptions of quality preservation, our results reveal slight declines in pass@1 and pass@5—typically in the range of 2–6%—especially when the draft model is substantially weaker. Nevertheless, overall generation

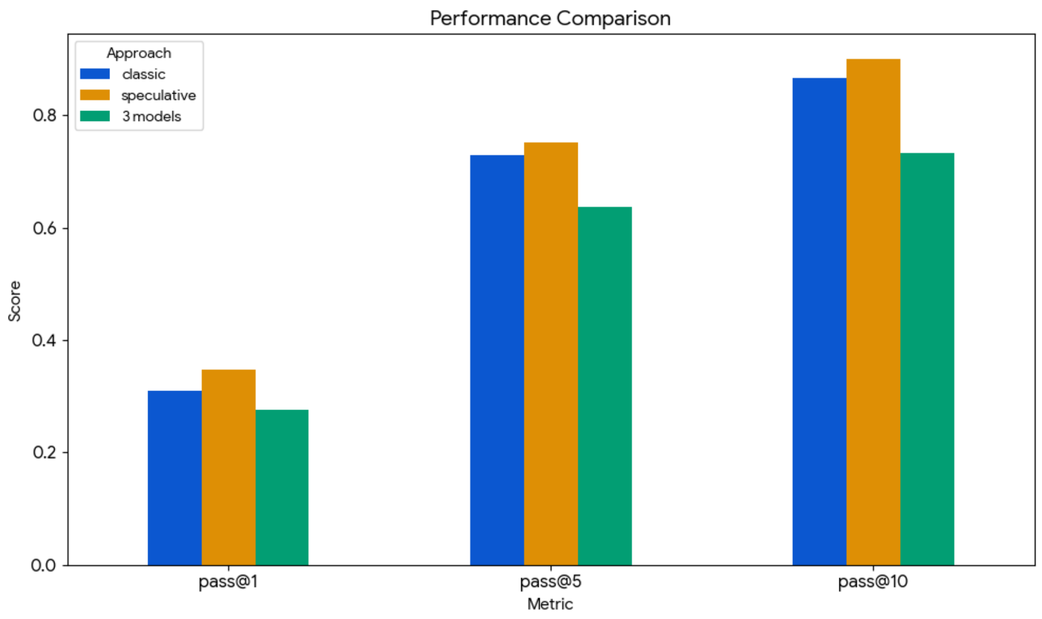

Figure 9: Comparison of classical sampling, speculative sampling, and 3-model speculative sampling on HumanEval

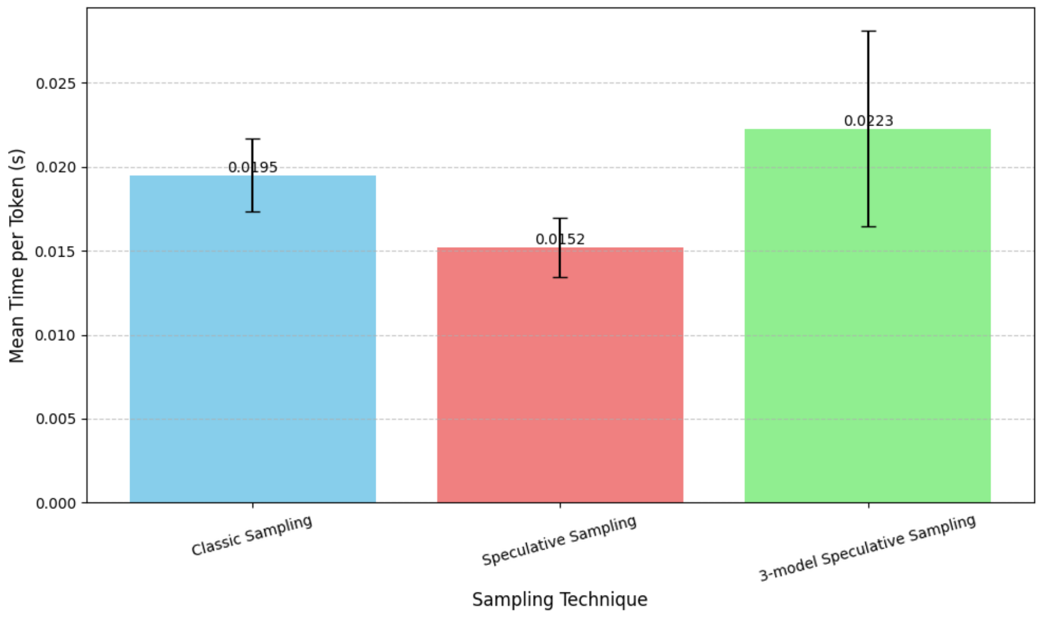

Figure 10: Mean time per token with standard deviation for different sampling techniques

quality remains acceptable for many practical scenarios, and the decoding process benefits from the target model's ability to correct obvious draft errors.

Taken together, these findings support speculative decoding as an effective speed-accuracy trade-off mechanism for high-throughput inference. Its ease of integration—without requiring model retraining or architectural changes—makes it especially attractive for deployment in latency-sensitive or resource-limited environments.

6.1 Limitations

Our study has several limitations. First, our implementation of speculative decoding improves throughput at the cost of a slight reduction in task accuracy, reflecting a fundamental speed-quality trade-off that we acknowledge but do not attempt to resolve with more advanced adaptive mechanisms. Furthermore, the simultaneous loading of both a draft and a target model significantly increases memory requirements compared to classical decoding, which can pose a practical limitation for deployment on resource-constrained hardware. Finally, our evaluation is confined to the task of code generation using the HumanEval benchmark and the Qwen2.5 model family; the generalizability of these findings to other domains, such as open-domain dialogue or text summarization, or to other model architectures has not been verified.

Future Work

Future work will focus on enhancing speculative decoding's adaptability. This includes introducing dynamic draft length scheduling based on uncertainty, and adjusting acceptance thresholds in response to token-level confidence. To address the observed accuracy trade-off, future approaches may also explore more conservative verification strategies, use stronger draft models, or incorporate lightweight intermediary models for hierarchical validation. Another promising direction is to extend speculative decoding to multilingual, multimodal, and instruction-tuned models, where generation behaviors are more complex. Finally, combining speculative decoding with model compression techniques like quantization or with retrieval-augmented generation pipelines could further improve performance and scalability in real-world systems.

References

[1] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] R. Anil, A. M. Dai, O. Firat, M. Johnson, D. Lepikhin, A. Passos, S. Shakeri, E. Taropa, P. Bailey, Z. Chen, et al. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*, 2023.

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

[4] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. D. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[5] Google Research. Looking back at speculative decoding, 2024. `https://research.google/blog/looking-back-at-speculative-decoding/`.

[6] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks. *Advances in neural information processing systems*, 29, 2016.

[7] B. Hui, J. Yang, Z. Cui, J. Yang, D. Liu, L. Zhang, T. Liu, J. Zhang, B. Yu, K. Lu, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

[8] Y. Leviathan, M. Kalman, and Y. Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR, 2023.

[9] A. Narayanan, S. Verma, E. Ramadan, P. Babaie, and Z.-L. Zhang. Deepcache: A deep learning based framework for content caching. In *Proceedings of the 2018 Workshop on Network Meets AI & ML*, pages 48–53, 2018.

[10] A. Polino, R. Pascanu, and D. Alistarh. Model compression via distillation and quantization. *arXiv preprint arXiv:1802.05668*, 2018.

[11] A. S. Ross, M. C. Hughes, and F. Doshi-Velez. Right for the right reasons: Training differentiable models by constraining their explanations. *arXiv preprint arXiv:1703.03717*, 2017.

[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.