CopySpec: Accelerating LLMs with Speculative Copy-and-Paste

Razvan-Gabriel Dumitru[1] **Minglai Yang**[1] **Vikas Yadav**[2] **Mihai Surdeanu**[1]
[1]University of Arizona [2]ServiceNow Research
razvandumm@gmail.com
{mingly, msurdeanu}@arizona.edu
vikas.yadav@servicenow.com

Abstract

We introduce *CopySpec*, a simple yet effective technique to tackle the inefficiencies LLMs face when generating responses that closely resemble previous outputs or responses that can be verbatim extracted from context. *CopySpec* identifies repeated sequences in the model's chat history or context and speculates that the same tokens will follow, enabling seamless copying without compromising output quality and without requiring additional GPU memory. To evaluate the effectiveness of our approach, we conducted experiments using seven LLMs and five datasets: MT-Bench, CNN/DM, GSM8K, HumanEval, and our newly created dataset, *MT-Redundant*. *MT-Redundant*, introduced in this paper, transforms the second turn of MT-Bench into a request for variations of the first turn's answer, simulating real-world scenarios where users request modifications to prior responses. Our results demonstrate significant speed-ups: up to 2.35× on CNN/DM, 3.08× on the second turn of select *MT-Redundant* categories, and 2.66× on the third turn of GSM8K's self-correction tasks. Importantly, we show that *CopySpec* integrates seamlessly with speculative decoding, yielding an average 49% additional speed-up *over* speculative decoding for the second turn of *MT-Redundant* across all eight categories. While LLMs, even with speculative decoding, suffer from slower inference as context size grows, CopySpec leverages larger contexts to accelerate inference, making it a faster complementary solution. Our code and dataset are publicly available at https://github.com/RazvanDu/CopySpec.

1 Introduction

Large Language Models (LLMs) have revolutionized natural language processing (NLP), enabling great performance across a range of applications, including code generation, machine translation, and question answering. However, the computational demands of LLMs, particularly during inference, pose significant challenges for real-time applications and scalability in resource-constrained environments. Sequential token generation, a core bottleneck in standard decoding, limits throughput and increases latency. Speculative Decoding (Leviathan et al., 2022; Chen and Xu, 2023) has emerged as a promising approach to mitigate this issue by employing a smaller draft model to generate multiple token sequences, which are then verified by the larger target model. Despite its potential, existing speculative decoding methods often fail to fully exploit the inherent redundancies in previously LLM-generated outputs and context. For example, in tasks such as summarization, retrieval augmented generation (RAG), and multi-turn conversations, etc., response generations include several text pieces verbatim from the context. In tasks such as code fixing, error can be just from a specific line of code which requires editing and regeneration while large portion of the code can be

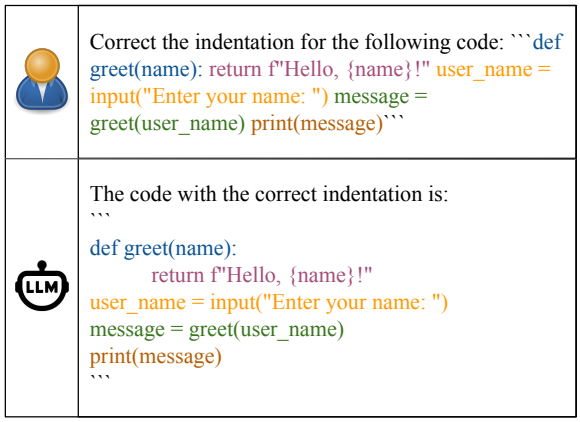

Figure 1: An example of redundant information, represented by blocks of the same color, that can be directly copied during inference without re-computation. This highlights the potential of our approach to make inference more efficient by leveraging repeated information, reducing computational overhead, and improving speed.

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 26313–26344
November 4-9, 2025

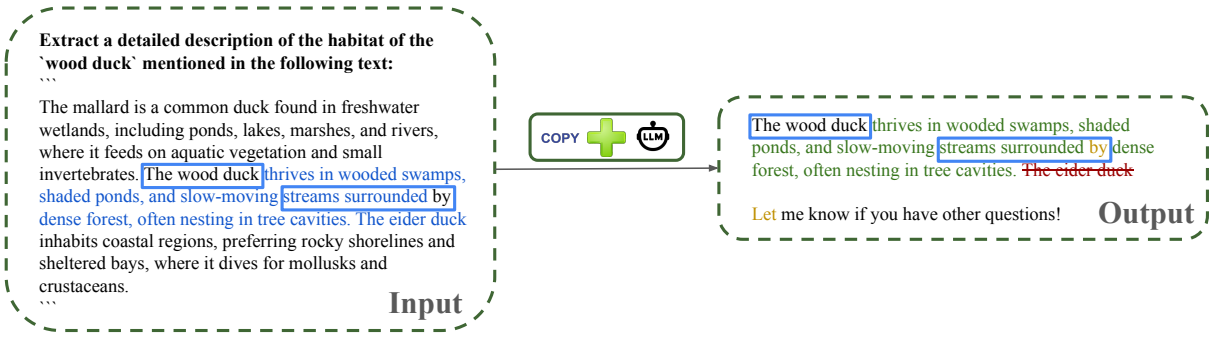

Figure 2: The figure illustrates the speculative copying process, *CopySpec* applied to extract the habitat description of the "wood duck.". The input text provides the context and instructions. During generation, the system identifies sequences of 3 consecutive tokens (we use words as tokens here for illustrative simplicity) that repeat within the input. The blue rectangle in the input highlights the matching token sequence detected, which serves as the starting point for speculative copying. From this match, the next 10 tokens are copied into the output. In the output, the copied tokens are shown in blue and validated through speculative copying. Tokens accepted by the model are highlighted in green, continuing the description seamlessly, while rejected tokens are shown in red with a strikethrough. Extra tokens generated during the validation process are marked in yellow/gold, demonstrating how the model extends the copied content as needed. This figure demonstrates how *CopySpec* efficiently leverages repeated sequences to enhance text generation accuracy and speed by integrating both copied and dynamically generated content. Accepted tokens are exactly those the target model would generate next under the same policy, so the user-visible text is unchanged relative to standard decoding.

copy-pasted as it is. Further, only small edits are required in previous generation for frequent queries such as format changes, persona or style transfer, etc and in case of improving LLM outputs such as self-verification.

In this work, we present *CopySpec*, a mixed speculative decoding framework designed to exploit redundancies or repeated generations from previous iterations of responses or verbatim text available in context. *CopySpec* incorporates a copying mechanism into the draft process, enabling the model to detect and exploit predictable patterns in token sequences (see Figure 2 for a summary of the approach). By inferring subsequent tokens directly from prior context, *CopySpec* reduces the computational burden by simply copy pasting the repeating response text instead of generating it. Importantly, *CopySpec* can be combined with any speculative decoding method to further improve LLM inference speed. Basically, *CopySpec* complements any existing speculative decoding method by simply copy pasting text where repeated text can be copied from context, or else response is generated by the draft model. Our experiments on various benchmarks—including HumanEval (Chen et al., 2021), CNN/DM (See et al., 2017), GSM8K (Cobbe et al., 2021), and MT Bench (Zheng et al., 2023)—demonstrate that CopySpec delivers up to an additional 49% speed-up over speculative decoding, without compromising output quality. In fact, with greedy decoding (T=0), the final text is identical to standard decoding, since only tokens matching the target model's decision are accepted and all others are rolled back.

2 Related Work

2.1 Complementing Speculative Decoding with Copying Mechanisms

Recent work has advanced speculative decoding through multi-token verification (Leviathan et al., 2022; Cai et al., 2024), adaptive pipelines (Liu et al., 2024), token tree structures (Miao et al., 2024), pipelined execution (Yang et al., 2024b), draft–target distillation (Zhou et al., 2024), and retrieval-based validation (He et al., 2024), with further gains from optimal transport (Sun et al., 2024) and early-layer reuse (He and Wang, 2023), as well as decoding-time long-context mitigation via positional contrastive decoding (PCD) (Xiao et al., 2025).

Complementary efforts accelerate decoding via prompt reuse: PLD and PLD+ copy prompt n-grams (Saxena, 2023; Somasundaram et al., 2025), PPD overlaps streams for GPU efficiency (Chen et al., 2024), look-ahead decoding verifies disjoint blocks (Fu et al., 2024), and token recycling repur-

poses failed tokens (Luo et al., 2025). The closest to our method is speculative decoding with suffix alignment (Hu et al., 2024), though it introduces significantly more algorithmic complexity and overhead.

CopySpec complements all of the above with minimal overhead. Instead of restricting reuse to prompt-boundary n-grams, it uses a rolling hash to find any repeated γ-token suffix in context and speculates on the next block. These candidate tokens are verified directly, enabling integration with any speculative decoding setup and yielding additional speedups without compromising output quality.

2.2 Copying Mechanisms in Language Models

Copying mechanisms are widely adopted in NLP to handle tasks that require replicating predictable patterns or segments. Gu et al. (2016) introduced CopyNet, a method that enables RNN sequence-to-sequence models to predict words based on a mixed probabilistic model of two modes, where one selects words from the source sequence. Similarly, in summarization tasks, Pointer Networks (Vinyals et al., 2015) and Pointer-Generator Networks (See et al., 2017) demonstrated the effectiveness of combining copying and generation to improve output fidelity and handle out-of-vocabulary tokens.

In line with motivations from previous approaches that have emphasized the importance of copying mechanisms in various applications, our work provides the simplest and most effective method for LLM inference. *CopySpec* integrates a copying mechanism into speculative decoding, effectively reducing redundancy and enhancing efficiency across a wide range of tasks. By leveraging repeated patterns in the model's context, *CopySpec* introduces a novel approach to accelerate inference while maintaining high performance.

3 Method

Our method operates on the assumption that if the last γ tokens generated by an LLM appear in the context, the tokens that followed them in the input context are likely to follow again in the output. Figures 1 and 2 illustrate this concept, By accurately identifying the start of such a segment, we can simply copy-paste all tokens within the block in a single pass, bypassing the need for a draft model to produce them incrementally. CopySpec accepts a proposed token iff it equals the token the target model would produce next under that policy; otherwise it rejects and truncates the KV cache to the accepted prefix. (Appendix B) Therefore, for any prompt, CopySpec's output is token-for-token identical to the baseline decoder under the same policy. In the following subsections, we detail the implementation of this approach and its integration into a Speculative Decoding framework, demonstrating how it achieves substantial speed-ups.

3.1 Identifying the Tokens to Copy

To efficiently detect when the model begins generating a block that has already been produced earlier, we maintain a hash map containing all subsequences of γ tokens from the context. During the generation process, we search this hash map for matches to the last γ tokens generated. Adding a new tuple of tokens to the hash map and searching for a match after each generated token has a time complexity of $O(\gamma)$. Since γ is typically set to a small value (e.g., 3 or 5), the computational overhead for processing new tokens and finding matches is minimal and independent of the context size. This stands in contrast to alternative approaches that require searching the entire context for the last substring, which can become computationally expensive as the context grows.

Our technique efficiently leverages larger contexts, allowing inference to become faster as the context size increases. By keeping γ fixed, we ensure a balance between efficiency and precision. Additionally, we explored methods to utilize partial outputs without revealing the complete results and investigated how the semantic relationship between the preceding γ tokens and the subsequent token can guide the optimal choice of γ. Further details are provided in Appendix A.

3.2 Speculating on the Matched Tokens

After identifying a match of γ tokens in the context, we extract the subsequent tokens from the context, as shown in Figure 2. These extracted tokens, which we call $S_{copyspec}$, essentially simulate the behavior of a draft model where the probability for each token in $S_{copyspec}$ is treated as 100%. But instead of generating, *CopySpec* pastes the output from context, thus avoiding the expensive computation of generation by draft model. [1]

$S_{copyspec}$ is then verified directly by the main LLM. Each verification yields τ tokens that align with the LLM's ongoing generation, along with one

[1] If multiple matches exist for the last γ tokens, we simply select the first, though more efficient strategies may exist.

Model *(Instruct)*	Variant	Metric	MT-Redundant *0-shot* GPT-4 Score (↑)	CNN/DM *0-shot* ROUGE-L (↑)	GSM8K *3-turn* Accuracy (↑)	MT-Bench *0-shot* GPT-4 Score (↑)	HumanEval *0-shot* Accuracy (↑)
Qwen2.5-72B	CopySpec	Tokens/Sec	**6.42**±0.01	**8.68**±0.01	**7.01**±0.01	**5.55**±0.01	**7.01**±0.01
	Base model	Tokens/Sec	4.82±0.01	3.70±0.01	4.55±0.01	4.83±0.01	4.98±0.01
		Score	9.28	0.213	96%	9.18	87.8%
Qwen2.5-32B	CopySpec	Tokens/Sec	**13.82**±0.01	**18.34**±0.03	**14.84**±0.01	**12.15**±0.01	**14.41**±0.01
	Base model	Tokens/Sec	10.26±0.01	7.79±0.01	9.76±0.01	10.29±0.01	10.46±0.01
		Score	9.10	0.214	93%	8.97	89.6%
Qwen2.5-7B	CopySpec	Tokens/Sec	**54.05**±0.11	**47.15**±0.08	**63.37**±0.54	**46.85**±0.08	**48.79**±0.01
	Base model	Tokens/Sec	39.88±0.02	25.25±0.05	38.58±0.03	39.98±0.01	33.63±0.06
		Score	8.53	0.230	85%	8.41	82.3%
Llama3.1-70B	CopySpec	Tokens/Sec	**6.57**±0.01	**5.49**±0.01	**6.06**±0.01	**5.83**±0.01	**6.24**±0.01
	Base model	Tokens/Sec	4.98±0.01	4.19±0.01	4.77±0.01	4.98±0.01	5.05±0.01
		Score	8.74	0.204	90%	8.72	77.4%
Llama3.1-8B	CopySpec	Tokens/Sec	**49.28**±0.08	**37.44**±0.19	**49.60**±0.01	**45.84**±0.07	**46.49**±0.48
	Base model	Tokens/Sec	35.51±0.01	26.57±0.11	35.19±0.09	35.43±0.01	37.57±0.22
		Score	8.03	0.185	79%	7.54	65.9%

Table 1: Performance comparison across five instruct models (Qwen2.5-72B, Qwen2.5-32B, Qwen2.5-7B, Llama3.1-70B, and Llama3.1-8B) using CopySpec versus baseline configurations on multiple datasets, including MT-Redundant, CNN/DM, GSM8K, MT-Bench, and HumanEval. Metrics include model-specific scores (GPT-4, using the 0613 checkpoint: Score, ROUGE-L, Accuracy), and token generation rates (tokens/sec). Results demonstrate the effectiveness of CopySpec in enhancing computational efficiency without compromising quality, achieving notable speed-ups and high token-copying rates in diverse tasks and model sizes.

additional guaranteed token. This approach mirrors vanilla speculative decoding (Leviathan et al., 2022), where speculative tokens are appended to the context, and the longest prefix matching the LLM's output is accepted. In Figure 2, $S_{copyspec}$ is highlighted in blue. The output shows the τ accepted tokens in green, the extra guaranteed token in gold, and any rejected tokens in red.

After each newly pasted or generated token, or copying attempt, we re-evaluate the last γ tokens in the context to identify a new match, allowing the model to utilize longer copyable blocks whenever possible. This eliminates the need for manual token generation between copying steps.

If any tokens in $S_{copyspec}$ fail the verification step, the model generates a new token that diverges from the previously matched tokens. This ensures that the next copying attempt yields a different match, preventing the model from getting stuck in repetitive loops.

3.3 Merging with Speculative Decoding

To further enhance our technique, we have integrated it within a vanilla Speculative Decoding framework. At each step of the generation process, we attempt to find matches in the context. If a match for the last γ tokens is found, we use $S_{copyspec}$ as draft tokens, effectively simulating a draft model with perfect confidence in those tokens. If no match is identified, we rely on a smaller draft model to generate τ_2 draft tokens. This dual approach allows us to dynamically choose between leveraging repetitive patterns and utilizing speculative decoding for efficient token generation in contexts with little or no redundancy.

This integration provides the best of both worlds: Speculative Decoding accelerates inference when the context size is small or lacks redundancy, while CopySpec builds on this speed-up in subsequent steps by taking advantage of repetitive patterns as the context size increases.

It is also worth noting that when used as a standalone method, CopySpec does not require a draft model. This eliminates the need for additional GPU memory or modifications to the model, making it lightweight and easy to deploy. We also explore the interplay between these techniques in Section 6, while Section B provides a detailed account of the full implementation, including key-value caching. Furthermore, we mainly focus on the integration with vanilla speculative decoding(Leviathan et al., 2022) in the paper for generalizability but we also integrate it as part of the EAGLE(Li et al., 2025) framework and compare it against baselines in Appendix C to showcase the technique's potential.

4 Experiments

4.1 Models and Hyperparameters

We evaluated our technique on five instruction-tuned LLMs: Qwen2.5-72B, 32B, 7B (Yang et al., 2024a), LLaMa3.1-70B, 8B (Grattafiori et al., 2024), Vicuna-v1.3-13B, 7B(Zheng et al., 2023), using 4 A100 GPUs with a batch size of 1 and temperature 0. Unless stated, γ =3, $|S_{copyspec}|$ =10, and max generation length to 1024.

4.2 Evaluation Datasets

We evaluated our technique on five datasets, each targeting specific aspects of model performance: MT-Redundant, CNN/DM, GSM8K, MT-Bench, and HumanEval. MT-Redundant was designed to emphasize prompts requiring small variations to previous outputs, while CNN/DM focuses on extractive summarization. GSM8K evaluates the model's self-correction capabilities, MT-Bench highlights scenarios with minimal copying potential to measure the technique's overhead, and HumanEval assesses coding capabilities. To accommodate the increased computational demands of GSM8K and CNN/DM and our limited GPU resources, we restricted these datasets to 100 samples, ensuring they were of comparable size to the other datasets. For HumanEval, we employed the same instruction format as presented in EvalPlus (Liu et al., 2023). Detailed descriptions of all prompts used in our experiments are provided in Appendixes H and G.

4.3 Synergy with External Drafters

A detailed head-to-head between CopySpec and the strongest publicly available accelerators—EAGLE (Li et al., 2025), PLD (Somasundaram et al., 2025) and SAM-D (Hu et al., 2024)—appears in Appendix C. The headline figures (Table 7) show that combining EAGLE with span-level copying lifts Vicuna-7B throughput on MT-Redundant from $82 \rightarrow 95$ TPS, a *2.4*$\times$ boost over the greedy decoder.

4.4 MT-Redundant

Most existing NLP datasets focus on tasks involving either single-turn interactions or scenarios where the model must entirely change its response in the second turn. These setups fail to capture realistic use cases where a user might request slight variations or refinements to a previous answer. To address this gap and highlight the capabilities of our technique, we introduce a new dataset, *MT-Redundant*.

MT-Redundant is derived by modifying the second turn of MT-Bench (Zheng et al., 2023). In our dataset, the second turn replaces the original question with a prompt asking the model to review its previous answer and make specific adjustments or variations. This modification simulates real-world scenarios where incremental refinement or elaboration is required. Example prompts from the dataset are provided in Appendix G. For questions with reference answers, we retained the original reference for the first turn and created a new reference answer for the second turn to align with the revised prompts.

Our dataset spans a diverse range of practical use cases, categorized into eight groups: Coding, Extraction, Humanities, Math, Reasoning, Role-play, STEM, and Writing. These categories reflect realistic tasks encountered in various domains. Additionally, we adopted the same evaluation procedure from MT-Bench to ensure consistency and comparability of results.

By creating *MT-Redundant*, we aim to bridge the gap between artificial benchmarks and practical applications, providing a more representative evaluation for techniques like CopySpec in multi-turn interactions with repetitive information.

5 Discussion of Results

We analyze our main results in Table 1 and Table 2, which show the impact of our method on performance and the percentage of tokens copied across five LLMs and datasets. The results are aggregated for all turns in MT-Redundant and MT-Bench (two turns each) and the self-correction process in GSM8K (three turns). Speedups range from 1.15$\times$ on MT-Bench, which has minimal redundancy, using Qwen2.5-72B-Instruct, to 2.35$\times$ on CNN/DM.

While these results are notable, the key strength of our approach is its ability to enhance performance as the context grows. To illustrate this, we look at per-turn performance and analyze the effect of varying hyperparameters on the technique's effectiveness in a wide range of use-cases.

5.1 Speed-up by Turn and Category

We begin our analysis by examining the speedups achieved on MT-Redundant for both the first and second turns, as summarized in Table 3. The results

Model	MT-Redundant	CNN/DM	GSM8K	MT-Bench	HumanEval
Qwen2.5-72B	32.35%	82.48%	47.59%	20.53%	37.47%
Qwen2.5-32B	33.17%	81.82%	44.93%	22.61%	34.23%
Qwen2.5-7B	34.42%	65.67%	53.01%	22.86%	32.68%
Llama3.1-70B	31.42%	38.35%	30.07%	21.83%	27.54%
Llama3.1-8B	35.45%	38.32%	38.01%	30.01%	26.44%

Table 2: For each dataset, the share of tokens in the final model output that CopySpec successfully copied, expressed as a percentage of the total number of tokens generated.

	Turn 1		Turn 2	
Category	**Base Model**	**CopySpec**	**Base Model**	**CopySpec**
Coding	5.12 ±0.01	**5.62** ±0.01	4.61 ±0.01	**9.33** ±0.01
Extraction	4.76 ±0.01	**5.65** ±0.01	4.58 ±0.01	**8.30** ±0.01
Humanities	5.09 ±0.01	**5.33** ±0.01	4.55 ±0.01	**5.45** ±0.01
Math	5.17 ±0.01	**5.84** ±0.01	4.75 ±0.01	**10.14** ±0.01
Reasoning	5.08 ±0.01	**5.69** ±0.01	4.65 ±0.01	**10.84** ±0.01
Roleplay	5.08 ±0.01	**5.14** ±0.01	4.58 ±0.01	**14.10** ±0.03
Stem	5.12 ±0.01	**5.37** ±0.01	4.61 ±0.01	**6.78** ±0.01
Writing	5.12 ±0.01	**5.13** ±0.01	4.65 ±0.01	**10.59** ±0.01
Average	5.07 ±0.01	**5.47** ±0.01	4.62 ±0.01	**9.44** ±0.01

Table 3: Comparison of model speeds measured in tokens/sec across two turns on MT-Redundant using CopySpec (Qwen2.5-72B-Instruct, $\gamma = 3$). The technique leads to an overall speed-up of 2.04 on the second turn.

indicate a substantial average speedup of 2.04× for the second turn, compared to a more modest speedup of 1.08× for the first turn. Notably, the performance in tokens per second (TPS) achieved by the model increases for the second turn, which features a larger context size. In contrast, the baseline model experiences a decline in TPS as the context size increases. Another notable aspect is that the observed speedup is highly dependent on the specific use case. For instance, we observe speedups as low as 1.2× in the Humanities category and as high as 3.08× for Roleplay. However, regardless of the use case, the speedup for the second turn remains consistently positive across all models for both MT-Redundant and MT-Bench.

The results for all five models on MT-Redundant and MT-Bench are detailed in Appendix D.2 and E.2 respectively. On average, the second round of MT-Redundant achieves a significant 91% speedup across all models, compared to 31% for MT-Bench. Notably, even on MT-Bench, which has less redundancy, the TPS achieved by CopySpec in the second turn is almost always higher than the baseline model's TPS in the first turn.

5.2 The Effect of Gamma (γ)

We begin our analysis with Figure 3, which illustrates the tokens per second as a red line, alongside the percentage of tokens copied out of the total tokens generated, represented by a blue line for the LLaMa3.1-8B-Instruct model on HumanEval. The numbers adjacent to the dots indicate the number of attempts made to copy tokens. The figure demonstrates that as γ increases, a smaller percentage of tokens is accepted, but the number of copying attempts decreases exponentially, leading to a significantly smaller overhead. A similar pattern is observed for MT-Redundant and MT-Bench, as presented in Figure 7 in the appendix.

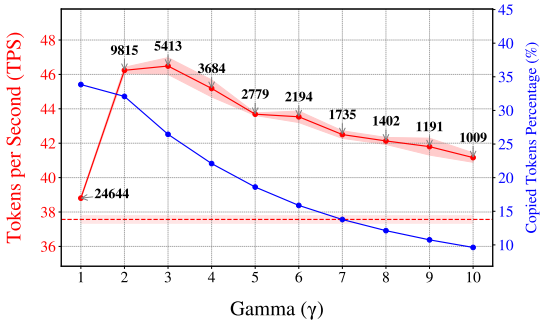

Figure 3: This figure shows how the copying parameter γ affects HumanEval performance using LLaMa3.1-8B-Instruct. The solid red line indicates tokens per second (TPS) with standard deviation shading; the dashed red line marks baseline TPS. The blue line shows the percentage of successfully copied tokens, with adjacent numbers indicating copying attempts.

Empirically, the optimal value of γ across datasets is three, with two yielding similar performance. It is also worth noting that all γ values ranging from 2 to 10, consistently results in significantly higher overall TPS, even across both turns on MT-Redundant and MT-Bench.

5.3 Number of Tokens to Copy and Overhead

We evaluate the impact of the number of tokens copied on performance and estimate CopySpec's overhead by setting the number of copied tokens to zero, isolating the cost of token searching. Results in Table 5 show minimal overhead with differences from the base model nearly within the margin of

Category	Turn 1				Turn 2			
	Base Model	**Spec. Dec.**	**Spec. Dec. + Copy ($\gamma = 3$)**	**Spec. Dec. + Copy ($\gamma = 5$)**	**Base Model**	**Spec. Dec.**	**Spec. Dec. + Copy ($\gamma = 3$)**	**Spec. Dec. + Copy ($\gamma = 5$)**
Coding	10.87 ±0.01	15.88 ±0.01	15.85 ±0.08	**16.17** ±0.01	9.73 ±0.01	14.74 ±0.01	22.12 ±0.03	**22.17** ±0.08
Extraction	10.09 ±0.01	14.07 ±0.02	**15.49** ±0.08	15.41 ±0.01	9.79 ±0.01	14.50 ±0.02	18.56 ±0.10	**18.69** ±0.01
Humanities	10.85 ±0.01	13.62 ±0.03	**13.86** ±0.02	13.88 ±0.01	9.75 ±0.01	12.79 ±0.02	**13.66** ±0.02	13.73 ±0.03
Math	11.01 ±0.01	16.94 ±0.05	17.23 ±0.01	**17.30** ±0.02	10.05 ±0.01	15.45 ±0.01	**24.28** ±0.03	24.11 ±0.04
Reasoning	10.80 ±0.02	13.96 ±0.02	14.18 ±0.20	**14.24** ±0.07	10.05 ±0.01	14.20 ±0.01	**21.56** ±0.09	20.35 ±0.07
Roleplay	10.90 ±0.01	12.80 ±0.04	12.84 ±0.01	**12.97** ±0.01	9.93 ±0.01	15.14 ±0.03	**29.02** ±0.01	27.95 ±0.09
Stem	10.90 ±0.01	14.25 ±0.03	14.33 ±0.01	**14.56** ±0.01	9.83 ±0.01	13.94 ±0.01	17.22 ±0.02	**17.26** ±0.02
Writing	10.92 ±0.01	12.56 ±0.05	12.64 ±0.01	**12.73** ±0.01	9.94 ±0.01	14.96 ±0.02	**26.64** ±0.04	25.08 ±0.08
Average	10.79 ±0.01	14.26 ±0.03	14.55 ±0.05	**14.66** ±0.02	9.88 ±0.01	14.47 ±0.02	**21.63** ±0.04	21.17 ±0.05

Table 4: Comparison of decoding strategies in MT-Redundant across two turns, using Qwen2.5-32B-Instruct as the target model and Qwen2.5-7B-Instruct as the draft model. The table demonstrates the impact of CopySpec integration at different parameter settings ($\gamma = 3$ and $\gamma = 5$), with the draft model generating 3 tokens. Results highlight significant improvements in speed and token copying efficiency, particularly in the second turn, due to the interplay between speculative copying and draft model generation.

error. Among the hyperparameters studied, setting $|S_{copyspec} = 10|$ delivers the best performance, while larger values, such as 50 or 100, increase overhead and reduce tokens-per-second efficiency.

Tokens Copied	MT-Redundant	MT-Bench
Base Model	35.63 ±0.04	35.30 ±0.16
0	35.46 ±0.01	35.22 ±0.04
5	47.64 ±0.11	44.69 ±0.11
10	**49.52** ±0.01	**45.74** ±0.01
50	45.56 ±0.08	41.59 ±0.04
100	39.41 ±0.06	35.76 ±0.05

Table 5: Tokens-per-second (TPS) on MT-Redundant and MT-Bench with LLaMa3.1-8B-Instruct. Results demonstrate that copying 10 tokens achieves optimal performance, while larger copying attempts introduce overhead, reducing overall efficiency.

Furthermore, we examine the effect of γ on τ (the average number of tokens accepted). Figure 4 illustrates the average number of tokens accepted per attempt on HumanEval using the LLaMA3.1-8B-Instruct model. We observe an interesting pattern: as γ increases, the average number of tokens accepted per copying attempt also increases, indicating that each attempt becomes more precise. However, this comes at the cost of fewer overall copying attempts, as demonstrated in Figure 3.

This finding is particularly relevant for integrating our technique into various speculative decoding frameworks. If a framework already accepts a high number of tokens per attempt, our technique remains advantageous by increasing γ, enabling more tokens to be copied with each attempt.

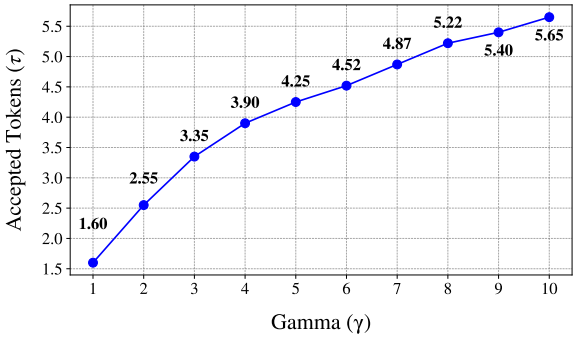

Figure 4: Average accepted tokens per copy attempt against γ using LLaMa-8B-Instruct ($|S_{\text{copyspec}}| = 10$), showcasing the correlation between γ and the accepted tokens.

6 Analyses

6.1 Orthogonality with Vanilla Speculative Decoding

We followed the steps outlined in section 3.3 to integrate our technique into a vanilla speculative decoding framework, as described in (Leviathan et al., 2022). Based on our observations from section 5.2, we experimented with two different values of γ (3 and 5) to analyze their impact on performance when used alongside speculative decoding.

We integrated CopySpec into a vanilla speculative decoding framework, following the steps in section 3.3 and the approach described in (Leviathan et al., 2022). Experiments with γ values of 3 and 5, summarized in table 4, show significant efficiency improvements in the second turn of MT-Redundant. A γ value of 5 achieves higher speedups in the first turn, while $\gamma = 3$ provides better TPS in the second turn, highlighting the need for task-specific tuning.

Variant	Turn 1				Turn 2				Turn 3			
	Copied	Tokens/Sec	τ_1	τ_2	Copied	Tokens/Sec	τ_1	τ_2	Copied	Tokens/Sec	τ_1	τ_2
Base Model	–	10.25±0.01	–	–	–	10.17±0.01	–	–	–	8.68±0.01	–	–
CopySpec ($\gamma = 3$)	5.76%	10.13±0.01	0.58	–	44.17%	15.72±0.01	4.90	–	82.79%	21.89±0.01	7.67	–
CopySpec ($\gamma = 5$)	1.01%	9.91±0.02	0.72	–	40.67%	14.79±0.01	6.96	–	82.78%	21.39±0.02	8.70	–
Spec. Dec.	–	13.47±0.02	–	2.55	–	12.99±0.03	–	2.31	–	11.27±0.01	–	2.75
Spec. Dec. + Copy ($\gamma = 3$)	2.59%	13.09±0.02	0.60	2.52	41.70%	16.37±0.04	5.85	1.86	81.81%	21.23±0.04	7.70	2.39
Spec. Dec. + Copy ($\gamma = 5$)	0.49%	**13.67**±0.03	0.90	2.55	39.26%	**16.59**±0.03	7.89	1.92	82.58%	**21.91**±0.02	8.71	2.35

Table 6: Performance comparison for self-correcting tasks with the draft model generating 3 tokens at a time. Qwen2.5-32B-Instruct is the target model, and Qwen2.5-7B-Instruct is the draft model. The Base Model averages 9.76 TPS, while Spec. Dec. + CopySpec ($\gamma = 5$) averages 16.75 TPS across all three rounds. τ_1 is the average tokens accepted by CopySpec, and τ_2 is the average tokens accepted by the draft model. Self-correction leads to an improvement in accuracy from 92% to 93%, for more details see table 24 in Appendix.

We also evaluated CopySpec with speculative decoding using drafts of 5 tokens instead of 3, with similar experiments conducted on MT-Redundant (table 15, in Appendix) and with 3 and 5 draft tokens on MT-Bench (table 21 and table 22, in Appendix). These results confirm that $\gamma = 5$ often outperforms $\gamma = 3$ when combined with Spec. Dec., emphasizing the importance of tuning γ for optimal performance. The results also show that adding CopySpec to Spec. Dec. almost never leads to a decrease in performance.

When paired with EAGLE, our verifier lifts Vicuna-v1.3-13B throughput on GSM8K from 62 TPS (plain EAGLE) to 142 TPS (turn 3)—*3.5*× the greedy baseline (Appendix C).

6.2 Complementarity with External Drafting Frameworks

Coupling a strong drafter with CopySpec delivers the fastest runs across the board—up to **2.4**× throughput on Vicuna-v1.3-7B and **3.1**× on Vicuna-v1.3-13B relative to the greedy decoder (Table 7). Enlarging the speculation window from 10 to 50 tokens still yields an additional 15–40 % gain on overlap-heavy tasks.

Method	Tokens/Sec	Speed-up
Base model	39.30 ± 0.18	1.00×
PLD (window= 5)	51.08 ± 0.11	1.30×
CopySpec (γ=5)	56.13 ± 0.01	1.43×
EAGLE	82.53 ± 0.18	2.10×
EAGLE + SAM-D	84.75 ± 0.05	2.16×
EAGLE + Copy (γ=5)*	$\mathbf{94.84 \pm 0.16}$	**2.41**×

Table 7: Throughput on **MT-Redundant** (*0-shot*) for Vicuna-v1.3-7B. Speed-ups are versus the base model. Complete per-task results appear in Tables 10–8. * indicates that the speculation window was increased to 50 tokens instead of the default 10.

Our experiments reveal three take-aways. First, coupling a strong drafter with span-level copying delivers the fastest runs across the board—up to **2.9**× throughput on Vicuna-v1.3-7B and **3.6**× on Vicuna-v1.3-13B relative to the greedy decoder(Appendix Table 10). Second, enlarging the speculation window from 10 to 50 tokens adds a further 15–40 % throughput when prompts contain large verbatim spans (*e.g.*, GSM8K turn 3 reaches 142 TPS on Vicuna-v1.3-13B). Third, competing methods such as PLD and SAM-D trail by a wide margin, underscoring that our lighter verifier integrates more cleanly with external drafting.

6.3 Effect on Reasoning

An important aspect of our analysis is evaluating the impact of our technique on the efficiency of self-correction. To this end, we implemented a self-refine framework, where the model generates Python code and iteratively refines it in two steps, following a process similar to (Madaan et al., 2023). Details of the prompts and example outputs used in our experiments are provided in Appendix H.1. Table 6 presents the results of self-correction with copying and Speculative Decoding (SD). Additionally, Tables 25 and 26 extend our evaluation to the more challenging AIME'24 (aim, 2024) and AIME'25 (aim, 2025) benchmarks.

We also extended our analysis to cases where the draft model generates 5 tokens at a time, as shown in Table 23 in the appendix. Additionally, Table 24 confirms that the tested models improve their final accuracy, validating the effectiveness of our self-correction implementation. Note that accuracy is not reported for the second round, as it focuses solely on critiquing the model's prior implementation. Across the entire self-correction process, we achieve TPS improvements of 63%, 52%, and 54% for the Qwen2.5-7B, Qwen2.5-32B, and Qwen2.5-72B instruct models, respectively.

Our technique becomes more effective in later turns as the model iterates over its prior reasoning. This is reflected in a significant rise in the percentage of copied tokens, tokens per second (TPS), and τ_1, the average number of tokens accepted. Each copying attempt also becomes more precise as the model refines its reasoning and the context grows.

When combined with SD using $\gamma = 5$, our approach achieves better results across all three turns, as shown in the table. The first turn benefits most from SD due to minimal copying, while later turns gain greater advantages from copying. This highlights the complementary nature of the two techniques and their combined effectiveness in improving efficiency and performance. Notably, while the TPS of the base model decreases by 0.85× as context size grows, our technique reverses this trend, increasing the TPS in the last turn by 2.52×, showcasing its ability to leverage larger contexts.

We also extended our analysis to cases where the draft model generates 5 tokens at a time, as shown in table 23 in the appendix. Additionally, table 24 confirms that the tested models improve their final accuracy, validating the effectiveness of our self-correction implementation. Note that accuracy is not reported for the second round, as it focuses solely on critiquing the model's prior implementation. Across the entire self-correction process, we achieve TPS improvements of 63%, 52%, and 54% for the Qwen2.5-7B, Qwen2.5-32B, and Qwen2.5-72B instruct models, respectively.

7 Conclusion

We introduced *CopySpec*, a method that identifies repeated token sequences in a growing context and copies them efficiently without additional GPU memory or significant cost. Using a rolling hash for γ tokens, *CopySpec* speculates on larger token blocks to reduce redundant computation.

Results across five LLMs and datasets, including MT-Redundant, show up to a 3.08× speed-up in second-turn inference and a 49% boost when combined with speculative decoding, without altering output quality. Future work includes dynamically tuning γ, refining match selection, and integrating *CopySpec* with parallel decoding frameworks.

Limitations

CopySpec provides a lightweight and effective approach for accelerating LLM inference by leveraging repeated patterns in the context. It integrates seamlessly with speculative decoding and shows consistent gains across models and tasks, particularly in multi-turn settings with incremental refinements. While MT-Redundant is designed to cover a wide range of realistic scenarios—including code revision, reasoning, summarization, and stylistic rewriting—it still assumes that the repeated content appears in close proximity and with high lexical overlap. Future work will examine how CopySpec performs under looser or cross-sentence redundancy patterns, such as those found in long-form document editing or open-ended dialogue.

The current approach uses fixed hyperparameters for the copying window γ and speculative block size $|S_{\text{copyspec}}|$, which may not be optimal for all settings. Although our analysis in Appendix A shows that γ can be reliably selected using a small number of samples based on cosine similarity trends, more adaptive strategies that adjust γ and block length dynamically based on local context or model uncertainty could further improve generality and efficiency.

Finally, when multiple candidate matches exist for the last γ tokens, the current system selects the first match encountered without regard to semantic alignment. To improve copy quality and avoid suboptimal completions, future work will explore context-aware ranking mechanisms that select the most useful copy targets based on semantic or structural compatibility with the ongoing generation.

Ethical Considerations

Our proposed method, *CopySpec*, is intended as a complementary addition to existing speculative decoding techniques and does not introduce any novel text generation of its own. Instead, it operates by selecting and copying text from context, with the final output verified by the main LLM used in the respective pipeline. Consequently, any potential biases or hallucinations in the output are attributable to the choice of the main LLM, which remains an independent and external component. While CopySpec inherently reduces hallucination risk by copying existing context, it may still propagate hallucinated content if such content exists in the context itself. The responsibility for verifying such content again lies with the main LLM employed in the speculative decoding process.

References

2024. American invitational mathematics examination (aime) 2024 — aime i & aime ii. https://artofproblemsolving.com/wiki/index.php/2024_AIME_I; https://artofproblemsolving.com/wiki/index.php/2024_AIME_II. AIME I held Jan 31–Feb 1, 2024; AIME II held Feb 7, 2024.

2025. American invitational mathematics examination (aime) 2025 — aime i & aime ii. https://artofproblemsolving.com/wiki/index.php/2025_AIME_I; https://artofproblemsolving.com/wiki/index.php/2025_AIME_II. AIME I held Feb 6, 2025; AIME II held Feb 12, 2025.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. 2024. Medusa: Simple LLM inference acceleration framework with multiple decoding heads. In *Forty-first International Conference on Machine Learning*.

Hao Mark Chen, Wayne Luk, Ka Fai Cedric Yiu, Rui Li, Konstantin Mishchenko, Stylianos I. Venieris, and Hongxiang Fan. 2024. Hardware-aware parallel prompt decoding for memory-efficient acceleration of llm inference. *Preprint*, arXiv:2405.18628.

Jia Chen and Hao Xu. 2023. Parallel decoding with speculative sampling for large language models. *arXiv preprint arXiv:2306.15478*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

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

Dvir Cohen, Lin Burg, Sviatoslav Pykhnivskyi, Hagit Gur, Stanislav Kovynov, Olga Atzmon, and Gilad Barkan. 2025. Wixqa: A multi-dataset benchmark for enterprise retrieval-augmented generation. *Preprint*, arXiv:2505.08643.

Kuicai Dong, Yujing Chang, Xin Deik Goh, Dexun Li, Ruiming Tang, and Yong Liu. 2025a. Mmdocir: Benchmarking multi-modal retrieval for long documents. *arXiv preprint arXiv:2501.08828*.

Kuicai Dong, Yujing Chang, Shijie Huang, Yasheng Wang, Ruiming Tang, and Yong Liu. 2025b. Benchmarking retrieval-augmented multimomal generation for document question answering. *arXiv preprint arXiv:2505.16470*.

Kuicai Dong, Derrick Goh Xin Deik, Yi Quan Lee, Hao Zhang, Xiangyang Li, Cong Zhang, and Yong Liu. 2024. MC-indexing: Effective long document retrieval via multi-view content-aware indexing. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 2673–2691, Miami, Florida, USA. Association for Computational Linguistics.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. 2024. Break the sequential dependency of LLM inference using lookahead decoding. In *Forty-first International Conference on Machine Learning*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Jiatao Gu, Zhaopeng Lu, Hang Li, and Victor OK Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1631–1640.

Zhang He and Xin Wang. 2023. Speed: Speculative pipelined execution for efficient decoding in large language models. *arXiv preprint arXiv:2310.12072*.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason Lee, and Di He. 2024. REST: Retrieval-based speculative decoding. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1582–1595, Mexico City, Mexico. Association for Computational Linguistics.

Yuxuan Hu, Ke Wang, Xiaokang Zhang, Fanjin Zhang, Cuiping Li, Hong Chen, and Jing Zhang. 2024. Sam decoding: Speculative decoding via suffix automaton. *Preprint*, arXiv:2411.10666.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2022. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2025. Eagle: Speculative sampling requires rethinking feature uncertainty. *Preprint*, arXiv:2401.15077.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is your code generated by chatGPT really correct? rigorous evaluation of large language models for code generation. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Xukun Liu, Yifan Zhang, Peiyi Wang, Tao Ge, Tianyu Liu, Yongqi Li, and Zhifang Sui. 2024. Adaptive draft-verification for efficient large language model decoding. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1234–1245.

Xianzhen Luo, Yixuan Wang, Qingfu Zhu, Zhiming Zhang, Xuanyu Zhang, Qing Yang, and Dongliang Xu. 2025. Turning trash into treasure: Accelerating inference of large language models with token recycling. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6816–6831, Vienna, Austria. Association for Computational Linguistics.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. *Preprint*, arXiv:2303.17651.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. 2024. Specinfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ASPLOS '24, page 932–949. ACM.

Apoorv Saxena. 2023. Prompt lookup decoding.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. *CoRR*, abs/1704.04368.

Shwetha Somasundaram, Anirudh Phukan, and Apoorv Saxena. 2025. PLD+: Accelerating LLM inference by leveraging language model artifacts. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 6075–6089, Albuquerque, New Mexico. Association for Computational Linguistics.

Ryan Sun, Tianyi Zhou, Xun Chen, and Lichao Sun. 2024. Specthub: Provable acceleration to multi-draft speculative decoding. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Advances in Neural Information Processing Systems*, volume 28, pages 2692–2700.

Zikai Xiao, Ziyang Wang, Wen Ma, Yan Zhang, Wei Shen, WangYan WangYan, Luqi Gong, and Zuozhu Liu. 2025. Mitigating posterior salience attenuation in long-context LLMs with positional contrastive decoding. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 724–733, Vienna, Austria. Association for Computational Linguistics.

Qwen An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxin Yang, Jingren Zhou, Junyang Lin, and 25 others. 2024a. Qwen2.5 technical report. *ArXiv*, abs/2412.15115.

Seongjun Yang, Gibbeum Lee, Jaewoong Cho, Dimitris Papailiopoulos, and Kangwook Lee. 2024b. Predictive pipelined decoding: A compute-latency trade-off for exact LLM decoding. *Transactions on Machine Learning Research*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Preprint*, arXiv:2306.05685.

Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. 2024. Distillspec: Improving speculative decoding via knowledge distillation. In *The Twelfth International Conference on Learning Representations*.

A Gamma (γ) and Semantic Implications

In our framework, the generation speed of CopySpec is intricately tied to the choice of γ, which governs the length of the left context used to identify repeated sequences. The selection of an optimal γ is critical, as it directly impacts the model's ability to efficiently reuse tokens from the context, thereby accelerating generation. A carefully chosen γ strikes a balance between providing sufficient contextual information for accurate copying and avoiding unnecessary computational overhead.

If γ is too small (e.g., $\gamma = 1$), the context provides insufficient information to reliably identify repetitions, resulting in missed reuse opportunities and slower generation. Conversely, when γ is too large, the excessive context introduces redundancy and dilutes the immediate semantic relevance. While the acceptance rate may increase, the total number of tokens generated per second decreases because the model spends more time processing generate tokens itself and fewer tokens are copied in practice.

The challenge, therefore, lies in finding an optimal γ that maximizes copying attempts while minimizing computational overhead. A well-chosen

γ ensures that the context is both semantically focused and computationally efficient, enabling the Copy mechanism to fully exploit repeated patterns in the generation process. This tradeoff underscores the importance of systematically tuning γ to achieve the best performance across datasets.

To measure the semantic alignment between a token w and its left-γ token context, we fine-tuned the token embeddings using a **left-γ skip-gram** model, a modification of the traditional skip-gram approach. Unlike the standard skip-gram model, which maximizes the probability of a target word given a symmetric context window, our approach considers only the preceding γ tokens as context.

Formally, instead of maximizing the probability $\prod_{(w,C)\in D} P(w|C)$, where C represents a symmetric context window around the word w, our **left-γ skip-gram** model is trained to maximize $\prod_{(t,C_{\text{left}\,\gamma})\in D} P(t|C_{\text{left}\,\gamma})$, where $C_{\text{left}\,\gamma}$ consists only of the last γ tokens in the sequence to predict the next token t. This ensures that the learned embeddings capture dependencies in a unidirectional manner, aligning with the way generative models process text.

By structuring the model in this way, we aim to quantify how much semantic meaning from the left-γ tokens contributes to predicting the next token. Cosine Similarity is particularly well-suited for evaluating the semantic alignment between the left-γ token context and the next token because it captures the directional similarity between their vector representations, regardless of magnitude. Since word embeddings encode semantic meaning in a high-dimensional space, CS provides a robust way to measure how well the left context conveys predictive information about the next token. Unlike Euclidean Distance, CS ensures that we focus solely on semantic coherence rather than raw frequency effects. This is crucial for CopySpec, as effective token reuse depends on the ability to recognize when a sequence of past tokens is not just lexically repeated but also semantically relevant to the next token. By analyzing trends in CS across different γ-values, we can assess whether increasing the context length improves meaningful copying or merely introduces redundant information, thereby helping us fine-tune γ for optimal efficiency.

The cosine similarity (CS) is computed as:

$$\text{CS}(\vec{v}_{C_{\text{left}\,\gamma}}, \vec{v}_t) = \frac{\vec{v}_{C_{\text{left}\,\gamma}} \cdot \vec{v}_t}{\|\vec{v}_{C_{\text{left}\,\gamma}}\| \|\vec{v}_t\|}.$$

Here, $\vec{v}_{C_{\text{left}\,\gamma}} = \frac{1}{\gamma} \sum_{i=1}^{\gamma} \vec{v}_{t_i}$ represents the average embedding of the most recent γ tokens, where $\{t_i\}_{i=1}^{\gamma}$ are the embeddings of the last γ tokens in the context.

To validate our intuitions, we conducted experiments to analyze the relationship between γ (context length) and semantic alignment. Figure 5 illustrates the trends in Cosine Similarity and generation speed (TPS) as γ varies.

By measuring Cosine Similarity and generation speed across varying γ-token contexts, we provide empirical evidence that fine-tuning **left-γ skip-gram** model for the best γ is essential for maximizing efficiency. Future work can explore adaptive strategies that dynamically adjust γ in the same hashmap based on context complexity, further optimizing the balance between copying effectiveness and computational cost.

B Copying and Speculative Decoding with Truncated KV States

This appendix describes how our framework integrates a *copying mechanism* with *speculative decoding*, including details on partial acceptance, key-value (KV) cache truncation.

B.1 Notation and Variables

Sequence $X_{1:t}$. Let $X_{1:t}$ be the currently accepted sequence of t tokens. Generating a new token moves us to position $t+1$.

Dictionary $\mathcal{D}$. $\mathcal{D}$ records repeated γ-length substrings and their earlier occurrences. If $X_{t-\gamma+1:t}$ appears in $\mathcal{D}$, we may copy subsequent tokens from that match.

Subsequence length γ. We use γ tokens to detect repeats. That is, the last γ tokens, $s = X_{t-\gamma+1:t}$, determine if a copy event is possible.

Match location p. If $\mathcal{D}$ indicates $X_{t-\gamma+1:t}$ appears at position p, we attempt to copy tokens starting from $p+\gamma$.

Chunk size m (copying). When a match is found, we form a copied chunk

$$\widetilde{X}_{1:m} \;=\; (\widetilde{x}_1, \ldots, \widetilde{x}_m) \;=\; X_{p+\gamma:\,p+\gamma+m-1}.$$

Draft limit δ (speculative). If copying is not used, we let the draft model propose up to δ tokens:

$$\widehat{X}_{1:\delta} \;=\; (\widehat{x}_1, \ldots, \widehat{x}_\delta).$$

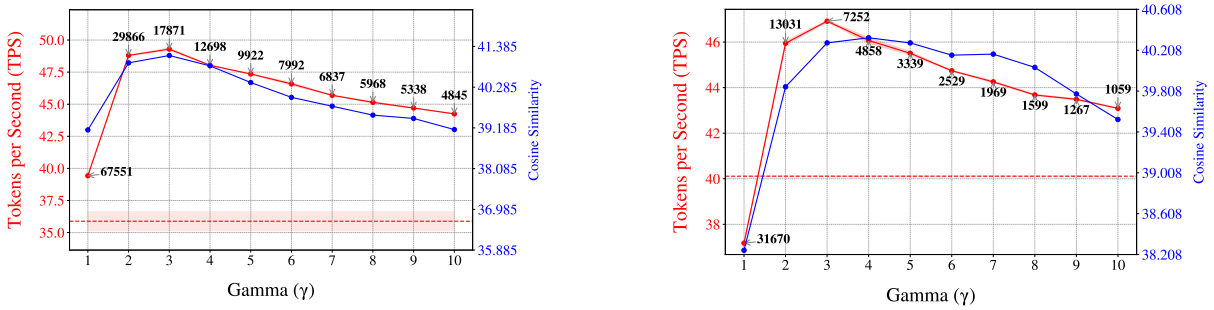

Figure 5: We use Qwen2.5-7B on both MT-Bench and MT-Redundant dataset. Cosine Similarity and Tokens per Second trends as a function of γ. The blue line indicates the Cosine Similarity, showing semantic alignment across varying γ-token contexts. The red line illustrates the Tokens per Second, reflecting generation speed. γ denotes the number of tokens considered in the context for each measurement. The left plot shows MT-Bench, and the right plot shows MT-Redundant.

Acceptance and Draft Models. The target model $p_{\text{target}}(\cdot \mid X_{1:n})$ decides whether each new token is accepted, while the draft model $p_{\text{draft}}(X_t \mid X_{1:n})$ only proposes tokens that must still pass p_{target}'s acceptance criterion.

Index i. In both copying and drafting, we iterate over newly proposed tokens with an index $i \in \{1, \ldots, m\}$ or $i \in \{1, \ldots, \delta\}$.

Accepted count k. Out of the m (copied) or δ (drafted) tokens, only $k \leq m$ or $k \leq \delta$ may be accepted under p_{target}. Rejected tokens are removed, and the key-value states are truncated to retain only $X_{1:t+k}$.

B.2 Acceptance Criterion and KV Truncation

Any new token x_{t+i} must pass an acceptance criterion under p_{target}; for example, at *temperature* 0, we only accept it if it is the `argmax` of the target model's conditional distribution. If the token fails, we reject it (and all subsequent tokens in the same chunk) and *roll back* to $X_{1:t+i-1}$.

Each layer ℓ of the target model stores key-value tensors $(\mathbf{K}_\ell, \mathbf{V}_\ell)$ up to the final accepted token. If $k < i - 1$ tokens in a chunk are accepted, we truncate $(\mathbf{K}_\ell, \mathbf{V}_\ell)$ to $t + k$ positions, ensuring the model remains consistent with the final accepted sequence.

B.3 Integrated Generation Procedure

Below is a single pseudocode listing that combines both *copying* and *speculative decoding*.

1. **Check for a Copy Opportunity:**
   (a) Let $s = X_{t-\gamma+1:t}$ be the most recent γ tokens of the accepted sequence $X_{1:t}$.
   (b) Check if s is in $\mathcal{D}$ (the dictionary of repeats).
   - If no match exists, go to Step 3.

   (c) Otherwise, let p be the first occurrence in $\mathcal{D}(s)$ satisfying $p + \gamma - 1 < t - \gamma + 1$ (ensuring no overlap).
   (d) Form a candidate chunk of length m:

$$\widetilde{X}_{1:m} = X_{p+\gamma:\, p+\gamma+m-1}.$$

   (e) Initialize $k = 0$, which tracks how many tokens from $\widetilde{X}_{1:m}$ are ultimately accepted.

2. **Attempt to Copy:**
   (a) For $i = 1$ to m:
   - Evaluate $\widetilde{x}_i$ (from $\widetilde{X}_{1:m}$) with the target model:

$$p_{\text{target}}\big(X_t \mid X_{1:t+i-1}\big).$$

   - If $\widetilde{x}_i$ passes the acceptance criterion (e.g. it is the `argmax` if temperature = 0), set $k \leftarrow k + 1$; otherwise, *reject* $\widetilde{x}_i$ and break out of this loop.

   (b) If $k < m$:
   - The final sequence is now $X_{1:t+k}$, which means only the first k tokens from $\widetilde{X}_{1:m}$ (i.e. $\widetilde{x}_1, \ldots, \widetilde{x}_k$) are accepted.
   - Truncate the target model's KV Cache states for all layers to length $t + k$ to discard any rejected tokens beyond position $t + k$.

   (c) Otherwise, if $k = m$, then all m copied tokens are fully accepted, making $X_{1:t+m}$ the new final sequence.

(d) **Update** $\mathcal{D}$ with any newly formed γ-subsequences ending at positions $t+j$ for $1 \le j \le k$.

3. **Speculative Decoding:**

(a) If no copying occurred, generate δ tokens from the draft model:

$$\widehat{X}_{1:\delta} \sim p_{\text{draft}}\big(X_t \mid X_{1:t}\big).$$

(b) Let $k = 0$. For $i = 1$ to δ:

- Evaluate $\widehat{x}_i$ (from $\widehat{X}_{1:\delta}$) using

$$p_{\text{target}}\big(X_t \mid X_{1:t+i-1}\big).$$

- If accepted, increment k. If rejected, break immediately.

(c) If $k < \delta$:

- Only $\widehat{x}_1, \ldots, \widehat{x}_k$ are accepted, so the final sequence is $X_{1:t+k}$.
- Truncate the target model's and draft model's KV Cache states to reflect $X_{1:t+k}$ only.

(d) If $k = \delta$, the entire draft $\widehat{X}_{1:\delta}$ is accepted, making $X_{1:t+\delta}$ the new final sequence.

(e) **Update** $\mathcal{D}$ with any newly formed γ-length subsequences up to position $t+k$.

4. **Repeat**: Increase t by the number of accepted tokens (either k, m, or δ) in this iteration. Continue until a stopping criterion (e.g. end-of-text token) is encountered.

Discussion of Truncation: Whenever fewer than m (in copying) or δ (in drafting) tokens are accepted, we *roll back* to the accepted prefix. The target model's key-value memory is truncated accordingly to reflect $X_{1:t+k}$. Thus, any rejected tokens do not affect the final context or the KV states.

C Baseline Comparisons and EAGLE

The acceleration techniques discussed in the related works include PLD and PLD-copy (Saxena, 2023; Somasundaram et al., 2025), PPD (Chen et al., 2024), look-ahead decoding (Fu et al., 2024), token recycling (Luo et al., 2025), and SAM-D (Hu et al., 2024). Of these, only PLD, PPD, and SAM have official publicly available code. Using the reference implementation from Somasundaram *et al.*, we were able to reproduce the throughput improvements reported for PLD, observing comparable end-to-end latency reductions relative to our baseline. In contrast, applying the PPD codebase as described by Chen *et al.* consistently yielded token-generation speeds below the unmodified decoder—i.e., worse-than-baseline performance in our environment. For SAM, we used the authors' implementation with the number of predicted tokens ($n_{predicts}$) set to 15 and the default hyperparameters for EAGLE integration presented in (Hu et al., 2024). The other methods—e.g. token recycling—do not provide official open-source implementations, so we were unable to attempt replication of their reported results but our technique is orthogonal to theirs.

For the EAGLE experiments, we omitted key–value caching in the EAGLE heads due to implementation challenges within our copy framework. Instead, we set EAGLE to generate batches of 20 tokens at a time; after each batch, we check whether the copy mechanism can be applied. Whenever copying is possible, we favor it over EAGLE-generated tokens and continue copying until no further matches are found; if copying is

Variant	Turn 1	Turn 2	Turn 3
Base Model	40.16±0.14	37.69±0.14	35.42±0.07
EAGLE	76.48±0.68	70.87±0.91	88.56±0.71
EAGLE + CopySpec($\gamma = 3$)	**80.85**±0.68	75.04±0.56	101.60±0.40
EAGLE + CopySpec($\gamma = 5$)	78.05±0.05	74.27±0.03	101.14±0.22
EAGLE + CopySpec($\gamma = 5$)*	77.94±0.98	73.17±0.80	**141.60**±0.96
EAGLE + SAM-D	60.27±0.15	**98.18**±0.07	118.56±0.08
CopySpec($\gamma = 3$)	49.89±0.24	41.70±0.28	96.85±0.44
CopySpec($\gamma = 5$)	47.66±0.04	40.38±0.02	99.83±1.67
CopySpec($\gamma = 5$)*	44.89±0.21	39.27±0.13	126.41±0.24
PLD(window=3)	45.74±0.10	56.48±0.06	59.05±0.02
PLD(window=5)	45.10±0.01	61.32±0.02	67.30±0.18

Table 8: We show the tokens/sec by turn on GSM8K using the Vicuna-v1.3-7B model. γ denotes CopySpec's prompt lookup size; * indicates that the speculation window was increased to 50 tokens instead of the default 10.

Variant	Vicuna-7B-v1.3	Vicuna-13B-v1.3
Base Model	28.50 ± 0.10	17.47 ± 0.01
CopySpec ($\gamma = 3$)	38.46 ± 0.12	22.70 ± 0.06
CopySpec ($\gamma = 5$)	35.81 ± 0.01	21.29 ± 0.03
CopySpec ($\gamma = 5$)*	34.34 ± 0.02	20.22 ± 0.04
EAGLE	63.43 ± 0.07	47.17 ± 0.24
EAGLE + CopySpec ($\gamma = 3$)	66.92 ± 0.04	49.78 ± 0.12
EAGLE + CopySpec ($\gamma = 5$)	66.96 ± 0.15	48.95 ± 0.11
EAGLE + CopySpec ($\gamma = 5$)*	**68.99** ± 0.68	**49.90** ± 0.08
EAGLE + SAM-D	60.70 ± 0.08	37.43 ± 0.04

Table 9: **WixQA (RAG):** Throughput in tokens/s for Vicuna-7B and Vicuna-13B with COPYSPEC alone and combined with EAGLE speculative decoding. Greedy decoding (T=0); outputs remain token-identical to baseline. * indicates that the speculation window was increased to 50 tokens instead of the default 10

Model	Variant	MT-Redundant *0-shot*	CNN/DM *0-shot*	GSM8K *3-turn*	MT-Bench *0-shot*	HumanEval *0-shot*
Vicuna-v1.3-7B	EAGLE	82.53±0.18	56.87±0.14	80.45±0.75	81.78±0.40	87.76±0.11
	EAGLE + CopySpec($\gamma = 3$)	87.00±0.33	60.62±0.34	88.27±0.56	82.92±0.57	101.96±0.82
	EAGLE + CopySpec($\gamma = 5$)	84.80±0.39	61.96±0.41	87.37±0.07	83.79±1.37	104.77±0.53
	EAGLE + CopySpec($\gamma = 5$)*	**94.84**±0.16	**65.63**±0.18	**108.07**±1.02	**87.39**±1.59	**108.39**±0.96
	EAGLE + SAM-D	84.75±0.05	50.04±0.02	85.18±0.13	73.51±0.66	90.00±0.10
	CopySpec($\gamma = 3$)	57.19±0.25	47.59±0.01	61.58±0.12	52.31±0.14	60.69±0.15
	CopySpec($\gamma = 5$)	56.13±0.01	42.95±0.14	60.30±0.30	50.40±0.03	57.00±0.05
	CopySpec($\gamma = 5$)*	55.79±0.04	41.18±0.08	61.71±0.23	49.02±0.07	54.64±0.03
	PLD(window=3)	50.47±0.01	42.73±0.06	53.49±0.06	45.10±0.10	50.77±0.05
	PLD(window=5)	51.08±0.11	44.45±0.04	56.93±0.05	45.05±0.12	51.50±0.09
	Base model	39.30±0.18	31.36±0.01	37.48±0.11	39.48±0.09	40.40±0.09
Vicuna-v1.3-13B	EAGLE	62.31±0.18	43.23±0.25	68.35±0.29	62.83±0.42	69.34±0.45
	EAGLE + CopySpec($\gamma = 3$)	65.44±0.63	45.81±0.30	70.39±0.25	64.33±0.56	69.97±0.22
	EAGLE + CopySpec($\gamma = 5$)	66.11±0.74	45.08±0.52	71.88±0.48	63.58±0.99	73.18±0.97
	EAGLE + CopySpec($\gamma = 5$)*	**73.20**±0.34	**46.94**±0.21	**82.27**±0.75	**65.52**±0.78	**74.16**±0.68
	EAGLE + SAM-D	50.85±0.13	27.63±0.02	47.31±0.01	45.10±0.01	50.17±0.03
	CopySpec($\gamma = 3$)	34.12±0.05	25.18±0.06	34.50±0.08	29.27±0.02	31.30±0.01
	CopySpec($\gamma = 5$)	32.96±0.06	22.98±0.01	33.49±0.05	28.19±0.06	30.67±0.02
	CopySpec($\gamma = 5$)*	32.24±0.03	21.42±0.01	33.31±0.02	26.96±0.05	29.16±0.05
	PLD(window=3)	31.57±0.09	25.71±0.02	31.99±0.06	28.61±0.01	31.51±0.05
	PLD(window=5)	31.10±0.02	25.56±0.02	31.77±0.02	27.57±0.04	30.10±0.05
	Base model	23.30±0.01	18.42±0.04	22.61±0.01	23.48±0.07	24.25±0.05

Table 10: Performance comparison of Vicuna-1.3(7B and 13B) under baseline, EAGLE, CopySpec, and PLD configurations measured by tokens/sec. Reported values are tokens per second (mean±std). γ denotes CopySpec's prompt lookup size; * indicates that the speculation window was increased to 50 tokens instead of the default 10.

not possible, we invoke EAGLE again to generate the next 20-token batch, repeating this generate-then-copy cycle until the desired output length is reached.

For prompt-led decoding (PLD), we fixed the prompt lookup window to three or five tokens, matching the configuration used for CopySpec. All experiments were conducted with Vicuna-1.3 in its 7B and 13B variants, since the models presented in our paper lack pretrained EAGLE weights and we experienced difficulties using the EAGLE weights with LLaMa 3.1, as the model was broken and failed to generate the correct text.

Tables 8 and 10 summarize our main results. We also include WixQA(Cohen et al., 2025) in Table 9 to stress-test CopySpec in a retrieval-augmented setting; recent benchmarks on long-document indexing and multimodal DocRAG provide broader context for this evaluation (Dong et al., 2024, 2025a,b). We highlight the following observations:

1. **EAGLE + CopySpec synergy**: Coupling a strong drafter (EAGLE) with span-level copying consistently yields the fastest runs—up to *2.9*× speed-up on 7 B and *3.6*× on 13 B relative to the plain decoder.
2. **Speculation window matters**: Expanding the draft from 10 to 50 tokens ("*") adds a further 15–40 % throughput, especially on tasks with high intra-prompt overlap (GSM8K: +188% on 7 B; +264% on 13 B).
3. **CopySpec alone is modest**: Without a drafter, CopySpec delivers only 40–60 % of the EAGLE-backed gains, showing that copying and multi-token drafting are complementary.
4. **PLD and SAM-D lag**: Prompt lookup (PLD) and SAM-D trail by a wide margin—often not surpassing plain EAGLE—hinting that our technique adds a smaller overhead and thus provides better orthogonality with other speculative decoding frameworks.
5. **Task-level variance**: Benchmarks that reward verbatim reuse (MT-Redundant, GSM8K) benefit most, while abstractive CNN/DM summaries see the smallest absolute gains, highlighting content-overlap as a driver of copying efficiency.
6. **Late-turn surge**: With a 50-token draft, EAGLE + CopySpec(γ = 5) jumps to 142 TPS on turn 3—3.5× the greedy baseline—showing that long drafts pay off once the history is rich in overlaps.
7. **SAM-D's crossover**: SAM-D starts slower than EAGLE on turn 1 but overtakes it by turn 2, consistent with the O(1) suffix-extension cost claimed in the SAM paper.

D Extra Results on MT-Redundant

This appendix presents a detailed analysis of the performance improvements achieved by the CopySpec approach compared to baseline methods. The tables provide comprehensive results across various categories and model configurations, highlighting the computational efficiency and speed-ups observed on the MT-Redundant dataset.

D.1 Analysis of Gamma (γ) on MT-Redundant

The analysis depicted in Figure 6 highlights the impact of the copying parameter γ on both computational performance and the model's ability to reuse tokens effectively. As γ increases, there is a notable rise in the percentage of copied tokens, demonstrating the model's improved ability to exploit repeated patterns within the context. However, this comes at the cost of reduced tokens per second (TPS) for higher γ values, due to the increased computational overhead associated with processing larger context windows.

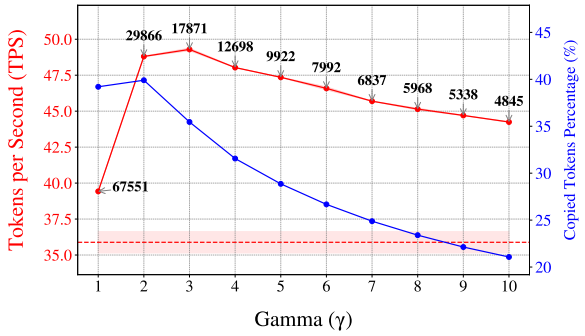

Figure 6: This figure illustrates the relationship between the copying parameter γ and the model's performance on the MT-Redundant dataset with the LLaMa3.1-8B-Instruct model. The notations are the same as in fig. 3.

D.2 Speed-up by Category on MT-Redundant

Table 11 summarizes the tokens-per-second (TPS) performance for the Qwen-32B-Instruct model across two turns. The first turn reflects scenarios with minimal contextual information, while the second turn demonstrates significant gains in speed due to the larger context size and CopySpec's ability to leverage repeated token patterns effectively. Notably, categories such as Coding and Math exhibit speed-ups exceeding 2× in the second turn.

In Table 12, we observe a similar trend for the Qwen-7B-Instruct model, with CopySpec consistently improving TPS across both turns. The second turn results show substantial gains in categories like Reasoning and Math, where repetitive patterns in the context are more prominent.

Table 13 presents the results for the LLaMa3.1-70B-Instruct model. Here, the impact of CopySpec is evident, especially in the second turn, with speed-ups reaching over 2× in categories such as Math. These results highlight the scalability of CopySpec across models of varying sizes.

The findings for the LLaMa3.1-8B-Instruct model are detailed in Table 14. The speed-ups in this case are slightly lower compared to larger models but still demonstrate consistent improvements across all categories, with notable efficiency gains in the second turn.

D.3 Merging with Speculative Decoding on MT-Redundant

Finally, Table 15 explores the integration of CopySpec with speculative decoding for the Qwen2.5-32B-Instruct model and Qwen2.5-7B-Instruct as the draft model. The results highlight how combining these approaches can yield even greater computational efficiency. The analysis includes varying γ values and draft token counts, showing that optimal parameter tuning further enhances performance, particularly in multi-turn scenarios.

E Extra Results on MT-Bench

This appendix presents a comprehensive evaluation of the CopySpec approach on the MT-Bench dataset across various configurations and categories. The results highlight the consistent improvements in tokens-per-second (TPS) performance achieved by CopySpec compared to baseline models, demonstrating its efficiency and scalability.

E.1 Analysis of Gamma (γ) on MT-Bench

Figure 7 presents a comprehensive visualization of how the copying parameter γ affects the performance of the LLaMa3.1-8B-Instruct model on the MT-Redundant dataset. The figure captures the interplay between the percentage of tokens successfully copied, the number of copying attempts, and the resulting tokens per second (TPS).

E.2 Speed-up by Category on MT-Bench

Table 16 provides the TPS performance of Qwen2.5-72B-Instruct on two turns. The speed-ups are most notable in categories such as Extrac-

	Turn 1			Turn 2		
Category	**Base Model**	**CopySpec**	**Speed-up**	**Base Model**	**CopySpec**	**Speed-up**
Coding	10.86 ±0.01	**11.66** ±0.02	1.07	9.72 ±0.01	**19.47** ±0.01	2.01
Extraction	10.09 ±0.01	**13.44** ±0.01	1.33	9.80 ±0.01	**18.17** ±0.01	1.85
Humanities	10.85 ±0.01	**11.57** ±0.01	1.07	9.75 ±0.01	**11.67** ±0.01	1.20
Math	11.01 ±0.01	**12.81** ±0.01	1.16	10.05 ±0.01	**23.18** ±0.01	2.31
Reasoning	10.80 ±0.02	**12.18** ±0.01	1.13	10.05 ±0.01	**20.17** ±0.01	2.01
Roleplay	10.90 ±0.01	**11.05** ±0.01	1.01	9.93 ±0.01	**27.80** ±0.01	2.80
Stem	10.90 ±0.01	**11.50** ±0.01	1.06	9.83 ±0.01	**14.61** ±0.01	1.49
Writing	**10.92** ±0.01	10.85 ±0.01	0.99	9.94 ±0.01	**24.51** ±0.01	2.46
Average	10.89 ±0.01	**11.88** ±0.01	1.10	9.88 ±0.01	**19.52** ±0.01	1.98

Table 11: Tokens per second on two turns across categories on MT-Redundant using CopySpec and Baseline with Qwen-32B-Instruct ($\gamma = 3$). Results follow the same notation as table 3.

	Turn 1			Turn 2		
Category	**Base Model**	**CopySpec**	**Speed-up**	**Base Model**	**CopySpec**	**Speed-up**
Coding	43.28 ±0.02	**47.16** ±0.10	1.09	37.48 ±0.01	**77.39** ±0.16	2.06
Extraction	39.45 ±0.01	**44.38** ±0.07	1.12	39.34 ±0.01	**73.79** ±0.15	1.88
Humanities	42.94 ±0.02	**44.73** ±0.09	1.04	36.71 ±0.01	**46.73** ±0.09	1.27
Math	44.27 ±0.02	**49.49** ±0.10	1.12	39.85 ±0.01	**84.93** ±0.43	2.13
Reasoning	43.06 ±0.02	**46.51** ±0.09	1.08	39.67 ±0.03	**86.13** ±0.14	2.17
Roleplay	43.14 ±0.11	**45.12** ±0.13	1.05	38.63 ±0.02	**108.37** ±0.18	2.81
Stem	42.96 ±0.04	**45.41** ±0.07	1.06	37.06 ±0.01	**57.54** ±0.11	1.55
Writing	43.50 ±0.01	**44.79** ±0.10	1.03	38.40 ±0.01	**87.91** ±0.12	2.29
Average	42.95 ±0.03	**46.82** ±0.09	1.09	38.51 ±0.01	**78.43** ±0.17	2.04

Table 12: Tokens per second on two turns across categories on MT-Redundant using CopySpec and Baseline with Qwen-7B-Instruct ($\gamma = 3$). Results follow the same notation as table 3.

tion and Coding, where repetitive patterns allow CopySpec to outperform the baseline consistently. Average speed-ups for both turns reinforce the efficiency gains achieved.

In Table 17, the performance of Qwen2.5-32B-Instruct is evaluated. CopySpec achieves significant speed-ups, particularly in the second turn, where contextual repetition becomes more prevalent. Categories like Math and Writing show marked improvements, underscoring CopySpec's ability to handle computationally intensive tasks effectively.

Table 18 highlights the results for Qwen2.5-7B-Instruct. While the base model already performs efficiently, CopySpec further enhances TPS, with average speed-ups exceeding 1.3× in the second turn. These results confirm that CopySpec scales well across different model sizes.

The performance of LLaMa3.1-70B-Instruct is detailed in Table 19. CopySpec achieves consistent improvements across both turns, with substantial gains in computationally intensive categories such as Coding and Extraction. These results demonstrate the robustness of CopySpec when applied to larger models.

Table 20 evaluates LLaMa3.1-8B-Instruct. While the model size is significantly smaller, CopySpec still yields notable improvements, particularly in the second turn, where repetitive token patterns amplify the efficiency of speculative copying.

E.3 Merging with Speculative Decoding on MT-Bench

Finally, Table 21 and Table 22 compares different speculative decoding configurations with and without CopySpec, using Qwen2.5-32B-Instruct as the target model and Qwen2.5-7B-Instruct as the draft model. This analysis explores the impact of varying γ values and draft token counts, demonstrating that the integration of CopySpec with speculative decoding consistently leads to enhanced performance. The results emphasize the adaptability of CopySpec across diverse operational settings.

These tables collectively validate the effectiveness of CopySpec in accelerating large language model inference while maintaining high output quality. The findings in this appendix complement those in Appendix D, reinforcing the method's utility across datasets and configurations.

	Turn 1			Turn 2		
Category	**Base Model**	**CopySpec**	**Speed-up**	**Base Model**	**CopySpec**	**Speed-up**
Coding	5.17 ±0.01	**5.94** ±0.01	1.15	4.81 ±0.01	**10.76** ±0.01	2.24
Extraction	4.90 ±0.01	**5.29** ±0.01	1.08	4.80 ±0.01	**7.60** ±0.01	1.58
Humanities	5.20 ±0.01	**5.39** ±0.01	1.04	4.78 ±0.01	**5.72** ±0.01	1.20
Math	5.23 ±0.01	**5.83** ±0.01	1.12	4.89 ±0.01	**12.58** ±0.01	2.57
Reasoning	5.18 ±0.01	**5.43** ±0.01	1.05	4.92 ±0.01	**8.49** ±0.01	1.73
Roleplay	5.16 ±0.01	**5.28** ±0.01	1.02	4.93 ±0.01	**10.01** ±0.01	2.03
Stem	5.21 ±0.01	**5.43** ±0.01	1.04	4.83 ±0.01	**6.38** ±0.01	1.32
Writing	5.21 ±0.01	**5.27** ±0.01	1.01	4.82 ±0.01	**9.48** ±0.01	1.97
Average	5.16 ±0.01	**5.48** ±0.01	1.06	4.85 ±0.01	**8.88** ±0.01	1.83

Table 13: Tokens per second on two turns across categories on MT-Redundant using CopySpec and Baseline with LLaMa3.1-70B-Instruct ($\gamma = 3$). Results follow the same notation as table 3.

	Turn 1			Turn 2		
Category	**Base Model**	**CopySpec**	**Speed-up**	**Base Model**	**CopySpec**	**Speed-up**
Coding	36.80 ±0.06	**44.31** ±0.07	1.20	34.61 ±0.01	**66.14** ±0.10	1.91
Extraction	35.49 ±0.01	**46.27** ±0.08	1.30	33.78 ±0.01	**71.84** ±0.07	2.13
Humanities	37.31 ±0.01	**40.66** ±0.23	1.09	33.90 ±0.01	**40.01** ±0.06	1.18
Math	37.02 ±0.07	**52.60** ±0.08	1.42	34.94 ±0.05	**64.90** ±0.07	1.86
Reasoning	36.83 ±0.01	**53.24** ±0.01	1.45	34.77 ±0.04	**60.76** ±0.09	1.75
Roleplay	36.85 ±0.02	**40.85** ±0.11	1.11	34.70 ±0.02	**64.18** ±0.13	1.85
Stem	37.28 ±0.04	**41.01** ±0.10	1.10	34.49 ±0.06	**45.01** ±0.09	1.31
Writing	36.94 ±0.02	**39.87** ±0.10	1.08	33.87 ±0.02	**48.01** ±0.09	1.42
Average	36.81 ±0.03	**44.85** ±0.10	1.22	34.38 ±0.03	**57.61** ±0.09	1.67

Table 14: Tokens per second on two turns across categories on MT-Redundant using CopySpec and Baseline with LLaMa3.1-8B-Instruct ($\gamma = 3$). Results follow the same notation as table 3.

F Extra Results on GSM8K

This appendix provides an in-depth analysis of the CopySpec approach applied to self-correcting tasks and speculative decoding. The results demonstrate the effectiveness of CopySpec in improving token processing speed, leveraging context repetition, and enhancing self-correction efficiency without compromising model accuracy.

Table 23 extends the analysis to speculative decoding scenarios, focusing on the performance of CopySpec combined with speculative decoding when the draft model drafts 5 tokens at a time for self-correcting tasks. The table highlights the impact of varying draft model outputs, where CopySpec, combined with speculative decoding ($\gamma = 5$), achieves the best overall performance. Metrics such as TPS and τ show consistent improvements, with the approach accepting a higher average number of tokens per attempt. This configuration effectively balances the benefits of speculative decoding with CopySpec's ability to handle token repetition efficiently.

Table 24 compares the performance of CopySpec and baseline models across three turns using the GSM8K dataset for self-correcting tasks. The metrics include tokens-per-second (TPS), the percentage of tokens copied, and the number of tokens successfully copied (τ) per attempt. CopySpec consistently achieves significant improvements, particularly in the second and third turns, where a larger context size enables better utilization of repetitive patterns. Notable gains are observed in TPS, with improvements exceeding 2× in some configurations, and the percentage of copied tokens highlights CopySpec's efficiency in refining self-corrections.

These results underscore the versatility of CopySpec in enhancing computational efficiency and self-correction capabilities across multiple scenarios. The combination of CopySpec with speculative decoding demonstrates its adaptability to diverse operational settings, paving the way for faster and more accurate large language model inference in tasks requiring iterative refinement.

G MT-Redundant Dataset Examples

This appendix provides one illustrative example from each of the eight categories in our new *MT-*

Category	Turn 1				Turn 2			
	Base Model	Spec. Dec.	Spec. Dec. + Copy ($\gamma = 3$)	Spec. Dec. + Copy ($\gamma = 5$)	Base Model	Spec. Dec.	Spec. Dec. + Copy ($\gamma = 3$)	Spec. Dec. + Copy ($\gamma = 5$)
Coding	10.87 ± 0.01	16.09 ± 0.13	15.88 ± 0.05	**16.09** ± 0.04	9.73 ± 0.01	15.77 ± 0.09	22.02 ± 0.01	**22.50** ± 0.01
Extraction	10.09 ± 0.01	14.20 ± 0.09	15.12 ± 0.09	**15.26** ± 0.01	9.79 ± 0.01	15.17 ± 0.08	18.41 ± 0.05	**18.45** ± 0.05
Humanities	10.85 ± 0.01	12.39 ± 0.10	**12.52** ± 0.03	12.50 ± 0.01	9.75 ± 0.01	12.39 ± 0.07	13.01 ± 0.04	**13.05** ± 0.01
Math	11.01 ± 0.01	17.61 ± 0.10	17.68 ± 0.06	**18.10** ± 0.01	10.05 ± 0.01	16.70 ± 0.11	24.48 ± 0.07	**24.84** ± 0.07
Reasoning	10.80 ± 0.02	13.09 ± 0.10	13.04 ± 0.04	**13.21** ± 0.02	10.05 ± 0.01	14.74 ± 0.06	20.33 ± 0.07	**21.12** ± 0.05
Roleplay	10.90 ± 0.01	11.14 ± 0.08	**11.19** ± 0.04	11.17 ± 0.02	9.93 ± 0.01	16.19 ± 0.10	28.43 ± 0.01	**28.44** ± 0.27
Stem	10.90 ± 0.01	13.33 ± 0.11	13.36 ± 0.06	**13.45** ± 0.01	9.83 ± 0.01	14.16 ± 0.08	16.73 ± 0.02	**16.95** ± 0.03
Writing	10.92 ± 0.01	11.30 ± 0.08	**11.34** ± 0.03	11.33 ± 0.01	9.94 ± 0.01	15.59 ± 0.12	25.46 ± 0.01	**25.16** ± 0.05
Average	10.79 ± 0.01	13.64 ± 0.10	13.77 ± 0.05	**13.89** ± 0.01	9.88 ± 0.01	15.09 ± 0.09	21.11 ± 0.04	**21.31** ± 0.07

Table 15: Tokens-per-second (TPS) performance on the MT-Redundant dataset, using Qwen2.5-32B-Instruct as the target model and Qwen2.5-7B-Instruct as the draft model, where the draft model generates 5 tokens per attempt. Results are presented using the same notation as Table 3 and a γ value of 3, highlighting the impact of varying the draft token count on computational efficiency.

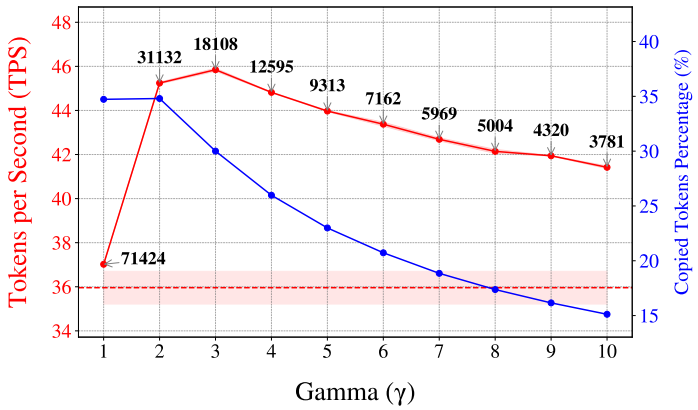

Figure 7: This figure illustrates the relationship between the copying parameter γ and the model's performance on the MT-Bench dataset with the LLaMa3.1-8B-Instruct model. The notations are the same as in fig. 3.

Redundant Dataset. *MT-Redundant* builds upon MT-Bench by modifying the second turn of each conversation into a request for variations or adjustments of the first turn's response, thus emulating real-world scenarios in which users seek revisions to previous outputs. Specifically, we replace the original second-turn prompt in MT-Bench (Zheng et al., 2023) with one that instructs the model to revisit and refine its previous answer. All assistant responses in this appendix are generated using Qwen2.5-72B-Instruct.

H Prompts Used

H.1 Example of Self-Correction

This appendix presents an example of self-correction in code generation on the GSM8K, AIME2024, and AIME2025 datasets. The results are presented in Table 24, Table 25, and Table 26. We generate an initial solution and apply multi-round prompting to iteratively refine and correct the generated code.

To ensure direct answer generation, we prompt the model to explicitly print the computed result, reducing intermediate ambiguities and improving overall accuracy.

H.2 Example of Extractive Summarization

This appendix provides an example of extractive summarization, where key sentences are selected directly from the original text to form a concise summary. The example, generated using Qwen2.5-72B-Instruct, demonstrates how to extract the most relevant information while preserving the original wording. Notably, the Qwen models show an interesting trend on the CNN/DM dataset, where larger models produce more extractive summaries that achieve slightly lower ROUGE-L scores.

H.3 Code Generation on HumanEval

This section presents an example of code generation using Qwen2.5-72B-Instruct on the Hu-

	Turn 1			Turn 2		
Category	**Baseline**	**CopySpec**	**Speed-up**	**Baseline**	**CopySpec**	**Speed-up**
Coding	5.12 ±0.01	**5.62** ±0.01	1.10	4.62 ±0.01	**7.10** ±0.01	1.54
Extraction	4.76 ±0.01	**5.64** ±0.01	1.19	4.48 ±0.01	**6.84** ±0.01	1.53
Humanities	5.09 ±0.01	**5.32** ±0.01	1.04	4.54 ±0.01	**4.98** ±0.01	1.10
Math	5.17 ±0.01	**5.84** ±0.01	1.13	4.81 ±0.01	**6.72** ±0.01	1.40
Reasoning	5.08 ±0.01	**5.69** ±0.01	1.12	4.80 ±0.01	**5.96** ±0.01	1.24
Roleplay	5.06 ±0.01	**5.14** ±0.01	1.02	4.59 ±0.01	**4.68** ±0.01	1.02
Stem	5.12 ±0.01	**5.38** ±0.01	1.05	4.62 ±0.01	**5.32** ±0.01	1.15
Writing	5.12 ±0.01	**5.12** ±0.01	1.01	4.69 ±0.01	**6.09** ±0.01	1.30
Average	5.07 ±0.01	**5.47** ±0.01	1.08	4.64 ±0.01	**5.96** ±0.01	1.28

Table 16: Tokens per second on two turns across categories on MT-Bench using CopySpec and Baseline with Qwen2.5-72B-Instruct ($\gamma = 3$). Results follow the same notation as table 3.

	Turn 1			Turn 2		
Category	**Base Model**	**CopySpec**	**Speed-up**	**Base Model**	**CopySpec**	**Speed-up**
Coding	10.86 ±0.01	**11.67** ±0.01	1.07	9.73 ±0.01	**17.03** ±0.01	1.75
Extraction	10.09 ±0.01	**13.39** ±0.04	1.33	9.59 ±0.01	**15.40** ±0.04	1.61
Humanities	10.86 ±0.01	**11.56** ±0.01	1.06	9.73 ±0.01	**11.14** ±0.01	1.14
Math	11.01 ±0.01	**12.77** ±0.07	1.16	10.15 ±0.01	**13.35** ±0.03	1.32
Reasoning	10.82 ±0.01	**12.18** ±0.01	1.13	10.22 ±0.01	**11.54** ±0.01	1.13
Roleplay	10.90 ±0.01	**11.04** ±0.01	1.01	10.16 ±0.01	**10.37** ±0.01	1.02
Stem	10.89 ±0.01	**11.51** ±0.01	1.06	9.84 ±0.01	**11.50** ±0.01	1.17
Writing	**10.90** ±0.01	10.82 ±0.02	0.99	9.99 ±0.01	**13.25** ±0.01	1.33
Average	10.91 ±0.01	**11.86** ±0.01	1.09	9.92 ±0.01	**12.57** ±0.01	1.27

Table 17: Tokens per second on two turns across categories on MT-Bench using CopySpec and Baseline with Qwen2.5-32B-Instruct ($\gamma = 3$). Results follow the same notation as table 3.

manEval dataset. The model generates an initial code implementation based on a given problem description and produces a self-contained Python script that correctly solves the task. The input consists of a problem description specifying the function signature, expected behavior, and an example test case. The generated solution includes function definitions, type hints, and example test cases to ensure correctness.

	Turn 1			Turn 2		
Category	**Base Model**	**CopySpec**	**Speed-up**	**Base Model**	**CopySpec**	**Speed-up**
Coding	43.04 ±0.25	**47.22** ±0.02	1.10	37.43 ±0.08	**60.06** ±0.01	1.60
Extraction	39.50 ±0.06	**44.41** ±0.01	1.12	38.94 ±0.09	**52.85** ±0.01	1.36
Humanities	43.06 ±0.07	**44.79** ±0.01	1.04	36.82 ±0.08	**43.05** ±0.01	1.17
Math	44.40 ±0.16	**49.46** ±0.12	1.11	39.39 ±0.36	**53.45** ±0.01	1.36
Reasoning	43.49 ±0.36	**46.57** ±0.01	1.07	40.96 ±0.19	**46.76** ±0.01	1.14
Roleplay	43.43 ±0.05	**45.35** ±0.01	1.04	38.72 ±0.08	**39.89** ±0.01	1.03
Stem	43.30 ±0.07	**45.47** ±0.01	1.05	37.34 ±0.09	**43.61** ±0.01	1.17
Writing	43.58 ±0.06	**44.72** ±0.01	1.03	38.80 ±0.08	**55.90** ±0.01	1.44
Average	42.80 ±0.13	**46.98** ±0.03	1.10	38.25 ±0.13	**49.57** ±0.01	1.30

Table 18: Tokens per second on two turns across categories on MT-Bench using CopySpec and Baseline with Qwen2.5-7B-Instruct ($\gamma = 3$). Results follow the same notation as table 3.

	Turn 1			Turn 2		
Category	**Base Model**	**CopySpec**	**Speed-up**	**Base Model**	**CopySpec**	**Speed-up**
Coding	5.18 ±0.01	**5.94** ±0.01	1.15	4.79 ±0.01	**7.63** ±0.01	1.59
Extraction	4.91 ±0.01	**5.28** ±0.01	1.08	4.65 ±0.01	**7.03** ±0.01	1.51
Humanities	5.21 ±0.01	**5.39** ±0.01	1.04	4.77 ±0.01	**5.35** ±0.01	1.12
Math	5.23 ±0.01	**5.83** ±0.01	1.12	4.96 ±0.01	**6.57** ±0.01	1.32
Reasoning	5.16 ±0.01	**5.43** ±0.01	1.05	4.96 ±0.01	**5.56** ±0.01	1.12
Roleplay	5.17 ±0.01	**5.28** ±0.01	1.02	4.94 ±0.01	**5.90** ±0.01	1.19
Stem	5.22 ±0.01	**5.41** ±0.01	1.04	4.85 ±0.01	**5.54** ±0.01	1.14
Writing	5.21 ±0.01	**5.27** ±0.01	1.01	4.81 ±0.01	**6.42** ±0.01	1.33
Average	5.16 ±0.01	**5.48** ±0.01	1.06	4.84 ±0.01	**6.25** ±0.01	1.29

Table 19: Tokens per second on two turns across categories on MT-Bench using CopySpec and Baseline with LLaMa3.1-70B-Instruct ($\gamma = 3$). Results follow the same notation as table 3.

	Turn 1			Turn 2		
Category	**Base Model**	**CopySpec**	**Speed-up**	**Base Model**	**CopySpec**	**Speed-up**
Coding	36.86 ±0.01	**44.35** ±0.06	1.20	34.42 ±0.01	**53.22** ±0.06	1.55
Extraction	35.32 ±0.07	**46.27** ±0.03	1.31	33.71 ±0.01	**51.48** ±0.06	1.53
Humanities	37.20 ±0.01	**40.88** ±0.06	1.10	33.78 ±0.02	**40.61** ±0.05	1.20
Math	36.99 ±0.01	**52.46** ±0.24	1.42	34.96 ±0.01	**58.47** ±0.07	1.67
Reasoning	36.70 ±0.04	**53.33** ±0.06	1.45	34.76 ±0.01	**53.86** ±0.06	1.55
Roleplay	36.77 ±0.01	**40.89** ±0.06	1.11	34.56 ±0.01	**49.16** ±0.06	1.42
Stem	37.19 ±0.01	**41.06** ±0.06	1.10	34.47 ±0.01	**41.88** ±0.06	1.21
Writing	36.85 ±0.01	**39.91** ±0.06	1.08	33.78 ±0.01	**38.72** ±0.06	1.15
Average	36.73 ±0.08	**44.89** ±0.08	1.22	34.30 ±0.01	**48.42** ±0.06	1.41

Table 20: Tokens per second on two turns across categories on MT-Bench using CopySpec and Baseline with LLaMa3.1-8B-Instruct ($\gamma = 3$). Results follow the same notation as table 3.

Category	Turn 1				Turn 2			
	Base Model	**Spec. Dec.**	**Spec. Dec. + Copy** ($\gamma = 3$)	**Spec. Dec. + Copy** ($\gamma = 5$)	**Base Model**	**Spec. Dec.**	**Spec. Dec. + Copy** ($\gamma = 3$)	**Spec. Dec. + Copy** ($\gamma = 5$)
Coding	10.86 ±0.01	15.97 ±0.01	15.91 ±0.06	**16.16** ±0.05	9.73 ±0.01	14.81 ±0.01	19.94 ±0.01	**19.97** ±0.11
Extraction	10.09 ±0.01	14.22 ±0.01	**15.39** ±0.06	15.36 ±0.05	9.59 ±0.01	14.55 ±0.01	**16.71** ±0.05	16.26 ±0.01
Humanities	10.86 ±0.01	13.66 ±0.01	**13.89** ±0.01	13.87 ±0.04	9.73 ±0.01	12.30 ±0.01	**12.93** ±0.02	12.85 ±0.01
Math	11.01 ±0.01	17.02 ±0.03	17.30 ±0.01	**17.32** ±0.02	10.15 ±0.01	15.38 ±0.01	16.04 ±0.06	**16.61** ±0.02
Reasoning	10.82 ±0.01	14.02 ±0.01	**14.34** ±0.02	14.26 ±0.01	10.23 ±0.01	12.99 ±0.01	13.18 ±0.02	**13.42** ±0.05
Roleplay	10.90 ±0.01	12.86 ±0.02	12.88 ±0.04	**12.94** ±0.01	10.16 ±0.01	12.11 ±0.01	12.18 ±0.03	**12.24** ±0.02
Stem	10.89 ±0.01	14.29 ±0.06	14.36 ±0.03	**14.47** ±0.02	9.84 ±0.01	13.13 ±0.01	13.71 ±0.01	**13.77** ±0.05
Writing	10.90 ±0.01	12.65 ±0.02	12.69 ±0.02	**12.71** ±0.03	9.99 ±0.01	11.69 ±0.01	**13.54** ±0.03	13.31 ±0.01
Average	10.79 ±0.01	14.34 ±0.02	14.60 ±0.03	**14.64** ±0.03	9.93 ±0.01	13.37 ±0.01	14.78 ±0.03	**14.80** ±0.04

Table 21: Tokens-per-second (TPS) performance on the MT-Bench dataset, using Qwen2.5-32B-Instruct as the target model and Qwen2.5-7B-Instruct as the draft model, where the draft model generates 3 tokens per attempt. Results are presented using the same notation as Table 3 and a γ value of 3, showcasing the improvements in speed and efficiency enabled by CopySpec.

Category	Turn 1				Turn 2			
	Base Model	**Spec. Dec.**	**Spec. Dec. + Copy** ($\gamma = 3$)	**Spec. Dec. + Copy** ($\gamma = 5$)	**Base Model**	**Spec. Dec.**	**Spec. Dec. + Copy** ($\gamma = 3$)	**Spec. Dec. + Copy** ($\gamma = 5$)
Coding	10.86 ±0.01	**16.09** ±0.09	15.89 ±0.02	16.06 ±0.05	9.73 ±0.01	15.72 ±0.06	20.08 ±0.03	**20.22** ±0.13
Extraction	10.09 ±0.01	14.28 ±0.06	15.08 ±0.01	**15.20** ±0.02	9.59 ±0.01	15.46 ±0.06	16.89 ±0.01	**16.93** ±0.01
Humanities	10.86 ±0.01	12.41 ±0.07	**12.52** ±0.01	12.45 ±0.04	9.73 ±0.01	11.67 ±0.04	**12.08** ±0.01	12.02 ±0.02
Math	11.01 ±0.01	17.60 ±0.15	17.76 ±0.02	**17.95** ±0.06	10.15 ±0.01	16.22 ±0.06	16.57 ±0.02	**17.08** ±0.01
Reasoning	10.82 ±0.01	**13.04** ±0.01	12.94 ±0.02	12.97 ±0.10	10.23 ±0.01	11.92 ±0.06	12.25 ±0.01	**12.29** ±0.04
Roleplay	10.90 ±0.01	11.15 ±0.04	**11.18** ±0.01	11.14 ±0.03	10.16 ±0.01	11.09 ±0.05	**11.11** ±0.01	11.13 ±0.03
Stem	10.89 ±0.01	13.34 ±0.07	13.35 ±0.04	**13.37** ±0.04	9.84 ±0.01	12.87 ±0.05	**13.12** ±0.02	13.12 ±0.03
Writing	10.90 ±0.01	11.32 ±0.04	**11.33** ±0.01	11.20 ±0.11	9.99 ±0.01	10.71 ±0.06	**11.89** ±0.01	11.74 ±0.01
Average	10.79 ±0.01	13.65 ±0.07	13.76 ±0.02	**13.79** ±0.06	9.93 ±0.01	13.21 ± 0.06	14.25 ± 0.02	**14.32** ± 0.04

Table 22: Tokens-per-second (TPS) performance on the MT-Bench dataset, using Qwen2.5-32B-Instruct as the target model and Qwen2.5-7B-Instruct as the draft model, where the draft model generates 5 tokens per attempt. Results are presented using the same notation as Table 3 and a γ value of 3, illustrating the scalability and efficiency of CopySpec under varied settings.

Variant	Turn 1				Turn 2				Turn 3			
	% Copied	**Tokens/s**	τ_1	τ_2	**% Copied**	**Tokens/s**	τ_1	τ_2	**% Copied**	**Tokens/s**	τ_1	τ_2
Base Model	–	10.25±0.01	–	–	–	10.17±0.01	–	–	–	8.68±0.01	–	–
CopySpec ($\gamma = 3$)	5.76%	10.13±0.01	0.58	–	44.17%	**15.72**±0.01	4.90	–	82.79%	**21.89**±0.01	7.67	–
CopySpec ($\gamma = 5$)	1.01%	9.91±0.02	0.72	–	40.67%	14.79±0.01	6.96	–	82.78%	21.39±0.02	8.70	–
Spec. Dec.	–	12.92±0.02	–	3.77	–	12.27±0.01	–	3.36	–	11.44±0.01	–	4.30
Spec. Dec. + Copy ($\gamma = 3$)	1.47%	12.67±0.02	0.53	3.77	40.23%	14.65±0.02	6.08	2.52	81.18%	20.81±0.01	7.71	3.39
Spec. Dec. + Copy ($\gamma = 5$)	0.30%	**12.99**±0.01	0.55	3.78	38.93%	14.95±0.01	7.81	2.59	81.84%	21.51±0.02	8.72	3.40

Table 23: Performance comparison for self-correcting tasks when the draft model generates 5 tokens at a time. Qwen2.5-32B-Instruct is the target model, and Qwen2.5-7B-Instruct is the draft model. τ_1 refers to the average tokens accepted by CopySpec, and τ_2 refers to the average number of tokens accepted by the draft model. The accuracy of the model improves by from 92% to 93%. The average TPS is highest for Spec. Dec. + Copy(γ = 5) at 15.59 while CopySpec alone achieves 14.84 TPS on average.

Model	Variant	Turn 1				Turn 2			Turn 3			
(Instruct)		% Copied	Tokens/s	τ	Acc	% Copied	Tokens/s	τ	% Copied	Tokens/s	τ	Acc
Qwen2.5-72B	CopySpec	6.12%	4.71±0.01	0.63	94%	47.49%	**7.49**±0.01	4.35	88.68%	**10.59**±0.01	7.94	**96%**
	Base Model	–	**4.74**±0.01	–		–	4.76±0.01	–	–	3.98±0.01	–	
Qwen2.5-32B	CopySpec	5.76%	10.13±0.01	0.58	92%	44.17%	**15.72**±0.01	4.90	82.78%	**21.89**±0.01	7.67	**93%**
	Base Model	–	**10.25**±0.01	–		–	10.17±0.01	–	–	8.68±0.01	–	
Qwen2.5-7B	CopySpec	9.36%	**41.01**±0.44	0.87	84%	60.34%	**75.34**±0.68	5.65	84.23%	**93.68**±0.26	7.35	**85%**
	Base Model	–	40.29±0.02	–		–	39.67±0.05	–	–	35.63±0.01	–	

Table 24: Performance comparison on the GSM8K dataset for self-correcting tasks across three turns, using CopySpec and the base model with Qwen2.5-Instruct variants. The table highlights significant improvements in tokens-per-second (TPS), percentage of tokens copied, and the number of tokens successfully copied (τ) per attempt when attempting to copy 10 tokens, with $\gamma = 3$. These results demonstrate the effectiveness of CopySpec in leveraging increased context size and refining self-correction efficiency without compromising accuracy.

Writing - Question (84): Write a persuasive email to convince your introverted friend, who dislikes public speaking, to volunteer as a guest speaker at a local event. Use compelling arguments and address potential objections. Please be concise.

Assistant:

```
Subject: An Opportunity to Shine and Inspire!

Hi [Friend's Name],

I hope this message finds you well! I wanted to share an exciting opportunity that I think would
be perfect for you. The [Local Organization] is hosting an event next month and they're looking
for guest speakers. I immediately thought of you because of your unique insights and the inspiring
journey you've had.

I understand that public speaking isn't your favorite thing, but hear me out on why this could be
a fantastic experience for you:

1. **Share Your Expertise**: You have so much valuable knowledge and experience to offer. [...]
```

Follow-up Question: Rewrite your answer and make sure to replace any placeholder with actual names.

Assistant:

```
Subject: An Opportunity to Shine and Inspire!

Hi Alex,

I hope this message finds you well! I wanted to share an exciting opportunity that I think would
be perfect for you. The Community Development Association is hosting an event next month and
they're looking for guest speakers. I immediately thought of you because of your unique insights
and the inspiring journey you've had.

I understand that public speaking isn't your favorite thing, but hear me out for a moment. This
could be a fantastic chance for you to:

1. **Share Your Expertise**: You have so much valuable knowledge and experience that [...]
```

GPT-4-0613 Judgment:

```
The assistant's response is excellent. It followed the user's instructions to rewrite the previous
message, replacing placeholders with actual names. The assistant used the name \"Alex\" for the
friend and \"Jamie\" for the sender, and replaced \"[Local Organization]\" with \"Community
Development Association\". The assistant also added specific details about Alex's expertise,
making the message more personalized and relevant. The assistant's response is accurate, relevant,
and detailed. It maintains the persuasive and supportive tone of the original message. Therefore,
the assistant's performance is commendable. Rating: [[10]]
```

Figure 8: Examples from the **Writing** category (Slides 81–90). This category focuses on creative and formal writing tasks, such as rephrasing, summarizing, or generating alternative drafts. The second turn typically requests modifications or alternate versions of the initial written piece. "[...]" represents truncated output that didn't fit in the image.

Roleplay - Question (97): Act as a math teacher. I will provide some mathematical equations or concepts, and it will be your job to explain them in easy-to-understand terms. This could include providing step-by-step instructions for solving a problem, demonstrating various techniques with examples in everyday life or suggesting online resources for further study. My first request is \"I need help understanding how probability works.\"

Assistant:

```
Absolutely, I'd be happy to help you understand probability! Probability is a way of quantifying
the likelihood of an event happening. It's used in many areas, from weather forecasting to making
decisions in games and even in everyday life.

### Basic Concepts

1. **Event**: An event is something that happens. For example, rolling a die and getting a 6 is an
event.
2. **Outcome**: An outcome is a possible result of an event. For example, when you roll a die, the
possible outcomes are 1, 2, 3, 4, 5, and 6. [...]
```

Follow-up Question: Return your last output as a JSON in which each couple of lines represent an entry.

Assistant:

```
Certainly! Here is the information from my last output formatted as a JSON object, where each
couple of lines represents an entry:

```json
{
  "Basic Concepts": {
    "Event": "An event is something that happens. For example, rolling a die and getting a 6 is an
event.",
    "Outcome": "An outcome is a possible result of an event. For example, when you roll a die, the
possible outcomes are 1, 2, 3, 4, 5, and 6.",
    "Sample Space": "The sample space is the set of all possible outcomes. For a die, the sample
space is {1, 2, 3, 4, 5, 6}.",
    "Probability": "The probability of an event is a number between 0 and 1 that represents the
likelihood of that event happening. A probability of 0 means the event will never happen, and a
probability of 1 means the event will always happen."
  },
  "Calculating Probability": "The probability of an event is calculated using the formula: [...]
```

GPT-4-0613 Judgment:

```
The assistant's response is excellent. It provides a comprehensive and easy-to-understand
explanation of the concept of probability. The assistant starts by defining the basic concepts
related to probability, then moves on to explain how to calculate probability with clear examples.
The assistant also provides real-life applications of probability and suggests online resources
for further study. The assistant even provides practice problems for the user to test their
understanding. The response is accurate, detailed, and highly relevant to the user's request.
Therefore, I would rate this response as a perfect score. Rating: [[10]]
```

Figure 9: Examples from the **Roleplay** category (Slides 91–100). Tasks in this category simulate real-world or imaginative scenarios, requiring the model to adjust its responses based on dynamic user requests and context shifts. "[...]" represents truncated output that didn't fit in the image.

Model	**Variant**	**Turn 1**		**Turn 2**	**Turn 3**		**Avg (Tokens/s)**
(Instruct)		**Tokens/s**	**Acc**	**Tokens/s**	**Tokens/s**	**Acc**	
Qwen2.5-32B	Base Model	10.30±0.01		9.79±0.01	8.61±0.01		9.44±0.01
	PLD (Lookup=3)	10.35±0.01	13.33%	12.14±0.01	12.42±0.01	23.33%	11.82±0.01
	COPYSPEC ($\gamma = 3$)	**12.40**±0.01		**14.78**±0.01	**16.47**±0.01		**14.46**±0.01
Qwen2.5-7B	Base Model	29.03±0.01		28.55±0.01	27.05±0.01		28.23±0.01
	PLD (Lookup=3)	**33.47**±0.03	10.00%	65.66±0.05	56.23±0.56	10.00%	50.19±0.13
	COPYSPEC ($\gamma = 3$)	32.41±0.02		**132.45**±0.17	**106.43**±0.01		**62.47**±0.03

Table 25: **AIME 2024 (self-refine).** Throughput (tokens/s) and accuracy grouped by turn. Turn 2 is a thinking-only round (no accuracy). Accuracies are identical across variants under greedy decoding because the output is the same, so we show them once per model. We see that our technique significantly outperforms PLD and the base model on the later turns where we have repetition.
```

Reasoning - Question (104): David has three sisters. Each of them has one brother. How many brothers does David have?

Assistant:

```
David has three sisters, and each of them has one brother. Since David is that one brother, he
does not have any other brothers. Therefore, David has **0** brothers.
```

Follow-up Question: Repeat your reasoning but David has 4 sisters now.

Assistant:

```
If David has four sisters, and each of them has one brother, that one brother is David himself.
Therefore, David still does not have any other brothers. So, David has **0** brothers.
```

GPT-4-0613 Judgment:

```
The assistant's response is correct. The assistant correctly reasoned that if each of David's
sisters has one brother, then that brother is David himself. This is true regardless of whether
David has three sisters or four. The assistant's response is clear and logically sound. However,
the reference answer is incorrect, as it suggests that David would have two brothers if each of
his sisters had two brothers, which is not the case. Therefore, the assistant's response is more
accurate than the reference answer. \n\nRating: [[10]]
```

Figure 10: Examples from the **Reasoning** category (Slides 101–110). This category evaluates logical and analytical thinking, with prompts asking models to refine prior explanations or provide additional clarifications in the second turn.

Model	Variant	Turn 1		Turn 2	Turn 3		Avg (Tokens/s)
(Instruct)		Tokens/s	Acc	Tokens/s	Tokens/s	Acc	
Qwen2.5-32B	Base Model	10.14±0.01		9.59±0.01	8.48±0.01		9.35±0.01
	PLD (Lookup=3)	10.95±0.01	16.66%	11.92±0.02	13.52±0.01	20.00%	12.07±0.02
	COPYSPEC ($\gamma = 3$)	**12.70**±0.01		**15.01**±0.02	**16.64**±0.01		**14.70**±0.01
Qwen2.5-7B	Base Model	29.17±0.31		28.68±0.31	27.41±0.28		28.42±0.31
	PLD (Lookup=3)	31.44±0.06	13.33%	75.42±0.06	67.44±0.03	13.33%	51.27±0.06
	COPYSPEC ($\gamma = 3$)	**31.66**±0.05		**122.08**±0.07	**104.85**±0.22		**62.00**±0.05

Table 26: **AIME 2025 (self-refine).** Throughput (tokens/s) and accuracy grouped by turn. Turn 2 is a thinking-only round (no accuracy). Accuracies are identical across variants under greedy decoding because the output is the same, so we show them once per model. We see that our technique significantly outperforms PLD and the base model on the later turns where we have repetition.

Math - Question (119): Benjamin went to a bookstore and purchased a variety of books. He bought 5 copies of a sci-fi novel, each priced at $20, 3 copies of a history book priced at $30 each, and 2 copies of a philosophy book for $45 each.
What was the total cost of his purchases?

Assistant:

```
To find the total cost of Benjamin's purchases, we need to calculate the cost of each type of book
and then sum these costs.

1. **Sci-fi novels:**
   - Number of copies: 5
   - Price per copy: $20
   - Total cost for sci-fi novels: \( 5 \times 20 = 100 \) dollars

2. **History books:**
   - Number of copies: 3
   - Price per copy: $30 [...]
```

Follow-up Question: Using the same reasoning as before, calculate the total cost of Benjamin's purchases if he bought 5 copies of a sci-fi novel priced at $25 each, 3 copies of a history book priced at $35 each, and 2 copies of a philosophy book priced at $50 each. Only change your answer by plugging in the new numbers.

Assistant:

```
To find the total cost of Benjamin's purchases with the new prices, we will follow the same steps:

1. **Sci-fi novels:**
   - Number of copies: 5
   - Price per copy: $25
   - Total cost for sci-fi novels: \( 5 \times 25 = 125 \) dollars

2. **History books:**
   - Number of copies: 3
   - Price per copy: $35
   - Total cost for history books: \( 3 \times 35 = 105 \) dollars

3. **Philosophy books:**
   - Number of copies: 2
   - Price per copy: $50
   - Total cost for philosophy books: \( 2 \times 50 = 100 \) dollars [...]
```

GPT-4-0613 Judgment:

```
The assistant's response is correct and helpful. It accurately calculates the total cost of
Benjamin's purchases with the new prices, following the same reasoning as in the previous
question. The assistant breaks down the calculation for each type of book and then sums the costs,
providing a clear and detailed explanation. The assistant's answer matches the user's request to
\"only change your answer by plugging in the new numbers.\" The assistant's answer is also
consistent with the reference answer, except that the reference answer incorrectly calculates the
cost of the history books and philosophy books with a 25% markup. The assistant [...]
```

Figure 11: Examples from the **Math** category (Slides 111–120). This category challenges the model to revise or elaborate mathematical solutions, often clarifying steps or offering alternative solution paths when asked. "[...]" represents truncated output that didn't fit in the image.

Coding - Question (126): Implement a function to find the median of two sorted arrays of different sizes with O(1) space complexity and O(n) time complexity.

Assistant:

Finding the median of two sorted arrays of different sizes with O(1) space complexity and O(n) time complexity is a challenging problem. The key idea is to use a linear-time selection algorithm, such as the "median of medians" algorithm, to find the k-th smallest element in the combined array. However, for simplicity and clarity, we can use a more straightforward approach that merges the two arrays while keeping track of the median.

Here's a step-by-step implementation:

1. **Merge the two arrays**: We will merge the two arrays while keeping track of the elements in a way that allows us to find the median.
2. **Find the median**: Depending on the total number of elements, the median will be either the middle element (if the total number of elements is odd) or the average of the two middle elements (if the total number of elements is even). **[...]**

Follow-up Question: Modify the function to check if either array is empty and print a message if it is.

Assistant:

Certainly! We can modify the function to include a check for empty arrays and print a message if either array is empty. Here's the updated implementation:

```python
def find_median_sorted_arrays(nums1, nums2):
    # Check if either array is empty
    if not nums1:
        print("The first array is empty.")
        if not nums2:
            print("Both arrays are empty. Cannot determine the median.")
            return None
        return find_median_single_array(nums2)

    if not nums2:
        print("The second array is empty.")
        return find_median_single_array(nums1)

    # Merge the two sorted arrays [...]
```

GPT-4-0613 Judgment:

The assistant's answer is incorrect. The user asked for a solution with O(1) space complexity and O(n) time complexity. The assistant's solution does not meet the space complexity requirement because it creates a new list, `merged`, which stores all elements from the two input arrays. This results in a space complexity of O(n), where n is the total number of elements in the two input arrays. The reference answer correctly states that it's not possible to find the median of two sorted arrays of different sizes with O(1) space complexity and O(n) time complexity. **[...]**

Figure 12: Examples from the **Coding** category (Slides 121–130). This category covers programming-related tasks such as debugging, refactoring, or implementing variants of a provided code snippet in response to a user's request. "[...]" represents truncated output that didn't fit in the image.

Extraction - Question (135): Identify the countries, their capitals, and the languages spoken in the following sentences. Output in JSON format.
a) Amidst the idyllic vistas, Copenhagen, Denmark's capital, captivates visitors with its thriving art scene and the enchanting Danish language spoken by its inhabitants.
b) Within the enchanting realm of Eldoria, one discovers Avalore, a grandiose city that emanates an ethereal aura. Lumina, a melodious language, serves as the principal mode of communication within this mythical abode.
c) Nestled amidst a harmonious blend of age-old customs and contemporary wonders, Buenos Aires, the capital of Argentina, stands as a bustling metropolis. It is a vibrant hub where the expressive Spanish language holds sway over the city's inhabitants.

Assistant:

```
json
[
  {
    "country": "Denmark",
    "capital": "Copenhagen",
    "languages": ["Danish"]
  },
  {
    "country": "Eldoria",
    "capital": "Avalore", [...]
```

Follow-up Question: Update your output to include whether the country is real or fictional.

Assistant:

```
json
[
  {
    "country": "Denmark",
    "capital": "Copenhagen",
    "languages": ["Danish"],
    "is_real": true
  },
  {
    "country": "Eldoria",
    "capital": "Avalore",
    "languages": ["Lumina"],
    "is_real": false [...]
```

GPT-4-0613 Judgment:

```
The assistant's response is accurate, relevant, and helpful. The user asked for an update to the
previous JSON output to include whether the country is real or fictional. The assistant correctly
identified Denmark and Argentina as real countries and Eldoria as a fictional one, and updated the
JSON output accordingly. The assistant's response is detailed and directly addresses the user's
request, providing the information in the requested format. Therefore, the assistant's [...]
```

Figure 13: Examples from the **Extraction** category (Slides 131–140). This category focuses on pulling specific information from the model's previous response or restructuring it (e.g., lists, bullet points) according to user specifications. "[...]" represents truncated output that didn't fit in the image.

Stem - Question (150): How have the Alps and Rhine River influenced settlement and agriculture in Western Europe? List three impacts.

Assistant:

```
The Alps and the Rhine River have significantly influenced settlement and agriculture in Western
Europe in various ways. Here are three key impacts:

1. **Natural Barriers and Strategic Locations**:
   - **Alps**: The Alps have acted as a natural barrier, influencing the distribution of
settlements. Towns and cities often developed in valleys and passes, such as the Brenner Pass,
which served as important trade routes. These locations became strategic points for commerce and
defense.
   - **Rhine River**: The Rhine River has been a crucial transportation corridor, facilitating
trade and communication. Cities like Basel, Strasbourg, and Cologne grew along its banks, becoming
major economic and cultural centers. The river's navigability has made it a vital artery for the
movement of goods and people.

2. **Agricultural Practices and Land Use**: [...]
```

Follow-up Question: Include the impacts you described previously and propose a simple experimental design to validate the first impact, using specific methods and metrics.

Assistant:

```
Certainly! Below is a summary of the impacts of the Alps and the Rhine River on settlement and
agriculture in Western Europe, followed by a proposed experimental design to validate the first
impact (settlement patterns influenced by the Alps).

### Summary of Impacts

1. **Settlement Patterns:**
   - **Alps as a Natural Barrier:** The Alps have acted as a natural barrier, influencing where
and how settlements developed. Areas with more accessible passes and valleys, such as the Swiss
Plateau and the Alpine forelands, saw denser populations due to easier travel and communication.
Conversely, higher and more rugged areas remained sparsely populated.
   - **Rhine River as a Transportation Route:** The Rhine River has been a crucial transportation
route, facilitating trade and communication. Cities like Basel, Strasbourg, and Cologne grew along
its banks, becoming important economic and cultural centers. The river's navigability has made it
a vital artery for the movement of goods and people, supporting urban development. [...]
```

GPT-4-0613 Judgment:

```
The assistant's response is highly detailed, relevant, and accurate. It provides a comprehensive
experimental design to validate the first impact of the Alps and Rhine River on settlement and
agriculture in Western Europe. The assistant outlines a clear hypothesis, methods for data
collection and analysis, case studies, metrics, and expected outcomes. The assistant also uses
specific methods such as GIS, spatial analysis, network analysis, and statistical analysis, and
provides specific metrics like settlement density, trade route connectivity, defensive structures,
and economic indicators. The assistant's response is also creative in its approach to validating
the impact. However, the response is cut off at the end, which leaves the expected outcomes
section incomplete. Rating: [[9]]
```

Figure 14: Examples from the **STEM** category (Slides 141–150). This category addresses a variety of scientific and technical topics, requiring models to adapt or refine explanations, data, or methodologies in the second turn. "[...]" represents truncated output that didn't fit in the image.

Humanities - Question (159): What are some business etiquette norms when doing business in Japan?

Assistant:

Doing business in Japan requires an understanding of the country's unique cultural and social norms. Here are some key business etiquette norms to keep in mind:

1. **Greetings and Bowing**:
   - **Bowing**: Bowing is a common form of greeting in Japan. The depth and duration of the bow can vary depending on the situation and the relationship between the individuals. A slight bow is generally appropriate for initial greetings.
   - **Handshakes**: While handshakes are becoming more common, especially with international business partners, they are often accompanied by or followed by a bow.

2. **Business Cards (Meishi)**:
   - **Exchange**: Business cards are exchanged at the beginning of a meeting. Present your card with both hands, holding it by the corners so that the text faces the recipient. **[...]**

Follow-up Question: Repeat the business etiquette norms you described earlier and explain how each of them can be specifically applied in a car wash business setting to enhance customer satisfaction.

Assistant:

Certainly! Here are the business etiquette norms for Japan, along with specific applications in a car wash business setting to enhance customer satisfaction:

1. **Greetings and Bowing**:
   - **Application**: Greet customers with a polite bow and a smile as they arrive. This shows respect and creates a welcoming atmosphere. For example, a slight bow and a warm "Irasshaimase" (Welcome) can make customers feel valued from the moment they enter your premises.

2. **Business Cards (Meishi)**:
   - **Application**: While business cards might not be as relevant in a car wash setting, you can still provide a professional and clean business card to regular customers or those who inquire about services. Ensure the card is presented respectfully, with both hands, and includes all necessary contact information.

3. **Dress Code**:
   - **Application**: Ensure that all staff members are dressed professionally and neatly. Uniforms should be clean and well-maintained. This conveys a sense of professionalism and **[...]**

GPT-4-0613 Judgment:

The assistant's response is highly detailed, relevant, and creative. It successfully translates the general business etiquette norms in Japan to a specific context, which is a car wash business. The assistant provides practical examples for each norm, showing how they can be applied to enhance customer satisfaction in a car wash setting. The assistant's response is also accurate, as it maintains the essence of the original norms while adapting them to a new context. The assistant's response is thorough and demonstrates a deep understanding of both the general norms and the specific business context. Therefore, the assistant's response is of high quality. Rating: [[10]]

Figure 15: Examples from the **Humanities** category (Slides 151–160). This category includes topics like literary analysis, historical context, or philosophical discussion, with the second turn often requesting deeper insight or alternate perspectives. "[...]" represents truncated output that didn't fit in the image.

Question: Please produce an *extractive* summary of the article below by choosing 2 or 3 key sentences from the original text:

(CNN)Five Americans who were monitored for three weeks at an Omaha, Nebraska, hospital after being exposed to Ebola in West Africa have been released, a Nebraska Medicine spokesman said in an email Wednesday. One of the five had a heart-related issue on Saturday and has been discharged but hasn't left the area, Taylor Wilson wrote. The others have already gone home. They were exposed to Ebola in Sierra Leone in March, but none developed the deadly virus. They are clinicians for Partners in Health, a Boston-based aid group. They all had contact with a colleague who was diagnosed with the disease and is being treated at the National Institutes of Health in Bethesda, Maryland. As of Monday, that health care worker is in fair condition. The Centers for Disease Control and Prevention in Atlanta has said the last of 17 patients who were being monitored are expected to be released by Thursday. More than 10,000 people have died in a West African epidemic of Ebola that dates to December 2013, according to the World Health Organization. Almost all the deaths have been in Guinea, Liberia and Sierra Leone. Ebola is spread by direct contact with the bodily fluids of an infected person.

Return only sentences from the original text that best capture the main ideas.

Assistant:

```
Five Americans who were monitored for three weeks at an Omaha, Nebraska, hospital after being
exposed to Ebola in West Africa have been released, a Nebraska Medicine spokesman said in an email
Wednesday. They were exposed to Ebola in Sierra Leone in March, but none developed the deadly
virus. The Centers for Disease Control and Prevention in Atlanta has said the last of 17 patients
who were being monitored are expected to be released by Thursday.
```

Figure 16: An example of self-correction in code generation on CNN/DM using Qwen2.5-72B-Instruct, demonstrating abstractive summarization.

Question: Please provide a self-contained Python script that solves the following problem in a markdown code block:

```
from typing import List, Tuple


def rolling_max(numbers: List[int]) -> List[int]:
    """ From a given list of integers, generate a list of rolling maximum element found until given moment
    in the sequence.
    >>> rolling_max([1, 2, 3, 2, 3, 4, 2])
    [1, 2, 3, 3, 3, 4, 4]
    """
```

Assistant:

````
Below is a Python script with a self-contained function that solves the problem and passes corresponding tests:
```python
from typing import List

def rolling_max(numbers: List[int]) -> List[int]:
    """ From a given list of integers, generate a list of rolling maximum element found until given moment
    in the sequence.
    >>> rolling_max([1, 2, 3, 2, 3, 4, 2])
    [1, 2, 3, 3, 3, 4, 4]
    """
    if not numbers:
        return []

    max_so_far = numbers[0]
    result = [max_so_far]

    for number in numbers[1:]:
        if number > max_so_far:
            max_so_far = number
        result.append(max_so_far)

    return result

# Example usage and test
if __name__ == "__main__":
    test_numbers = [1, 2, 3, 2, 3, 4, 2]
    print(rolling_max(test_numbers))  # Output: [1, 2, 3, 3, 3, 4, 4]
```
````

Figure 17: Example of code generation on the HumanEval dataset using Qwen2.5-72B-Instruct, demonstrating the model's ability to produce a self-contained Python solution with function definitions, type hints, and test cases.