Efficient Framework for Large Language Models

Dynamic Pruning, Layer-Specific Quantization, and Speculative Copying

Author: Razvan-Gabriel Dumitru **Advisor:** Mihai Surdeanu

Date: 2nd April, 2025

Abstract Overview

This framework targets optimizing large language models (LLMs) for efficiency while preserving performance.

Key Techniques Introduced:

- **Dynamic Pruning:** Selective removal of (likely) redundant components.
- **Layer-Specific Quantization:** Adjusting bit precision based on layer importance.
- **Speculative Copying:** Takes advantage of redundancies to improve LLM inference.

Core Outcomes:

- Reduced memory footprint and computational latency.
- Competitive performance on benchmark tasks.

Implications for real-world applications: mobile computing and edge AI.

Motivation

LLMs like GPT and Llama demonstrate state-of-the-art results but are computationally expensive.
Challenges:

- High computational and memory demands restrict their deployment.
- Many devices cannot handle these models without degradation in performance.

Need for Efficiency:

- Techniques that make LLMs viable for low-resource environments without sacrificing quality.

Real-world scenarios benefiting from efficient LLMs:

- Access to AI for minorities that cannot current afford it.
- Applications in edge AI, where resources are constrained.

Contributions

Three Key Innovations:

- **Dynamic Slicing for Pruning:** Adaptive layer-wise reduction based on importance.
- **Layer-Specific Quantization:** Achieving memory savings by varying bit precision across layers.
- **Speculative Copying:** Uses speculative decoding with string matching for faster inference.

Impact of Contributions:

- Framework enables deployment of competitive LLMs on resource-constrained devices.
- Introduces practical methods for balancing model size and performance.

Broader Significance: Paving the way for accessible AI in low-resource settings.

Roadmap for the Presentation

Structure of Presentation:

- **Part 1:** General Introduction and Problem Statement.
- **Part 3:** Paper 1 – Dynamic Slicing for Efficiency.
- **Part 4:** Paper 2 – Layer-Wise Quantization.
- **Part 2:** Paper 3 – Speculative Copying.
- **Part 5:** Results, Analysis, Future Work, and Conclusion.

What to Expect:

- Detailed methodologies.
- Experimental results.
- Real-world applications and implications.

Dynamic LLM Slicing Based on Layer Redundancy

Authors: Razvan-Gabriel Dumitru, Paul-Ioan Clotan, Vikas Yadav, Darius Peteleaza, Mihai Surdeanu

Published in: EMNLP Findings 2024

Introduction

Problem: Uniform pruning methods treat all layers equally, ignoring their importance.
Challenge: Balancing computational efficiency with maintaining model performance.
Proposed Solution: Dynamic pruning based on layer redundancy (LR) scores.

Observation: Layers contribute unequally to a model's overall performance.
Uniform pruning often removes critical information in some layers.
Key Insight: Adapting pruning based on the redundancy of individual layers improves efficiency.

What are current slicing methods and how is your approach different?

SliceGPT: (SOTA slicing)

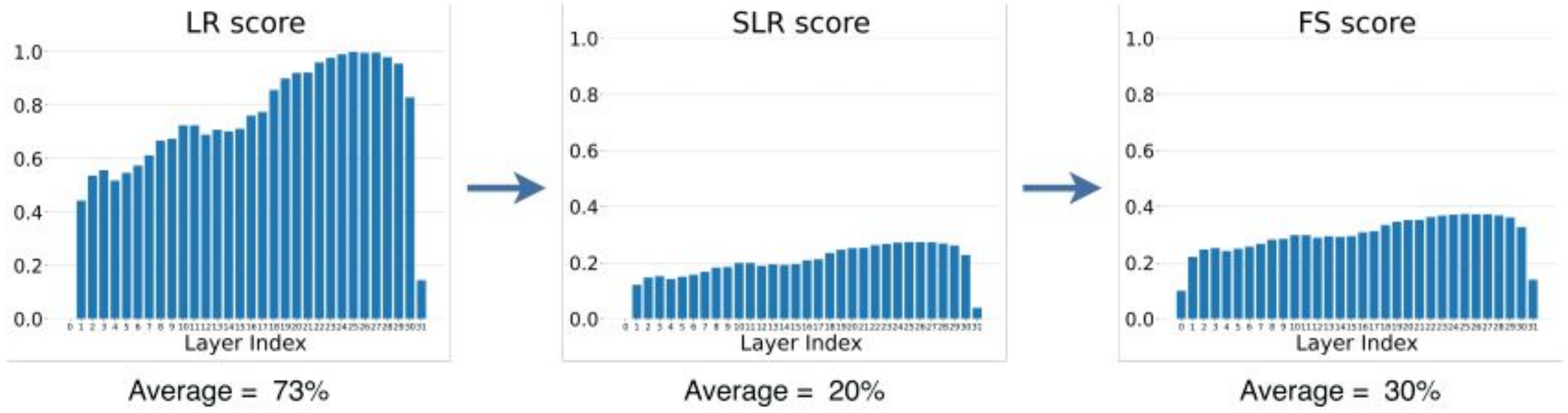
- The weights inside of the layers are sorted based on PCA.
- The less important parts are removed based on a fixed percentage.

Our approach:

- We consider how important layers are as whole, as well as within the weights of each layer.
- Our approach can be integrated with any slicing technique.

Dynamic Slicing: Key Idea

Compute a **Layer Redundancy (LR) Score** for each layer.
Use LR scores to determine the amount of pruning for each layer dynamically.
Combine uniform base pruning with redundancy-based adjustments.



Defining Layer Redundancy

LR Score measures redundancy using **cosine similarity**: $LR(\mathbf{L_i}) = \frac{\mathbf{L_i^I} \cdot \mathbf{L_i^O}}{\|\mathbf{L_i^I}\| \|\mathbf{L_i^O}\|}$

Higher cosine similarity = Higher redundancy.
Evaluate using validation data (pg-19) for accurate redundancy assessment.

How Dynamic Slicing Works

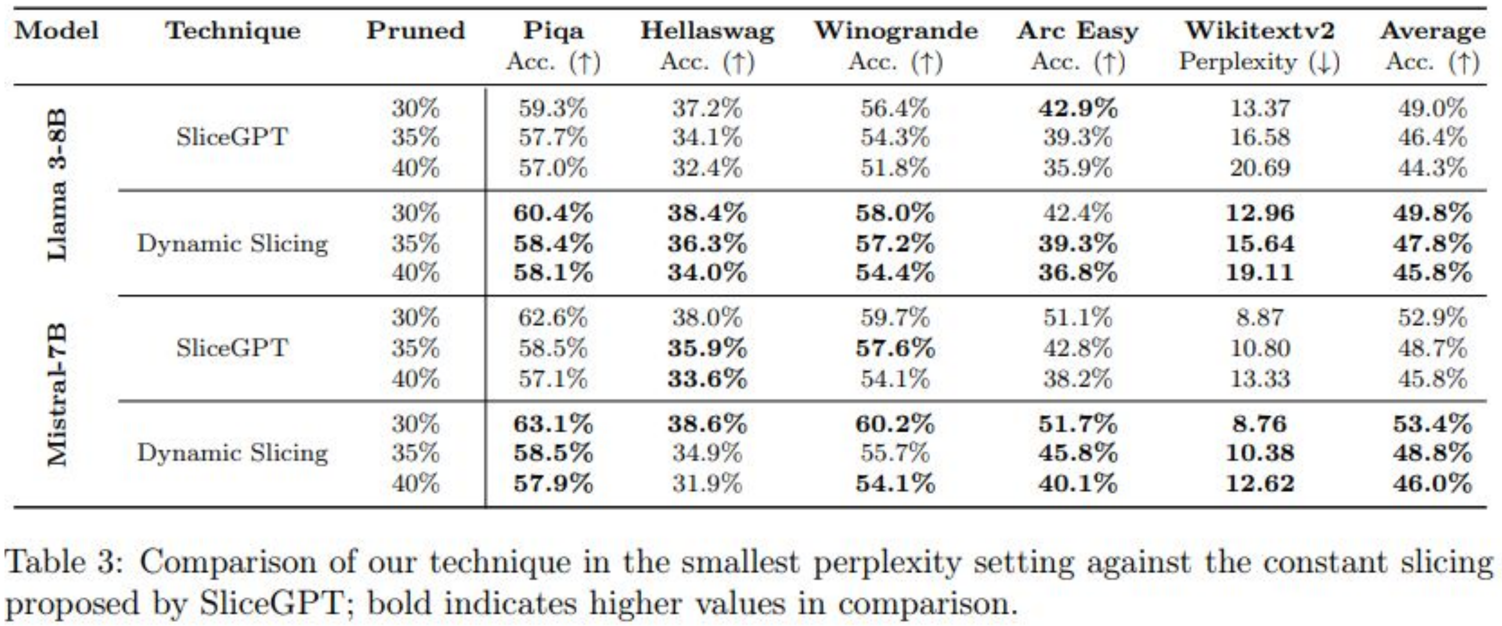
Two Components:

- **Slice Base (SB)**: Minimum pruning applied uniformly across all layers.
- **Slice Percentage (SP):** The average amount that should be cut from the layers.

$$SLR(\mathbf{L_i}) = \mathrm{LR}_i \cdot \frac{S_P - S_B}{\frac{1}{n}\sum_{i=1}^{n} \mathrm{LR}_i}$$

$$FS(\mathbf{L_i}) = SLR(\mathbf{L_i}) + S_B$$

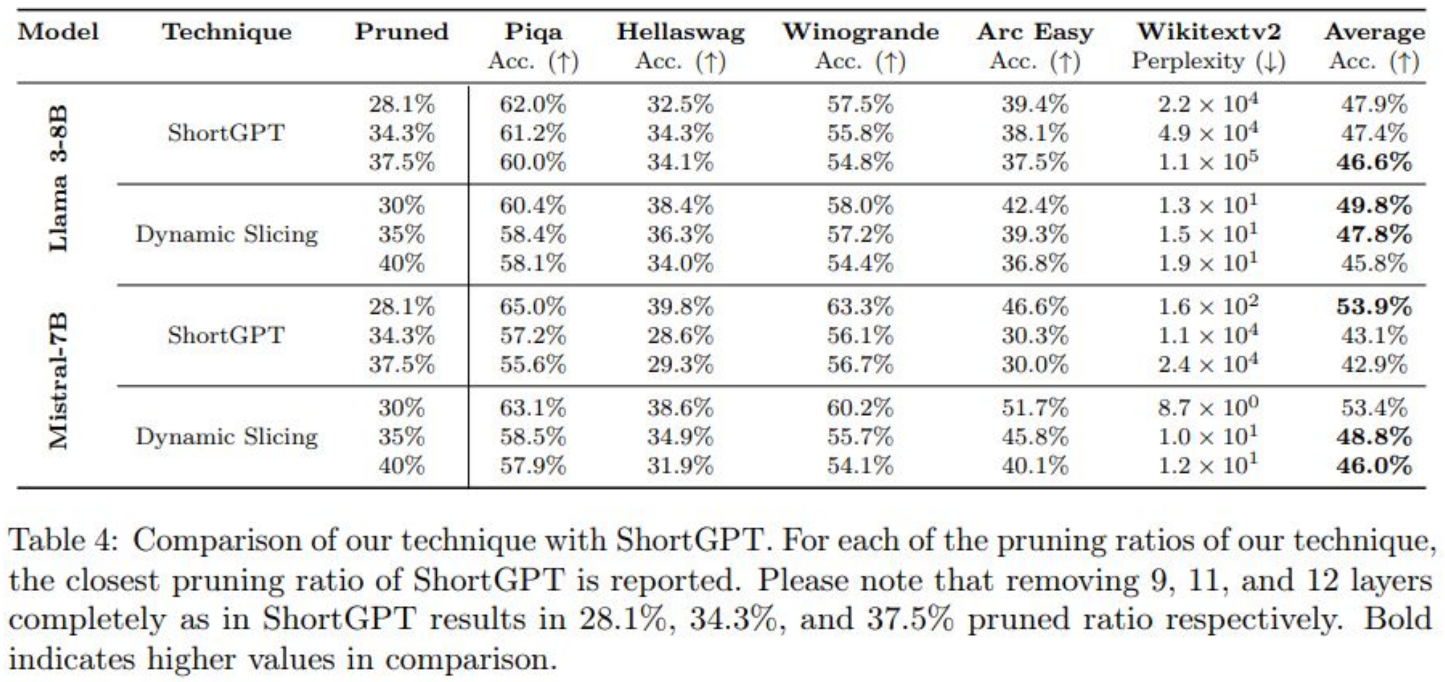
Performance Results - SliceGPT



Model	Technique	Pruned	Piqa Acc. (↑)	Hellaswag Acc. (↑)	Winogrande Acc. (↑)	Arc Easy Acc. (↑)	Wikitextv2 Perplexity (↓)	Average Acc. (↑)
Llama 3-8B	SliceGPT	30%	59.3%	37.2%	56.4%	**42.9%**	13.37	49.0%
		35%	57.7%	34.1%	54.3%	39.3%	16.58	46.4%
		40%	57.0%	32.4%	51.8%	35.9%	20.69	44.3%
	Dynamic Slicing	30%	**60.4%**	**38.4%**	**58.0%**	42.4%	**12.96**	**49.8%**
		35%	**58.4%**	**36.3%**	**57.2%**	**39.3%**	**15.64**	**47.8%**
		40%	**58.1%**	**34.0%**	**54.4%**	**36.8%**	**19.11**	**45.8%**
Mistral-7B	SliceGPT	30%	62.6%	38.0%	59.7%	51.1%	8.87	52.9%
		35%	58.5%	**35.9%**	**57.6%**	42.8%	10.80	48.7%
		40%	57.1%	**33.6%**	54.1%	38.2%	13.33	45.8%
	Dynamic Slicing	30%	**63.1%**	**38.6%**	**60.2%**	**51.7%**	**8.76**	**53.4%**
		35%	**58.5%**	34.9%	55.7%	**45.8%**	**10.38**	**48.8%**
		40%	**57.9%**	31.9%	**54.1%**	**40.1%**	**12.62**	**46.0%**

Table 3: Comparison of our technique in the smallest perplexity setting against the constant slicing proposed by SliceGPT; bold indicates higher values in comparison.

Performance Results - ShortGPT



Model	Technique	Pruned	Piqa Acc. (↑)	Hellaswag Acc. (↑)	Winogrande Acc. (↑)	Arc Easy Acc. (↑)	Wikitextv2 Perplexity (↓)	Average Acc. (↑)
Llama 3-8B	ShortGPT	28.1%	62.0%	32.5%	57.5%	39.4%	2.2×10^4	47.9%
		34.3%	61.2%	34.3%	55.8%	38.1%	4.9×10^4	47.4%
		37.5%	60.0%	34.1%	54.8%	37.5%	1.1×10^5	**46.6%**
	Dynamic Slicing	30%	60.4%	38.4%	58.0%	42.4%	1.3×10^1	**49.8%**
		35%	58.4%	36.3%	57.2%	39.3%	1.5×10^1	**47.8%**
		40%	58.1%	34.0%	54.4%	36.8%	1.9×10^1	45.8%
Mistral-7B	ShortGPT	28.1%	65.0%	39.8%	63.3%	46.6%	1.6×10^2	**53.9%**
		34.3%	57.2%	28.6%	56.1%	30.3%	1.1×10^4	43.1%
		37.5%	55.6%	29.3%	56.7%	30.0%	2.4×10^4	42.9%
	Dynamic Slicing	30%	63.1%	38.6%	60.2%	51.7%	8.7×10^0	53.4%
		35%	58.5%	34.9%	55.7%	45.8%	1.0×10^1	**48.8%**
		40%	57.9%	31.9%	54.1%	40.1%	1.2×10^1	**46.0%**

Table 4: Comparison of our technique with ShortGPT. For each of the pruning ratios of our technique, the closest pruning ratio of ShortGPT is reported. Please note that removing 9, 11, and 12 layers completely as in ShortGPT results in 28.1%, 34.3%, and 37.5% pruned ratio respectively. Bold indicates higher values in comparison.

Ablation Study: Slice Base

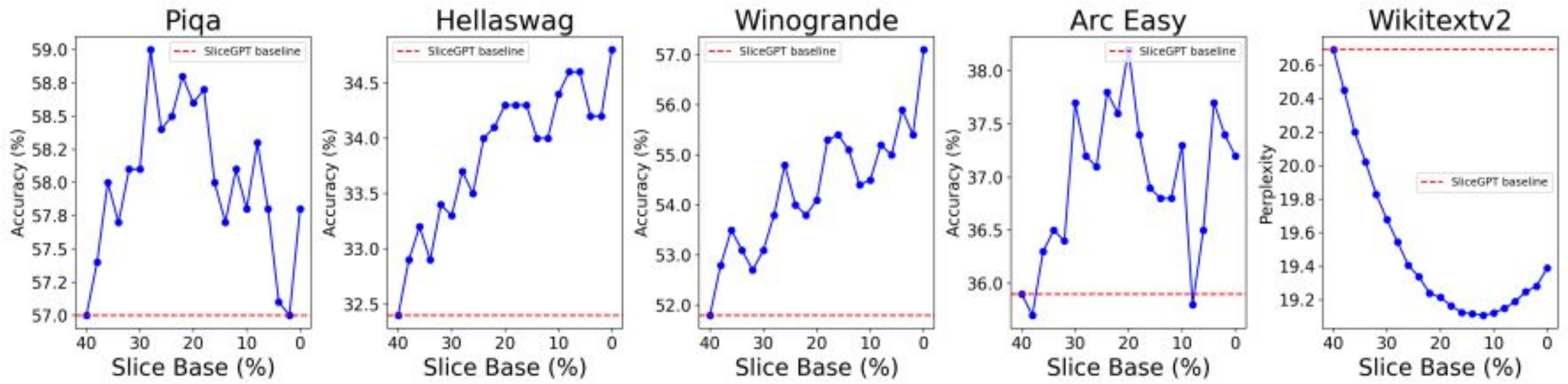


Figure 9: Llama3-8B with 40% of the network sliced on average, the red line is the baseline accuracy achieved by SliceGPT with a constant 40% slice.

Challenges and Future Directions

Challenges:

- Accurate redundancy estimation in extremely large models.
- Evaluation the ideal Slice Base value or finding other approaches.

Future Work:

- Combining dynamic slicing with layer-wise quantization.
- Extending slicing methodology to multimodal models.
- Experimenting with other approaches for layer slicing.

Layer-Wise Quantization Beyond Integer Bit-Levels

Authors: Razvan-Gabriel Dumitru, Vikas Yadav, Rishabh Maheshwary, Paul-Ioan Clotan, Sathwik Tejaswi Madhusudhan, Mihai Surdeanu

In Review: ACL 2025

Introduction

Quantization compresses model size by reducing precision.
Common techniques: Uniform bit-level quantization across layers.
Challenge: Uniform quantization doesn't account for layer importance.
Proposed Solution: Layer-specific quantization based on importance.

What are current quantization methods and how is your approach different?

Uniform Quantization: (Quanto, GPT-Q, SpQR, e.t.c.)

- Applies the same bit precision across all layers.
- Simplifies implementation but can degrade performance in critical layers.
- Some techniques allow for variable bit-levels but they still have the same level for all layers.

Our Method:

- Extends any of these techniques giving them more flexibility.
- Can improve techniques that allow for variable bit-levels such as SpQR.

Why Layer-Wise Quantization?

Not all layers are equally important for model performance.
Uniform quantization can degrade accuracy in critical layers.
Key Insight: Assign higher precision to critical layers and lower precision to less important ones.

Contributions of Layer-Wise Quantization

Introduced variable quantization levels for different layers.
Developed two methods for quantifying layer importance:

- **Layer Input Modification (LIM).**
- **Z-Score Distribution (ZD).**

Significant memory savings with minimal accuracy loss.

Layer Importance Scores

LIM Score:

- Measures how much a layer transforms its input into output on a data set(pg-19).
- Formula: $$\mathrm{LIM}(L_i) = -\frac{\mathrm{Input}(L_i)\cdot\mathrm{Output}(L_i)}{\|\mathrm{Input}(L_i)\|\|\mathrm{Output}(L_i)\|}$$

ZD Score:

- Uses parameter weight distribution to estimate importance.
- Formula: $$\mathrm{ZD}(L_i) = \frac{|L_i^{\text{Z-score}>1}|}{N_i}$$

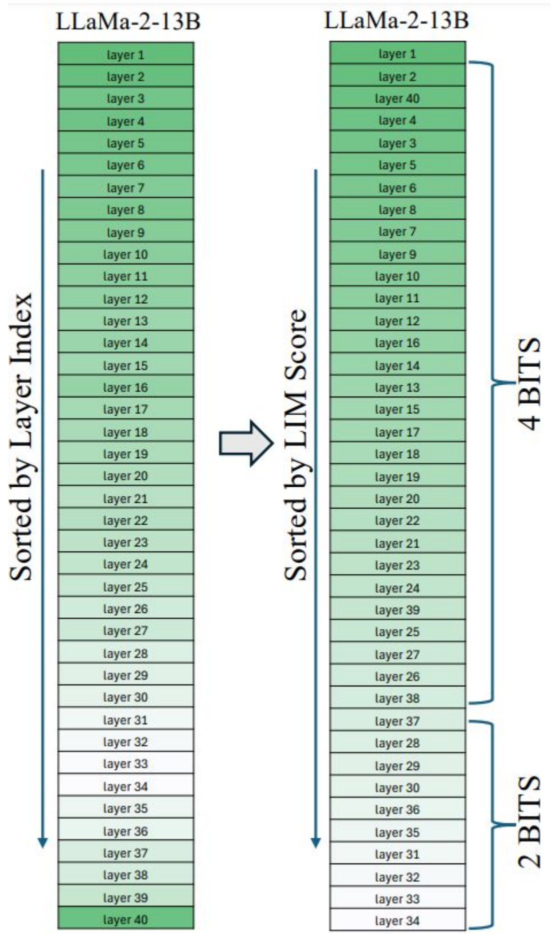
Quantization Methodology

Assign layers to different precision levels based on importance scores.
Example:

- High-importance layers ⟶ 4-bit precision.
- Low-importance layers ⟶ 2-bit precision.

Maintains overall memory budget while prioritizing critical layers.

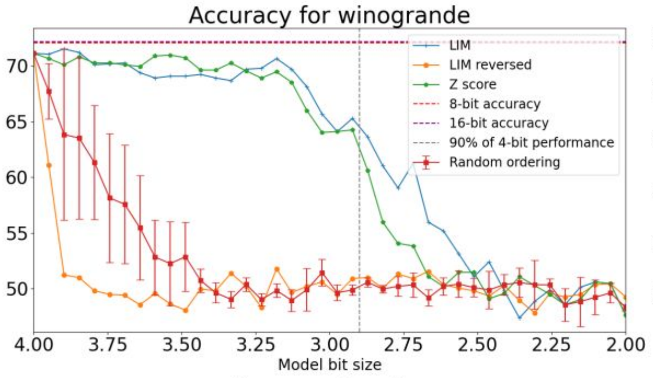
Example for 3.5-bit

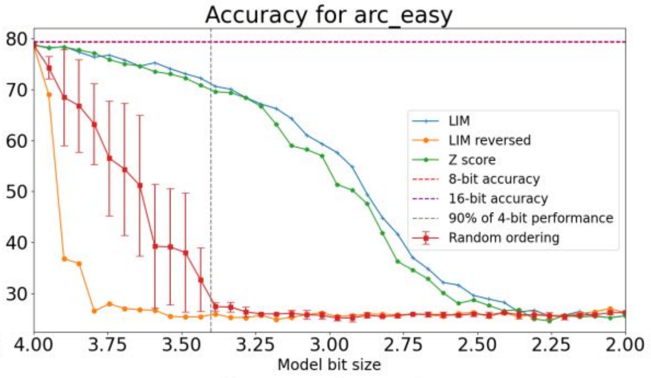


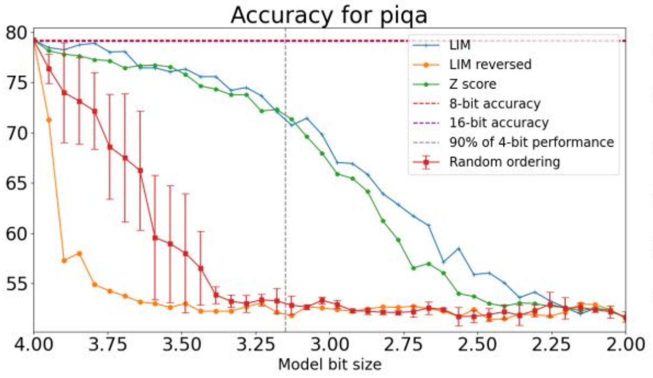
Experimental Setup

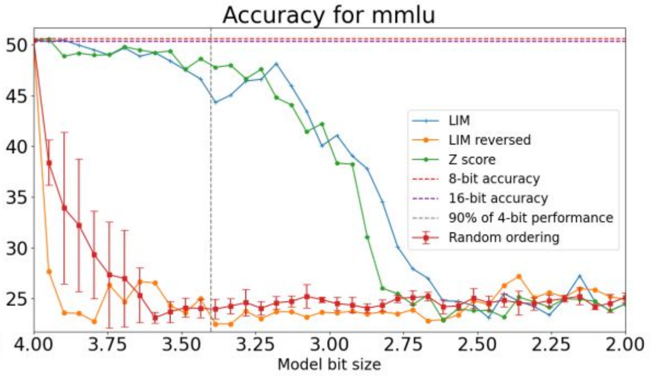
Models: LLaMA-7B, LLaMA-13B, Mistral-7B, and Qwen-7B.
Evaluation Metrics: Accuracy on benchmarks (ARC-Easy, PIQA, HellaSwag, etc.).
Baselines: Uniform quantization, pruning techniques.

LLaMa2-13B Quantization with Quanto

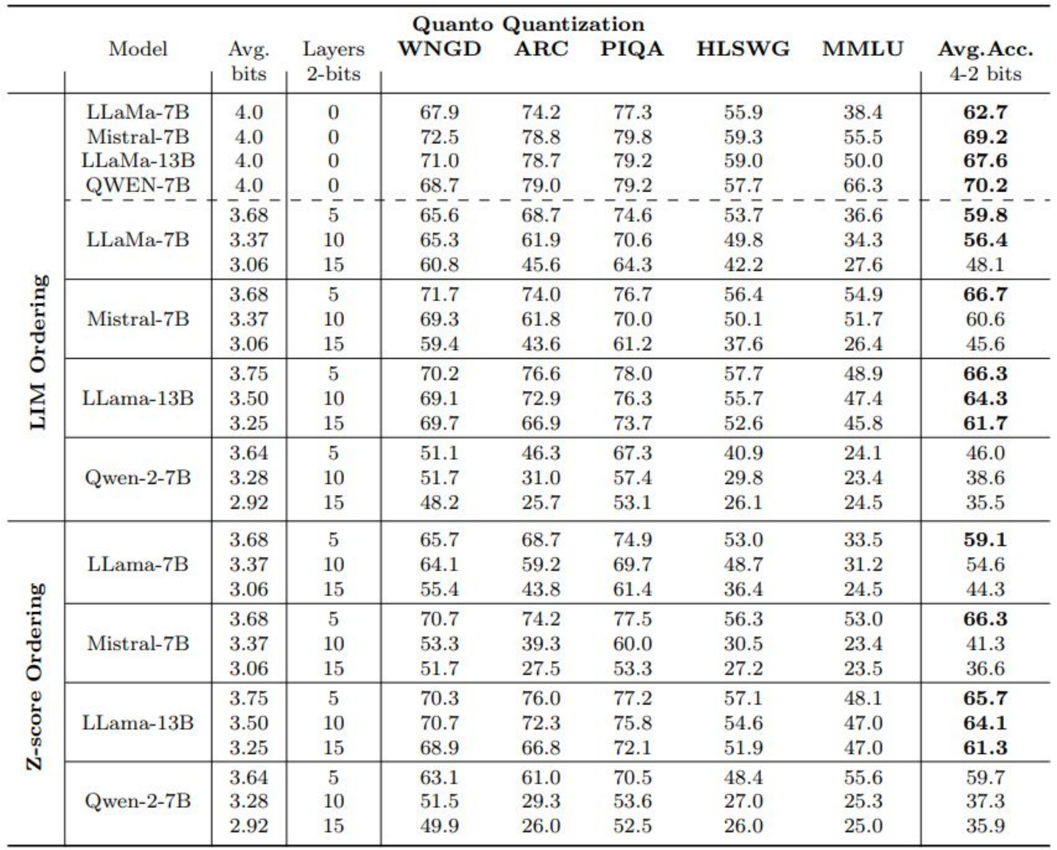








Comparison with Uniform Quantization Using Quanto



	Model	Avg. bits	Layers 2-bits	Quanto Quantization WNGD	ARC	PIQA	HLSWG	MMLU	Avg.Acc. 4-2 bits
LIM Ordering	LLaMa-7B	4.0	0	67.9	74.2	77.3	55.9	38.4	**62.7**
	Mistral-7B	4.0	0	72.5	78.8	79.8	59.3	55.5	**69.2**
	LLaMa-13B	4.0	0	71.0	78.7	79.2	59.0	50.0	**67.6**
	QWEN-7B	4.0	0	68.7	79.0	79.2	57.7	66.3	**70.2**
	LLaMa-7B	3.68	5	65.6	68.7	74.6	53.7	36.6	**59.8**
		3.37	10	65.3	61.9	70.6	49.8	34.3	**56.4**
		3.06	15	60.8	45.6	64.3	42.2	27.6	48.1
	Mistral-7B	3.68	5	71.7	74.0	76.7	56.4	54.9	**66.7**
		3.37	10	69.3	61.8	70.0	50.1	51.7	60.6
		3.06	15	59.4	43.6	61.2	37.6	26.4	45.6
	LLama-13B	3.75	5	70.2	76.6	78.0	57.7	48.9	**66.3**
		3.50	10	69.1	72.9	76.3	55.7	47.4	**64.3**
		3.25	15	69.7	66.9	73.7	52.6	45.8	**61.7**
	Qwen-2-7B	3.64	5	51.1	46.3	67.3	40.9	24.1	46.0
		3.28	10	51.7	31.0	57.4	29.8	23.4	38.6
		2.92	15	48.2	25.7	53.1	26.1	24.5	35.5
Z-score Ordering	LLama-7B	3.68	5	65.7	68.7	74.9	53.0	33.5	**59.1**
		3.37	10	64.1	59.2	69.7	48.7	31.2	54.6
		3.06	15	55.4	43.8	61.4	36.4	24.5	44.3
	Mistral-7B	3.68	5	70.7	74.2	77.5	56.3	53.0	**66.3**
		3.37	10	53.3	39.3	60.0	30.5	23.4	41.3
		3.06	15	51.7	27.5	53.3	27.2	23.5	36.6
	LLama-13B	3.75	5	70.3	76.0	77.2	57.1	48.1	**65.7**
		3.50	10	70.7	72.3	75.8	54.6	47.0	**64.1**
		3.25	15	68.9	66.8	72.1	51.9	47.0	**61.3**
	Qwen-2-7B	3.64	5	63.1	61.0	70.5	48.4	55.6	59.7
		3.28	10	51.5	29.3	53.6	27.0	25.3	37.3
		2.92	15	49.9	26.0	52.5	26.0	25.0	35.9

Table 7: Accuracy on full evaluation datasets of different models quantized with Quanto. All layers start at 4-bits; we then quantize N number of layers in 2-bits where N is mentioned in the "Layers 2-bits" column. We also show results for 8 to 4 bit quantization. Average performances within 90% of the 4-bit model are highlighted in bold.

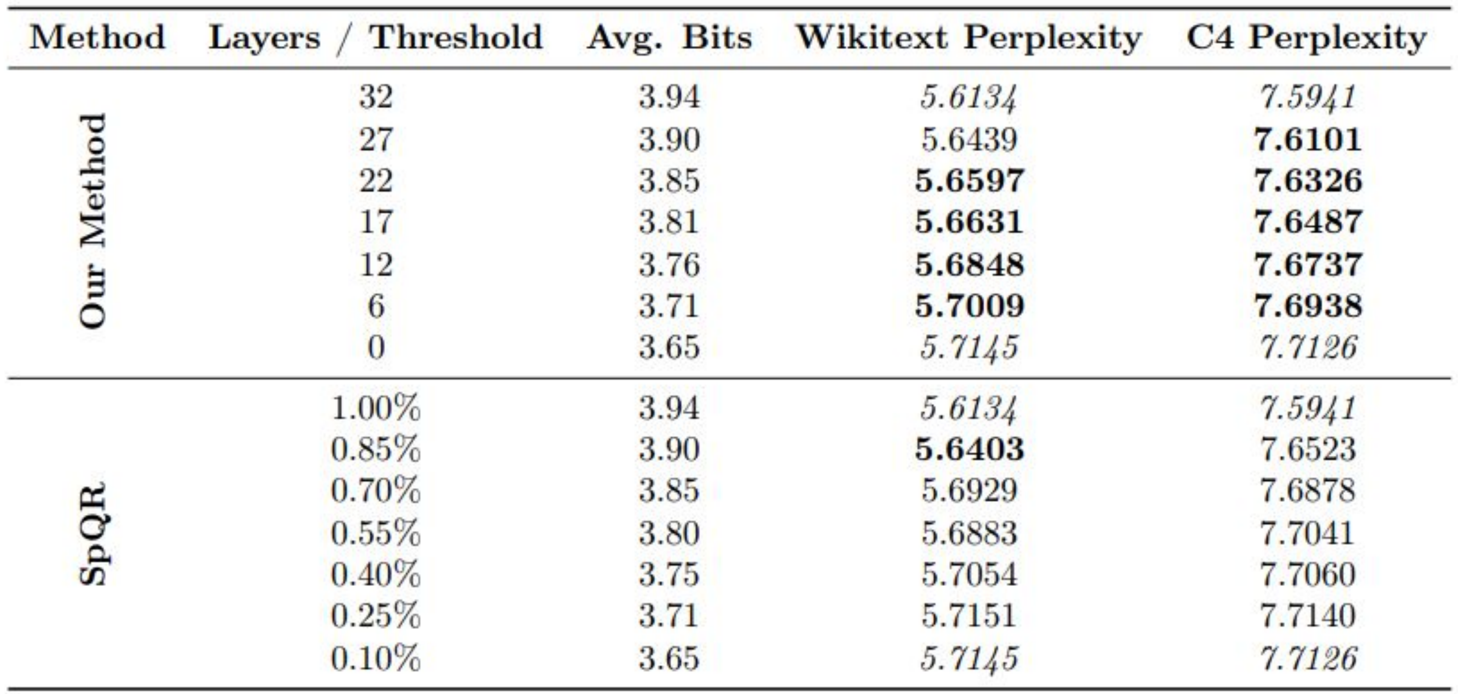
Quantization and Memory Savings

Significant reduction in memory usage:

- 4-bit uniform quantization ⟶ 20GB VRAM.
- Layer-wise quantization ⟶ 16GB VRAM with similar performance.

Real-World Application: Allows LLMs to run on consumer GPUs and it allows to fit in the exact memory.

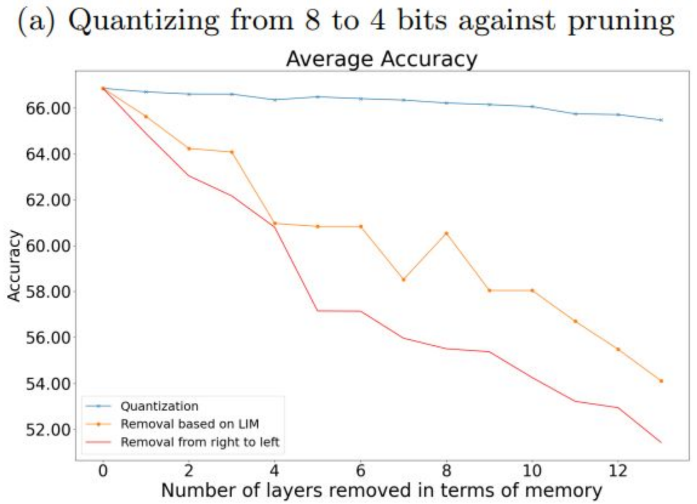
Improving Dynamic Quantization



Method	Layers / Threshold	Avg. Bits	Wikitext Perplexity	C4 Perplexity
Our Method	32	3.94	*5.6134*	*7.5941*
	27	3.90	5.6439	**7.6101**
	22	3.85	**5.6597**	**7.6326**
	17	3.81	**5.6631**	**7.6487**
	12	3.76	**5.6848**	**7.6737**
	6	3.71	**5.7009**	**7.6938**
	0	3.65	*5.7145*	*7.7126*
SpQR	1.00%	3.94	*5.6134*	*7.5941*
	0.85%	3.90	**5.6403**	7.6523
	0.70%	3.85	5.6929	7.6878
	0.55%	3.80	5.6883	7.7041
	0.40%	3.75	5.7054	7.7060
	0.25%	3.71	5.7151	7.7140
	0.10%	3.65	*5.7145*	*7.7126*

Table 10: Comparison of our method and SpQR [15] on LLaMa2-7B on the same avg. bit

Ablation Study: Quantization as Pruning



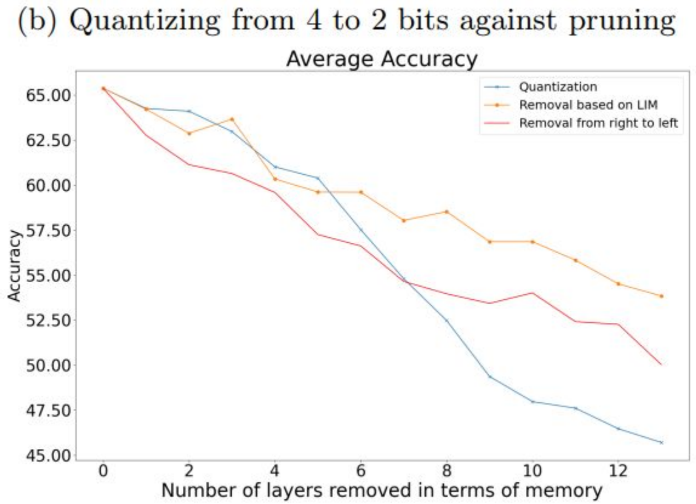


Figure 13: We compare quantization against pruning as a method to reduce the memory requirement of a model (LLaMa-2-7b here). One increment means two layers moved to lower quantization for the blue line (quantization), and one layer removed for the red and orange lines (pruning), thus reducing the same amount of memory. We show the average accuracy over MMLU, Winogrande, PIQA, and Hellaswag.

Challenges and Future Directions

Challenges:

- Complexity in defining accurate importance scores.
- Interaction between layers at different precision levels requires further study.
- Performance degradation at very low precision **(<2 bits)**.

Future Work:

- Different importance scores.
- Using different quantization methods at different bit-levels.

CopySpec: Accelerating LLMs with Speculative Copy-and-Paste Without Compromising Quality

Authors: Razvan-Gabriel Dumitru, Minglai Yang, Vikas Yadav, Mihai Surdeanu

In Review: for ICML 2025

Abstract Overview

- **Goal:** Introduce CopySpec, a technique that identifies repeated token sequences and speculatively copies them without added GPU memory cost.
- **Key Outcomes:**
    - Up to *3.08*× speed-up on multi-turn tasks without loss in output quality.
    - 49% extra speed-up on top of vanilla speculative decoding.
    - Faster inference *as context grows* (rather than getting slower).
- **Implications:** Enhances LLM efficiency for real-time or resource-constrained deployments.

Motivation

- **Growing Demand for LLMs:**
  - High latency and huge compute requirements remain a bottleneck for large-scale or edge deployments.
- **Redundant Sequences:**
  - Many generated outputs repeat large parts of earlier content (e.g., multi-turn chats, iterative code refinement).
- **Need for a Lightweight Solution:**
  - Must avoid extra memory overhead.
  - Should be compatible with existing inference pipelines (no major architectural changes).

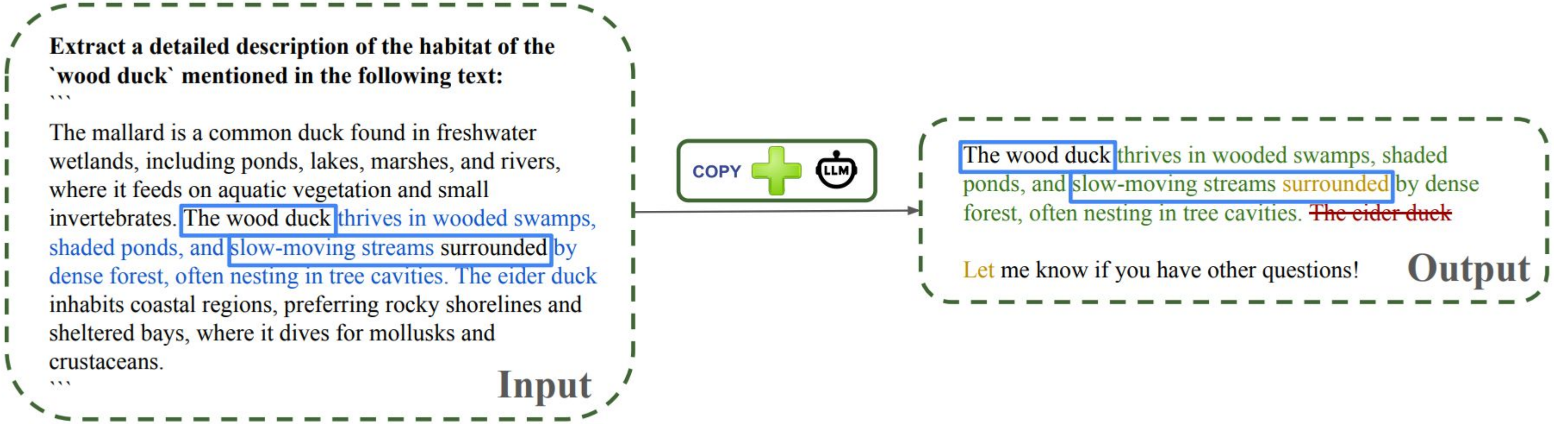
Contributions

1. **CopySpec Mechanism:** Dynamically identifies and reuses repeated sequences during generation.
2. **Seamless Integration:** Merges easily with vanilla speculative decoding; no special hardware or extra memory.
3. **Substantial Speed-ups:** Improves tokens-per-second on multiple benchmarks (CNN/DM, GSM-8K, HumanEval, MT-Bench, and a new MT-Redundant dataset).
4. **Robust Across Models:** Evaluated on five instruction-tuned LLMs (e.g., Qwen2.5, LLaMa3.1 variants).

Roadmap for the Presentation

Speculative Copying with CopySpec

- **Idea in a Nutshell:**
  - During generation, if the last *γ* tokens have already appeared in the context, predict the next tokens by "copying" the matching sequence rather than generating them one by one.



Extract a detailed description of the habitat of the \`wood duck\` mentioned in the following text:

```
The mallard is a common duck found in freshwater wetlands, including ponds, lakes, marshes, and rivers, where it feeds on aquatic vegetation and small invertebrates. The wood duck thrives in wooded swamps, shaded ponds, and slow-moving streams surrounded by dense forest, often nesting in tree cavities. The eider duck inhabits coastal regions, preferring rocky shorelines and sheltered bays, where it dives for mollusks and crustaceans.
```

Input

The wood duck thrives in wooded swamps, shaded ponds, and slow-moving streams surrounded by dense forest, often nesting in tree cavities. ~~The eider duck~~

Let me know if you have other questions!

Output

Introduction

Sequential Generation Bottlenecks:

- Standard decoding is slow, generating tokens one at a time.

Existing Speedup: Speculative Decoding

- Uses a smaller “draft” model to propose multiple tokens, which the larger model verifies.
- However, it doesn’t directly exploit repeated text segments.

Why Copying Mechanisms

Natural Redundancies:

- Users often request slight modifications to previous answers (e.g., “Rewrite your code snippet with a small tweak…”).

Label-Free:

- Unlike fill-in-the-middle methods, CopySpec doesn’t require special masking or tokens; works in standard left-to-right inference.

Broad Applicability:

- Chat, summarization, code generation, multi-turn Q&A, self-correction workflows.

Method Overview

Core Steps:

1. **Search**: Find a match for the last *γ* tokens in the existing context.
2. **Speculate**: Propose the subsequent tokens (e.g., 10 tokens) from that context as the “predicted” block.
3. **Verify**: The main LLM checks the block; accepted tokens become part of the output.
4. **Continue** until the entire output is generated.

Overhead: O(γ) for each newly generated token to check the rolling hash.

- With small γ (2–5), overhead is minimal.

Identifying Tokens to Copy

Rolling Hash Approach:

- Hash all γ-length substrings of the prompt + generated context.
- Example: If γ=3, store and look up 3-token sequences.

Trade-offs:

- **Smaller γ** → more frequent matches, but might produce false positives.
- **Larger γ** → fewer attempts but more confident matches.

Speculating on the Matched Tokens

Accepted Tokens Per Attempt (τ1):

- The number of tokens actually verified and appended in one copy attempt.

Rejected Tokens:

- If the LLM disagrees with the speculated text at any point, it generates a new token that diverges.

No Additional Memory:

- No extra “draft” model is needed if you use CopySpec alone.

Experimental Setup

Models

- Qwen2.5-72B, Qwen2.5-32B, Qwen2.5-7B, LLaMa3.1-70B, and LLaMa3.1-8B.

Hardware

- 4× A100 GPUs with batch size 1.

Hyperparameters

- Typically γ=3, copying block size=10 tokens, temperature=0.

Datasets

MT-Redundant (introduced in the paper)

- Adaptation of MT-Bench with a second turn requesting minor revisions.

CNN/DM (summarization)

GSM-8K (multi-turn math & self-correction)
MT-Bench (multi-turn dialogue)
HumanEval (code generation)

Results

Speed-Ups:

- Up to *2.35*× on CNN/DM.
- Up to *3.08*× in second turn of select MT-Redundant categories (e.g., “Roleplay”).
- *2.66*× on third-turn self-correction in GSM-8K.

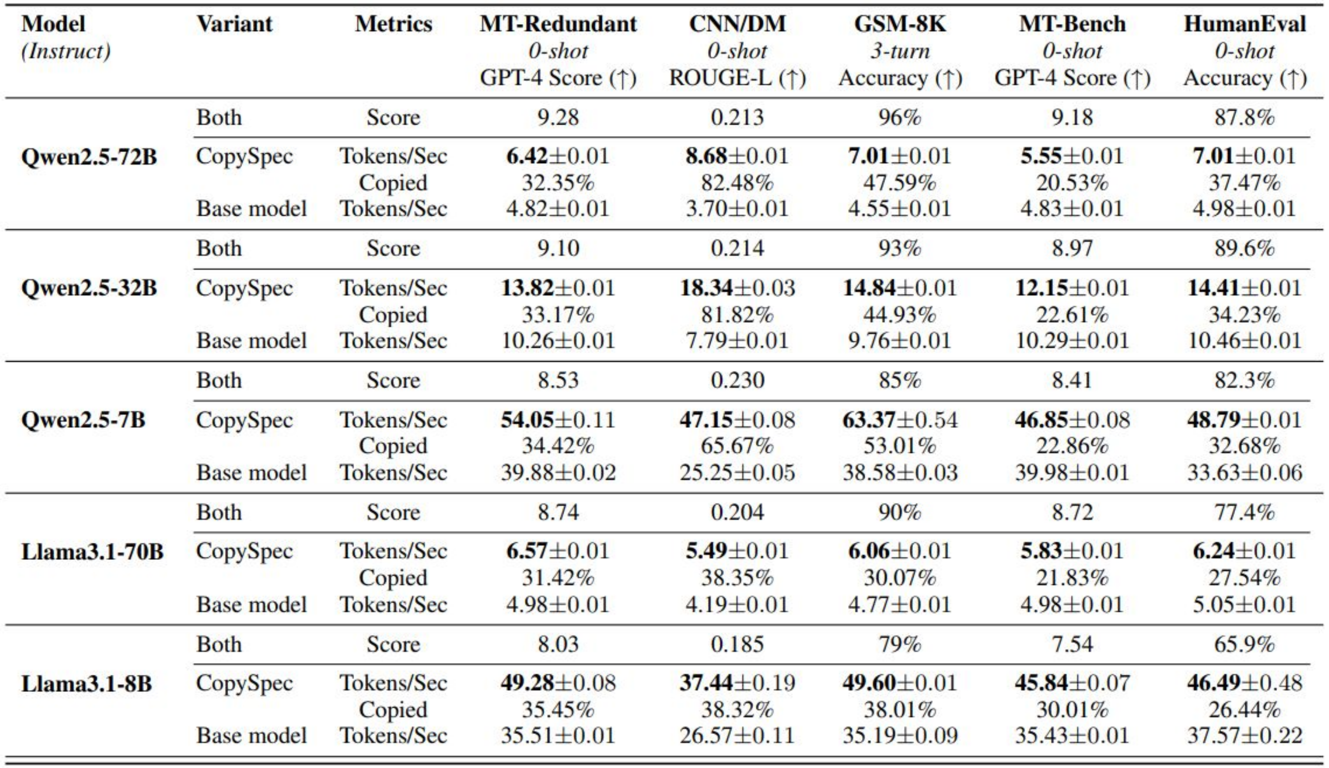
Quality:

- No drop in overall accuracy or GPT-4 evaluation scores.

Orthogonality with Spec Dec:

- 49% additional gain beyond vanilla speculative decoding.

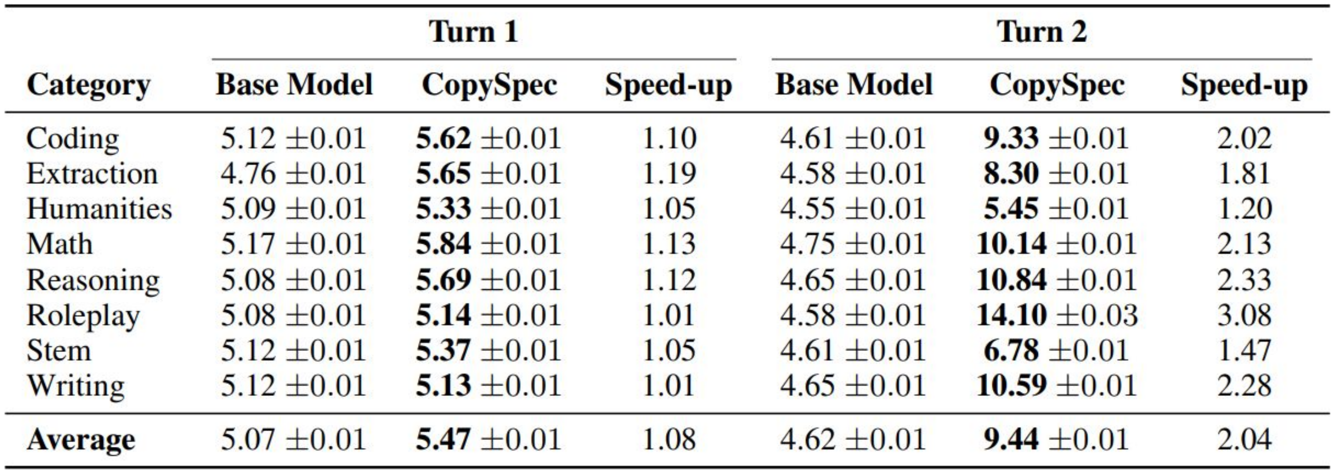
Results



Model	Variant	Metrics	MT-Redundant	CNN/DM	GSM-8K	MT-Bench	HumanEval
(Instruct)			*0-shot*	*0-shot*	*3-turn*	*0-shot*	*0-shot*
			GPT-4 Score (↑)	ROUGE-L (↑)	Accuracy (↑)	GPT-4 Score (↑)	Accuracy (↑)
	Both	Score	9.28	0.213	96%	9.18	87.8%
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
		Copied	32.35%	82.48%	47.59%	20.53%	37.47%
	Base model	Tokens/Sec	4.82±0.01	3.70±0.01	4.55±0.01	4.83±0.01	4.98±0.01
	Both	Score	9.10	0.214	93%	8.97	89.6%
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
		Copied	33.17%	81.82%	44.93%	22.61%	34.23%
	Base model	Tokens/Sec	10.26±0.01	7.79±0.01	9.76±0.01	10.29±0.01	10.46±0.01
	Both	Score	8.53	0.230	85%	8.41	82.3%
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
		Copied	34.42%	65.67%	53.01%	22.86%	32.68%
	Base model	Tokens/Sec	39.88±0.02	25.25±0.05	38.58±0.03	39.98±0.01	33.63±0.06
	Both	Score	8.74	0.204	90%	8.72	77.4%
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
		Copied	31.42%	38.35%	30.07%	21.83%	27.54%
	Base model	Tokens/Sec	4.98±0.01	4.19±0.01	4.77±0.01	4.98±0.01	5.05±0.01
	Both	Score	8.03	0.185	79%	7.54	65.9%
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
		Copied	35.45%	38.32%	38.01%	30.01%	26.44%
	Base model	Tokens/Sec	35.51±0.01	26.57±0.11	35.19±0.09	35.43±0.01	37.57±0.22

Table 1. Performance comparison across five 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, GSM-8K, MT-Bench, and HumanEval. Metrics include model-specific scores (GPT-4, using the 0613 checkpoint: Score, ROUGE-L, Accuracy), token generation rates (tokens/sec), and percentage of tokens copied. 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.

Results



Category	Turn 1			Turn 2		
	Base Model	CopySpec	Speed-up	Base Model	CopySpec	Speed-up
Coding	5.12 ±0.01	**5.62** ±0.01	1.10	4.61 ±0.01	**9.33** ±0.01	2.02
Extraction	4.76 ±0.01	**5.65** ±0.01	1.19	4.58 ±0.01	**8.30** ±0.01	1.81
Humanities	5.09 ±0.01	**5.33** ±0.01	1.05	4.55 ±0.01	**5.45** ±0.01	1.20
Math	5.17 ±0.01	**5.84** ±0.01	1.13	4.75 ±0.01	**10.14** ±0.01	2.13
Reasoning	5.08 ±0.01	**5.69** ±0.01	1.12	4.65 ±0.01	**10.84** ±0.01	2.33
Roleplay	5.08 ±0.01	**5.14** ±0.01	1.01	4.58 ±0.01	**14.10** ±0.03	3.08
Stem	5.12 ±0.01	**5.37** ±0.01	1.05	4.61 ±0.01	**6.78** ±0.01	1.47
Writing	5.12 ±0.01	**5.13** ±0.01	1.01	4.65 ±0.01	**10.59** ±0.01	2.28
Average	5.07 ±0.01	**5.47** ±0.01	1.08	4.62 ±0.01	**9.44** ±0.01	2.04

Table 2. Comparison of model speeds measured in tokens/sec across two turns and eight categories on MT-Redundant using CopySpec and Baseline approaches (Qwen2.5-72B-Chat, $\gamma = 3$). Results demonstrate consistent speed-ups in the second turn due to enhanced token copying capabilities, with variations in performance across categories highlighting task-specific efficiency gains.

Ablation Studies

Varying γ

- γ=2 or 3 typically best. Very small γ=1 → overhead from too many attempts. Larger γ≥10 → fewer attempts but less total benefit.

Copying Block Size

- 10 tokens at a time outperforms large blocks (50–100) due to overhead.

Self-Correction

- Gains become larger as the context grows across multi-turn code refinement.

Varying γ

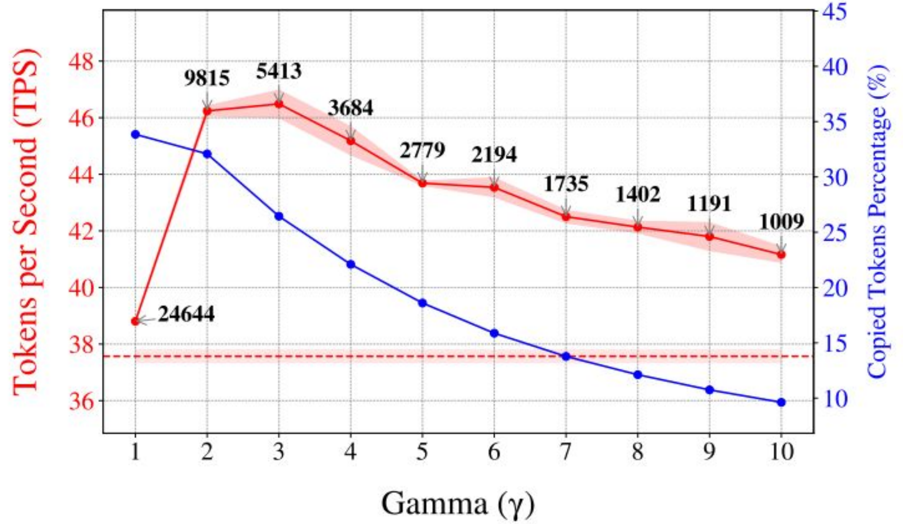


Figure 3. This figure illustrates the relationship between the copying parameter γ and the model's performance on the HumanEval dataset with the LLaMa3.1-8B-Instruct model. The solid red line represents tokens per second (TPS), with shaded areas indicating the standard deviation. The dashed red line shows the baseline TPS without copying. The blue line represents the percentage of tokens successfully copied during generation. Numbers adjacent to data points denote the number of copying attempts.

Varying γ

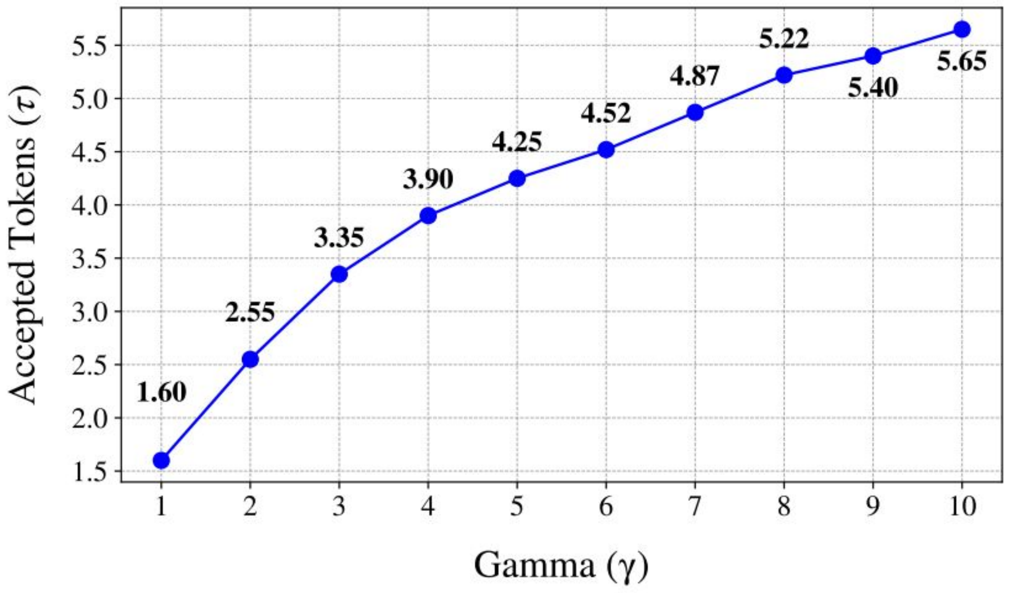
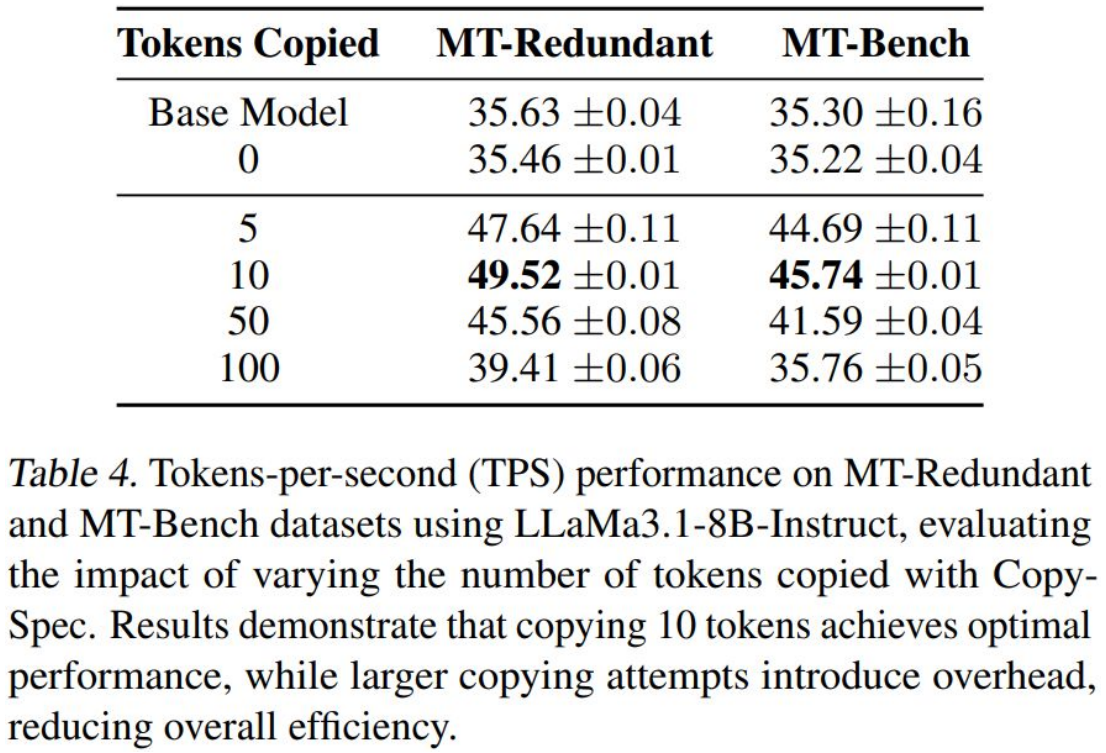


Figure 4. This figure shows the average number of tokens accepted per copying attempt as a function of γ, using the LLaMa3.1-8B model on HumanEval. Each copying attempt speculates on 10 tokens ($|S_{\text{speculate}}| = 10$).

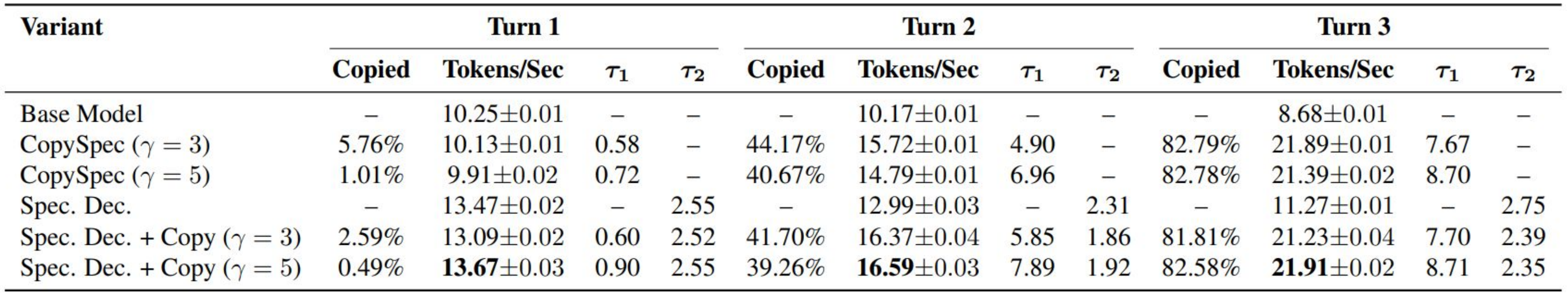
Copying Block Size



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 4. Tokens-per-second (TPS) performance on MT-Redundant and MT-Bench datasets using LLaMa3.1-8B-Instruct, evaluating the impact of varying the number of tokens copied with Copy-Spec. Results demonstrate that copying 10 tokens achieves optimal performance, while larger copying attempts introduce overhead, reducing overall efficiency.

Self-Correction



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 5. 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 19 in Appendix.

Orthogonality with Speculative Decoding

- **Vanilla Spec Dec** → speeds up *short* or *non-repetitive* outputs.
- **CopySpec** → speeds up *long* or *redundant* contexts, especially second/third turns.
- **Combined:**
  - Always at least as fast as whichever method dominates.
  - Gains compound for highly repetitive instructions.

Effect on Reasoning & Self-Refinement

Iterative Code Debugging

- Later turns reuse large chunks from prior code or explanations.

Observation

- The more tokens in the context, the more likely repeated sequences can be copied.
- In practice, tokens-per-second (TPS) *increases* with context length (instead of decreasing) using CopySpec.

Challenges and Future Directions

Automatic γ Tuning: Dynamically adjusting based on current acceptance rate.

Match Selection: Better logic if multiple repeated sequences appear.

Parallel/Tree Decoding: Integrate with tree-based or multi-draft decoding methods.

Responsible Usage: Faster generation can amplify misuse if not monitored carefully.

Conclusion

Key Takeaway:

- CopySpec leverages repeated context to accelerate LLM inference, *particularly* in multi-turn or self-refine tasks.

Benefits:

- Significant speed-ups, no quality trade-off, no extra GPU memory.

Thanks for your attention!

Q/A