Efficient Beam Search for LLMs Using Trie-Based Decoding

Brian J Chan[1†] **MaoXun Huang**[2†] **Jui-Hung Cheng**[1] **Chao-Ting Chen**[1]

Hen-Hsen Huang[3]

[1]Department of Computer Science, National Chengchi University, Taiwan
[2]Department of Computer Science, Cornell University, U.S.
[3]Institute of Information Science, Academia Sinica, Taiwan
110703065@g.nccu.edu.tw mh2653@cornell.edu
{110703007,110703038}@g.nccu.edu.tw hhhuang@iis.sinica.edu.tw
[†]Equal contribution

Abstract

This work presents a novel trie (prefix-tree)-based parallel decoding method that addresses the memory inefficiency of batch-based beam search. By sharing a single KV cache across beams with common prefixes, our approach dramatically reduces memory usage and enables efficient decoding. We evaluated our method across three attention architectures, Multi-Head Attention (Phi-3.5-mini-instruct), Grouped Query Attention (Llama-3.1-8B-Instruct), and Sliding Window Attention (Mistral-Small-24B-Instruct-2501), using CNN/DailyMail for abstractive summarization and HumanEval for code generation. Our experiments demonstrate substantial memory savings (4–8×) and up to 2.4× faster decoding, without compromising generation quality. These results highlight our method's suitability for memory-constrained environments and large-scale deployments.

1 Introduction

Large language models (LLMs) face significant deployment challenges due to their high memory requirements. For example, the 8-billion-parameter Llama 3.1 model, when deployed in float16 precision, requires approximately 15.7GB of GPU memory solely for its model parameters. Processing an 8k token sequence adds another 2.5GB for the key-value (KV) cache. These constraints make efficient memory utilization a critical factor in optimizing LLM performance.

Memory optimization not only reduces hardware requirements but also accelerates inference. Modern GPUs often exhibit faster computation speeds than memory transfer rates, leading to a memory-bound performance bottleneck. Addressing this bottleneck has spurred innovations like Flash Attention (Dao et al., 2022; Dao, 2024), which minimizes memory operations. Efficient memory usage reduces the overhead of transferring data within the GPU, enhancing both speed and scalability.

The decoding process plays a pivotal role in the performance and quality of sequence generation in LLMs. Typical decoding strategies fall into three categories: greedy decoding, top-k sampling, and beam search. Greedy decoding selects the most probable token at each step, offering simplicity but often failing to recover from suboptimal decisions. Top-k sampling introduces diversity by choosing the next token from the k most probable options based on their probabilities. While effective for generating varied outputs, top-k sampling is prone to hallucination (Dziri et al., 2021), limiting its applicability for tasks requiring high factual accuracy, such as programming, math, or retrieval-augmented generation (RAG) (Lewis et al., 2020; Pham and Vo, 2024).

Beam search, on the other hand, maintains multiple candidate sequences (beams) at each time step and ultimately selects the one with the highest overall probability. Unlike greedy search, beam search can "look ahead" to identify sequences that may start with lower-probability tokens but lead to better overall outcomes. By keeping multiple hypotheses, beam search can recover from locally suboptimal decisions, often yielding better results than greedy decoding in certain tasks that require high accuracy, like recommendation (Li et al., 2023) and coding. However, its computational cost and memory demands make it less practical for real-world applications, especially at scale.

Beam search's high memory consumption stems from its handling of KV caches. While beam search explores multiple branches originating from a shared prefix, conventional batch-based implementations allocate independent KV caches for each branch, leading to significant memory redundancy, as overlapping tokens across branches are stored multiple times. Such inefficiencies make memory optimization crucial for scalable and cost-effective deployment.

In this paper, we propose a novel trie (prefix

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 14795–14807
November 4-9, 2025 ©2025 Association for Computational Linguistics

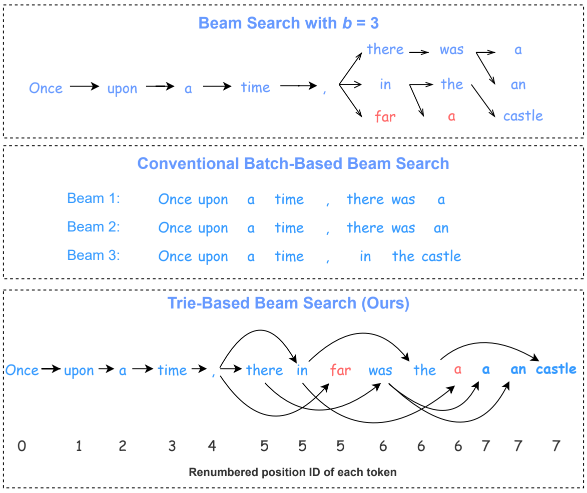
Beam Search with $b = 3$
Once
upon
a
time
,
there
in
far
was
the
a
a
an
castle
Conventional Batch-Based Beam Search
Beam 1: Once upon a time , there was a
Beam 2: Once upon a time , there was an
Beam 3: Once upon a time , in the castle
Trie-Based Beam Search (Ours)
Once upon a time , there in far was the a a an castle
0 1 2 3 4 5 5 5 6 6 6 7 7 7
Renumbered position ID of each token

Figure 1: With beam width $b = 3$, the top panel shows multiple explored hypotheses. The middle illustrates conventional beam search, which stores redundant prefixes (e.g., "Once upon a time,") in separate caches. The bottom shows our trie-based approach, consolidating overlaps into a shared trie to reduce memory while preserving beam width and behavior. Red tokens (e.g., "far", "a") indicate pruned branches, and position IDs are reassigned to match conventional beam search.

tree)-based decoding method that significantly reduces memory usage in beam search by leveraging the hierarchical structure of shared prefixes among branches. Our approach consolidates all branches into a single shared kernel using a trie search strategy. As shown in Figure 1, this method stores only unique tokens corresponding to shared prefixes, reducing memory consumption by eliminating redundant KV cache entries. For instance, in the illustrated example, our approach requires storing only 12 tokens, compared to the 21 tokens required by conventional batch-based methods.

The idea of trie-based decoding introduces two challenges. First, tokens from different beams may inadvertently attend to one another, resulting in corrupted outputs. Second, eliminated tokens may remain in memory, contradicting the goal of efficient memory usage. To address these issues, we adapt the attention mechanism to isolate branch-specific contexts and employ dynamic pruning to remove low-probability branches, ensuring both correctness and memory efficiency. These innovations enable our approach to achieve substantial memory savings while maintaining inference speed, offering a scalable solution for deploying LLMs in resource-constrained settings.

We conduct experiments to evaluate our trie-based decoding approach against greedy decoding and conventional batch-based beam search across three attention variants and two datasets, showing the correctness and robustness of our method. The results demonstrate that our method achieves comparable performance to batch-based beam search with the same beam width, while substantially reducing memory usage, particularly for larger beam widths. Our contributions are as follows:

1. We propose a trie-based decoding method that significantly reduces memory usage during beam search by consolidating KV caches among beams with common prefixes, effectively addressing a critical limitation of batch-based beam search.

2. Under dense attention, our approach is theoretically equivalent to conventional beam search while substantially reducing memory overhead. Empirical results across three transformer architectures, Multi-Head, Grouped Query, and Sliding Window Attention, demonstrate that it preserves output quality, with differences from conventional beam search being statistically insignificant.

3. We release our implementation,[1] offering a scalable and practical decoding alternative. Unlike greedy decoding or top-k sampling, our method retains beam search's robustness with significantly lower computational overhead, enabling efficient deployment of LLMs.

2 Related Work

The evolution of decoding methods for language models in natural language processing (NLP) has been a subject of extensive research, focusing on improving both efficiency and output quality. This section reviews key developments in decoding strategies, including beam search, sampling methods, hybrid approaches, and recent advancements in computational efficiency.

The work of Bahdanau et al. (2016) marked a pivotal moment in NLP, introducing the attention mechanism, which allowed models to dynamically focus on relevant parts of the input sequence during generation. This breakthrough significantly improved translation quality, especially for long sentences. Notably, the study employed beam search as its decoding method—a technique that had already gained traction in statistical machine translation.

[1] https://github.com/brian030128/tridecode

Following this milestone, beam search became the dominant decoding method, as evidenced by its use in prominent works like Vaswani et al. (2017) and Wu et al. (2016). Beam search's ability to maintain multiple hypotheses during decoding often resulted in outputs that were more coherent and grammatically accurate, solidifying its popularity in constrained tasks.

Comparative analyses of decoding strategies have highlighted the trade-offs between beam search and sampling. Ippolito et al. (2019) showed that while sampling methods generate more diverse outputs, they often compromise coherence and factual accuracy. Similarly, Massarelli et al. (2020) emphasized the susceptibility of sampling to hallucinations, contrasting this with beam search's strengths in accuracy and faithfulness, albeit at the cost of diversity. These trade-offs have inspired hybrid approaches, such as combining initial sampling with beam search (Massarelli et al., 2020), to leverage the strengths of both methods.

Computational efficiency has become increasingly critical with the growth of model size and complexity. For example, Vaswani et al. (2017) proposed a high-level algorithm that reduces the beam search space by bounding the length deviation, improving both memory efficiency and speed without sacrificing output quality. While our method operates at a lower level, it can integrate with such techniques to further optimize memory usage.

The issue of prefix overlap has also been studied in speculative decoding. SpecInfer (Miao et al., 2024), Spector and Re (2023), and Medusa (Cai et al., 2024) introduced tree-structured draft-token approaches with tree attention to improve efficiency. Qin et al. (2025) further enhanced SpecInfer with dynamic beam width.

In contrast, speculative decoding focuses on accelerating local sampling, whereas our work reduces the memory footprint of global beam search. These are complementary directions, and our optimized beam search could even be integrated into speculative decoding frameworks to improve draft-token tree generation.

A trie, or prefix tree, is a tree-based data structure designed for efficient storage and retrieval of strings based on their prefixes (Briandais, 1959; Fredkin, 1960). A trie represents common prefixes of strings as shared nodes, enabling compact storage and efficient traversal. Each node corresponds to a character, and the path from the root to any node represents a prefix of the stored string.

In the context of NLP, tries have been employed in tasks such as language modeling, dictionary construction, and decoding. Their ability to compactly represent shared prefixes makes them particularly suitable for beam search, where multiple beams often share a large number of overlapping prefixes. As decoding progresses, most beams converge on a dominant path, leading to substantial redundancy in the KV cache across different beams.

Our work leverages the trie structure to address this redundancy. By organizing beams into a trie, we consolidate overlapping prefixes into a single representation, significantly reducing memory usage. This trie-based approach ensures efficient storage of shared contexts while maintaining the integrity of independent beams during decoding. It highlights the natural fit of trie for optimizing beam search in LLMs, where memory constraints and computational efficiency are critical.

3 Methodology

This section introduces a trie-based decoding approach that addresses inefficiencies by consolidating overlapping prefixes into a shared representation, significantly reducing memory usage while maintaining comparable performance to traditional beam search. We outline the conventional batch-based beam search process, explain the proposed trie-based approach, and detail key innovations such as tree-based attention masking and garbage collection for efficient memory management.

3.1 Batch-Based Beam Search

The high-level concept of batch-based beam search is outlined in Algorithm 1. In transformer-based token generation, each newly generated token attends to the KV cache of previously generated tokens. Due to the nature of matrix operations in attention mechanisms, all tokens within a sequence must share consistent hidden state dimensions. Consequently, in batched beam search, each beam maintains a separate and complete context KV cache to preserve distinct dimensional spaces.

During beam search, most candidate beams are eliminated early on as their cumulative probability scores fall outside the beam width b. As decoding progresses, new branches predominantly grow from the single dominant path. This leads to significant redundancy, with multiple beams sharing overlapping prefixes, as illustrated in Figure 1.

Algorithm 1 Standard Batch-Based Beam Search

```
Require: LLM in batch inference $P(\mathbf{x}_{\text{batch}}|\mathbf{X}_{\text{batch}})$, beam width $b$, prompt $x_1, \ldots, x_t$, and target sequence length $T$
Initialize beam $B_0 \leftarrow \{(x_1, \ldots, x_t)\}$
Initialize empty KV cache kv
for $i = t, \ldots, T-1$ do
    Stack all sequences in $B_i$ into a batch tensor $\mathbf{X}_{\text{batch}}$
    Compute probabilities and update KV cache: $\hat{B}_{i+1}, \text{kv} \leftarrow P(\mathbf{x}_{\text{batch}}|\mathbf{X}_{\text{batch}}, \text{kv})$
    Select top $b$ sequences: $B_{i+1} \leftarrow \texttt{top-b}(\hat{B}_{i+1})$
end for

return Sequence in $B_T$ with the highest cumulative probability
```

Algorithm 2 Trie-Based Beam Search

```
Require: LLM with trie attention $P(x|\mathcal{T}, M)$, where $\mathcal{T}$ is the trie structure, $M$ is the attention mask corresponding to $\mathcal{T}$, and $x$ is the next token to predict; beam width $b$, target sequence length $L$, prompt, garbage collection interval $g$
Initialize a trie $\mathcal{T} \leftarrow$ initialize_trie(prompt)
Initialize attention mask $M$
Serialize the trie to input: input $\leftarrow \texttt{serialize}(\mathcal{T})$
for $i = |\text{input}|, \ldots, L-1$ do
    if $i \bmod g = 0$ then
        garbage_collect()
        $M \leftarrow \texttt{recompute\_mask}(\mathcal{T}, b)$
    end if
    Predict the $b$ best tokens to expand $\mathcal{T}$: V $\leftarrow \text{argsort}_b P(x|\text{input}, M)$
    $\mathcal{T} \leftarrow \texttt{update\_trie}(\mathcal{T}, V)$
    $M \leftarrow \texttt{update\_mask}(\mathcal{T}, M)$
    Serialize updated trie for next iteration: input $\leftarrow \texttt{serialize}(\mathcal{T})$
end for

return Sequence in $\mathcal{T}$ with the highest cumulative probability
```

3.2 Trie-Based Decoding

Our trie-based decoding approach leverages this redundancy by merging all branches with shared prefixes into a single dimension using a prefix tree traversal. This eliminates the need to store duplicated tokens across beams, significantly reducing memory usage.

However, this approach introduces two challenges. First, if the merged tensor is directly processed by the language model, tokens from different branches could attend to each other, corrupting the outputs. Second, eliminated tokens would persist in the tensor, unnecessarily occupying memory and undermining the goal of memory conservation. The following subsections present our solutions to these challenges.

3.3 Tree Attention for Trie-based Decoding

Combining multiple sequence branches into a single dimension improves processing efficiency but risks unwanted cross-branch interactions. For example, in Figure 1, tokens like “castle” and “was,” which belong to different branches, should not influence one another during attention operations. To ensure branch independence, we construct a specialized causal attention mask that mirrors the structure of a trie, as detailed in Algorithm 3.

During the attention mechanism, masks are applied by assigning large negative values to specific attention scores prior to the softmax operation. This ensures that the masked positions receive zero attention weight, thereby eliminating their influence on isolated branches. This standard transformer practice ensures isolation via masking. Since attention weights are computed relatively during the softmax phase, applying a mask beforehand effectively isolates cross-branch tokens without introducing interference.

As illustrated in Figure 2, this mask enforces that tokens attend only to other tokens within their respective branches, maintaining the integrity of each beam during the decoding process. The attention mask is dynamically updated in two key steps. First, after selecting the top b tokens at each decoding step, we update the mask to reflect the relationships between these tokens and their parent branches. Second, following garbage collection (Section 3.5), we update the mask to account for the changes in the KV cache, ensuring consistency with the updated tree structure.

3.4 Maintenance of Positional Integrity

In the trie-based beam search illustration, the renumbered position IDs are designed to simulate the exact behavior of conventional beam search, where the position ID of each token plays a crucial

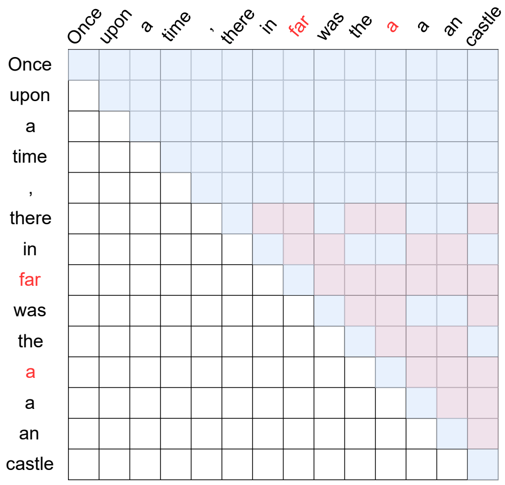

Figure 2: Causal attention masking that mirrors the trie structure. Rows and columns represent tokens in the trie. Blue cells indicate valid self-attention within a branch, while pink cells (masks) block cross-branch connections, preserving branch isolation.

role in contextual understanding within the transformer architecture. Unlike conventional beam search, where each beam maintains independent position IDs, our trie-based approach merges shared prefixes and consolidates the tokens from all beams into a single structure. However, neighboring tokens within the same branch may be separated by tokens from other branches in this shared structure.

To preserve the integrity of positional information, the renumbered position IDs in our approach are assigned to match the positions in the original beam search. This ensures that the contextual dependency between tokens in the same branch remains intact, even when tokens from different branches are interleaved. By aligning the position IDs with those in the conventional beam search, our method achieves equivalent contextual understanding while maintaining memory efficiency. This alignment is critical for ensuring that the model generates outputs consistent with the original beam search behavior, while leveraging the benefits of the trie structure to reduce redundancy.

3.5 Garbage Collection

Garbage collection (GC) consolidates and reclaims unused memory. Because GPU memory operations are expensive, we minimize overhead by deferring token removal and KV cache reorganization. Instead of updating at every decoding step, we accumulate changes and trigger garbage collection only after a predefined threshold is reached. The procedure executes in three stages:

Algorithm 3 Causal Mask Construction for the Trie Structure

```
Require: Trie $\mathcal{T}$, input length $t$, beam width $b$
Initialize attention mask $M \in \mathbb{R}^{b\times(t+|\mathcal{T}|)} \leftarrow -\infty$ {Initialize mask with negative infinity}
$M[:, : t] \leftarrow 0$ {Allow attention to the input sequence}
Initialize temporary nodes: $\mathbf{V} \leftarrow$ leaf_nodes($\mathcal{T}$)
while true do
    reached_root ← true
    for $i = 1$ to $b$ do
        $M[i, \texttt{idx}(V_i) + t] \leftarrow 0$ {Allow attention to current node}
        if parent($V_i$) $\neq \emptyset$ then
            $V_i \leftarrow$ parent($V_i$) {Move up the tree}
            reached_root ← false {Continue traversal}
        end if
    end for
    if reached_root then
        return $M$
    end if
end while
```

1. **Marking**: Traverse the tree bottom-up from leaf nodes to the root, marking all unvisited nodes for removal (CPU).
2. **Pruning**: Eliminate marked nodes from the CPU-side reference structure via a lightweight traversal (CPU).
3. **Compaction**: Compact the KV cache by discarding marked tokens, using `torch.index_select` to retain only unmarked entries (GPU).

At each GC, we reconstruct the decoding sequence from the surviving nodes. This design follows the amortized philosophy of scapegoat trees (Galperin and Rivest, 1993): rather than paying incremental maintenance costs at every step, we periodically rebuild the structure in linear time, achieving predictable long-term efficiency.

4 Experiments

4.1 Experimental Setup

We evaluated our trie-based decoding on three representative transformer models chosen to

Model	Dataset	Beam	Mem/Tok (MB)↓			Tok/Sec↑			Score (mean)		Gain (×)	
			Origin	Trie	Δ±CI	Origin	Trie	Δ±CI	Origin	Trie	Mem.	Speed
Llama 3.1 8B	CNN	1	0.32	*N/A*	*N/A*	5.32	*N/A*	*N/A*	0.20	*N/A*	*N/A*	*N/A*
		3	0.85	0.34	$-0.51^{\dagger}\pm0.00$	4.45	10.96	$+6.51^{\dagger}\pm0.13$	0.21	0.20	2.50×	2.46×
		9	2.45	0.43	$-2.02^{\dagger}\pm0.01$	3.69	10.02	$+6.33^{\dagger}\pm0.18$	0.20	0.20	5.70×	2.72×
	HumanEval	1	0.45	*N/A*	*N/A*	5.61	*N/A*	*N/A*	0.60	*N/A*	*N/A*	*N/A*
		3	1.01	0.58	$-0.43^{\dagger}\pm0.01$	4.98	12.53	$+7.55^{\dagger}\pm0.05$	0.65	0.65	1.74×	2.52×
		9	2.67	0.85	$-1.82^{\dagger}\pm0.02$	4.41	11.81	$+7.40^{\dagger}\pm0.10$	0.66	0.65	3.14×	2.68×
		15	4.35	1.09	$-3.26^{\dagger}\pm0.04$	4.04	11.23	$+7.19^{\dagger}\pm0.15$	0.65	0.65	4.00×	2.78×
Mistral-Small	CNN	1	0.37	*N/A*	*N/A*	7.23	*N/A*	*N/A*	0.17	*N/A*	*N/A*	*N/A*
		3	1.04	0.41	$-0.63^{\dagger}\pm0.01$	5.52	6.56	$+1.04^{\dagger}\pm0.07$	0.17	0.17	2.54×	1.19×
		6	2.04	0.49	$-1.55^{\dagger}\pm0.01$	4.13	5.74	$+1.61^{\dagger}\pm0.10$	0.16	0.16	4.16×	1.39×
	HumanEval	1	0.52	*N/A*	*N/A*	4.32	*N/A*	*N/A*	0.74	*N/A*	*N/A*	*N/A*
		3	1.21	0.68	$-0.53^{\dagger}\pm0.02$	3.71	7.32	$+3.61^{\dagger}\pm0.03$	0.78	0.78	1.78×	1.97×
		6	2.23	0.86	$-1.37^{\dagger}\pm0.02$	3.33	6.59	$+3.26^{\dagger}\pm0.06$	0.77	0.77	2.59×	1.98×
Phi-3.5-mini	CNN	1	1.36	*N/A*	*N/A*	14.13	*N/A*	*N/A*	0.19	*N/A*	*N/A*	*N/A*
		3	4.00	1.39	$-2.61^{\dagger}\pm0.11$	11.51	10.81	$-0.70^{\dagger}\pm0.07$	0.20	0.19	2.88×	0.94×
		9	11.95	1.39	$-10.56^{\dagger}\pm0.19$	4.23	8.88	$+4.65^{\dagger}\pm0.09$	0.19	0.19	8.59×	2.10×
	HumanEval	1	1.11	*N/A*	*N/A*	13.79	*N/A*	*N/A*	0.65	*N/A*	*N/A*	*N/A*
		3	3.23	1.27	$-1.96^{\dagger}\pm0.08$	12.82	13.99	$+1.17^{\dagger}\pm0.10$	0.69	0.69	2.55×	1.09×
		9	9.63	2.21	$-7.42^{\dagger}\pm0.29$	10.89	11.86	$+0.97^{\dagger}\pm0.10$	0.69	0.70	4.36×	1.09×
		15	15.97	3.11	$-12.86^{\dagger}\pm0.46$	9.85	10.96	$+1.11^{\dagger}\pm0.13$	0.70	0.70	5.14×	1.11×

Table 1: Comparison of our trie-based decoding vs. conventional beam search. Means over 1,000 samples; Δ = Trie − Origin. Statistical significance ($p < 0.01$) for efficiency deltas is indicated by †. **Score** reports no significant difference in quality metrics (ROUGE-L for CNN; Accuracy for HumanEval) across all models, datasets, and beam widths. **Memory Efficiency Gain** = Origin / Trie; **Speed Efficiency Gain** = Trie / Origin. For beam size = 1 (greedy decoding), both methods are identical; thus, comparison cells are marked *N/A*.

demonstrate the generalizability of our approach across popular attention mechanisms: Multi-Head Attention (Phi-3.5-mini-instruct; Abdin et al., 2024), Grouped Query Attention (Llama-3.1-8B-Instruct[2]), and Sliding Window Attention (Mistral-Small-24B-Instruct-2501[3]).

Experiments were conducted on two diverse generation tasks: abstractive summarization (CNN/-DailyMail dataset; Nallapati et al., 2016) evaluated using ROUGE-L scores (Lin, 2004), and code generation (HumanEval dataset; Chen et al., 2021) evaluated using binary accuracy.

In addition to generation quality, we evaluated two efficiency metrics. Memory efficiency was defined as memory consumption per processed token:

$$\frac{\text{Peak memory} - \text{Model memory}}{\text{Input length} + \text{Output length}}$$

Here, "Model memory" denotes the fixed GPU memory required to load the model, which is excluded from comparisons. Decoding speed was measured in tokens per second, computed as the ratio of output length to inference time.

All evaluations were conducted in a one-shot setting across beam widths (b). Experiments employed four Tesla V100-SXM2-32GB GPUs for Llama 3.1 and Mistral, and a single GPU for Phi-3.5, demonstrating both single- and multi-GPU compatibility of our algorithm. Statistical significance was assessed using paired t-tests for ROUGE-L, memory efficiency, and decoding speed, and the McNemar test for Accuracy.

4.2 Output Fidelity

Table 1 summarizes the experimental results. Despite substantial efficiency gains, our trie-based decoding is intended to be mathematically equivalent to conventional beam search. Across beam widths, it achieves nearly identical ROUGE-L scores on CNN and comparable accuracies on HumanEval, with no statistically significant differences ($p < 0.01$). Minor variations arise from implementation details, numerical precision, and the lack of batch invariance (He and Lab, 2025).

[2] https://github.com/meta-llama/llama-models/blob/main/models/llama3_1/MODEL_CARD.md

[3] https://huggingface.co/mistralai/Mistral-Small-24B-Instruct-2501

A further limitation stems from sparse attention mechanisms (Child et al., 2019): because they modify the effective dependency structure, the flattened trie may not perfectly replicate the behavior of dense-attention beam search. This structural divergence can introduce additional discrepancies, albeit typically small in practice.

To further validate fidelity, we compared per-token logit distributions from trie-based decoding $T_t \in \mathbb{R}^{B \times V}$ and batch-based beam search $B_t \in \mathbb{R}^{B \times V}$ at each decoding step t. Logits were evaluated up to the first divergence point in the decoding tree. At $b = 3$ across all models and datasets, the average softmax-normalized differences remained below 10^{-5}, effectively at machine precision, confirming equivalence in behavior.

These complementary analyses confirm that trie-based decoding reproduces the outputs of conventional beam search to machine precision in practice. Both the sequence-level results in Table 1 and the per-token logit comparisons show outputs that are indistinguishable across all tested settings. Sparse attention mechanisms may theoretically alter the dependency structure and limit strict equivalence, but such effects did not manifest empirically in our experiments. Overall, trie-based decoding provides a faithful and efficient alternative to beam search, while sparse-attention–aware extensions remain a direction for future work.

4.3 Efficiency Analysis

Memory Efficiency As shown in Table 1, our trie-based decoding substantially reduces memory usage across all models and beam widths. For larger beam widths (e.g., 9 or 15), we observe memory savings of 4–8 times for Phi-3.5-mini and 4–6 times for Llama 3.1 and Mistral-Small. Notably, our method achieves memory usage comparable to greedy decoding (beam width of 1), as illustrated in Figure 3, highlighting its suitability for deployment in memory-constrained environments.

Time Efficiency Table 1 also confirms that our approach consistently improves decoding speed, especially at larger beam widths. For instance, Phi-3.5-mini achieves a speedup of 2.42× at beam width 9, while Mistral attains 1.38× at beam width 6. Although speed was not our primary optimization goal, these improvements are significant, highlighting reduced memory transfer overhead and further enhancing practical applicability.

As summarized in Table 1, multiplicative gains in memory (Memory Gain = Origin / Trie) and speed (Speed Gain = Trie / Origin) consistently increase with wider beams, highlighting the scalability and robustness of trie-based decoding. Although optimal beam width remains task-dependent, our results show that it preserves the output quality of conventional beam search while substantially improving efficiency.

Configuration	Trie	GC	Saved Tokens
Original Beam Search			0.0 ± 0.0
Trie-based w/o GC	✓		360.3 ± 22.1
Trie-based (Ours)	✓	✓	535.8 ± 32.3

Table 2: Results of the ablation study on Phi-3.5-mini with beam width of 3 on HumanEval. “Saved Tokens” denotes the average number of KV cache entries avoided.

5 Discussion

5.1 Ablation Study

We performed an ablation study to evaluate the individual contributions of the two core components in our approach: (1) the **trie-based attention masking**, which removes redundant tokens by consolidating shared prefixes; and (2) the **garbage collection mechanism**, which reclaims memory by eliminating obsolete branches from the KV cache.

Experiments were conducted using the Phi-3.5-mini model on the HumanEval dataset with a beam width of 3. The results are summarized in Table 2. The trie-based attention masking alone yields substantial savings of approximately 360 tokens per run in KV cache storage, confirming its effectiveness in mitigating redundancy from shared prefixes. However, without GC, obsolete branches persist in memory, limiting scalability. Incorporating GC further improves efficiency, reaching an average saving of 536 tokens, and consistently prevents accumulation of unused branches. These results demonstrate that the trie reduces redundancy, while GC sustains efficiency.

5.2 Memory Usage During Decoding

As shown in Figure 3, all methods exhibit a temporary memory spike during the prefilling stage, most noticeable for the Phi-3.5-mini model. The spike originates from a large intermediate output tensor of size (beam width × input length × vocab size) generated during prefilling. Although this tensor is

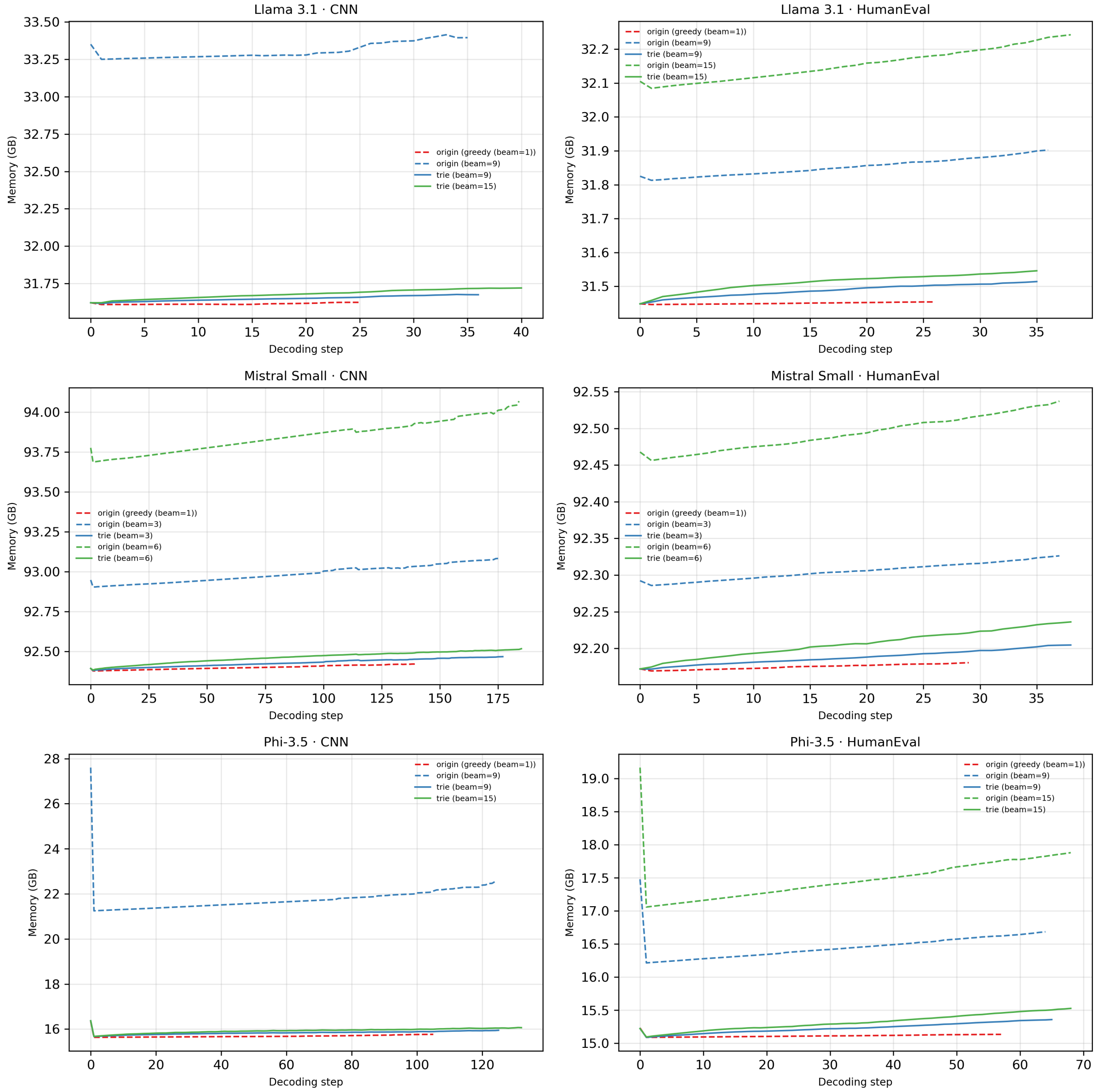

Figure 3: Memory usage during decoding across three models and tasks. Dashed lines show standard beam search; solid lines show trie-based decoding. Each point is the average at a decoding step, truncated when fewer than 80% of samples remain. Our trie-based decoding consistently reduces memory usage, closely matching greedy decoding.

released immediately afterward, the spike is disproportionately pronounced in Phi-3.5-mini due to its smaller model size, accentuating its relative impact on overall memory usage.

5.3 Overhead of Garbage Collection

We conducted an experiment on Llama 3.1 8B using 30 beams to generate 1,000 tokens, as shown in Figure 4. GC consumed less than 20% of the time required for a single decoding pass and was further amortized over 15 decoding steps, resulting in negligible impact on overall decoding time. While the GC overhead does increase with the number of generated tokens, it scales more slowly than the decoding time. This is primarily because attention operations scale quadratically, whereas GC scales linearly. Although CPU and GPU operation costs are not directly comparable, this observation provides insight into the relative insignificance of GC overhead. Given this negligible overhead, we did not perform further optimization, though additional refinements remain possible.

5.4 Trie-based Decoding for Reasoning

We evaluate our trie-based beam search on reasoning-heavy tasks using the MATH500 dataset

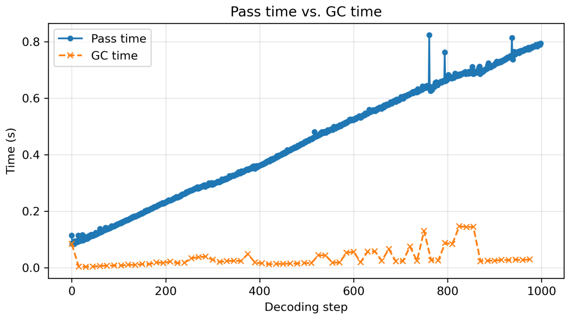

Figure 4: Comparison between GC time and decoding pass time across 1,000 decoding steps with a beam width of 30. GC is triggered every 15 decoding steps in this experiment. The results show that GC overhead remains low and scales more favorably than the decoding pass time, which increases steadily over time.

with chain-of-thought (CoT) prompting ("think step by step") (Hendrycks et al., 2021; Lightman et al., 2023). As baselines, we compare against top-k sampling ($k = 3$), a standard approach for reasoning tasks, and greedy decoding, a special case of top-k sampling ($k = 1$) and of beam search (beam width $b = 1$).

Table 3 shows that trie-based decoding consistently improves accuracy as beam width increases, outperforming both greedy decoding and top-k sampling. For example, accuracy rises from 0.23 (Llama 3.1 8B) and 0.41 (Phi-3.5-mini) under top-k sampling to 0.40 and 0.47 with a beam width of 15. These results confirm that wider beams enhance reasoning consistency and correctness. We also observe that beam search tends to produce longer CoT sequences, consistent with prior findings on the positive correlation between reasoning length and accuracy (Guo et al., 2025).

In terms of efficiency, greedy decoding and top-k sampling use a single beam, resulting in lower memory usage and faster decoding. By design, trie-based decoding maintains multiple beams, increasing memory consumption and reducing throughput. This gap is expected and not a direct comparison. Nevertheless, the overhead remains modest and practically tolerable.

6 Conclusion

This work presents trie-based decoding, a novel beam search method for LLMs that significantly reduces memory usage by consolidating shared prefixes among beams. Our approach demonstrates substantial improvements in memory efficiency, and its effectiveness has been validated across three popular modern transformer architectures, including Multi-Head, Grouped Query, and Sliding Window Attention.

Trie-based decoding is especially beneficial for tasks with large contexts and wide beams, such as code generation. Our approach requires no additional training or specialized hardware, offering a practical, scalable, and cost-effective solution for deploying LLMs in resource-constrained settings.

Results on Llama 3.1 8B

Method	Mem/Tok	Tok/Sec	Acc.	$\bar{\ell}_{out}$
Greedy	0.28	14.03	0.30	411.98
Top-3	0.29	14.08	0.23	383.68
Trie ($b = 3$)	0.38	12.13	0.33	421.80
Trie ($b = 9$)	0.68	9.59	0.38	453.68
Trie ($b = 15$)	1.00	8.22	0.40	440.44

Results on Mistral-Small 24B

Method	Mem/Tok	Tok/Sec	Acc.	$\bar{\ell}_{out}$
Greedy	0.18	8.38	0.62	434.63
Top-3	0.18	8.21	0.35	383.24
Trie ($b = 3$)	0.30	7.90	0.63	434.78
Trie ($b = 6$)	0.44	7.31	0.65	448.66

Results on Phi-3.5-mini 3.8B

Method	Mem/Tok	Tok/Sec	Acc.	$\bar{\ell}_{out}$
Greedy	0.81	12.36	0.43	277.25
Top-3	0.81	12.36	0.41	271.60
Trie ($b = 3$)	1.22	10.93	0.44	290.95
Trie ($b = 9$)	2.37	8.74	0.43	306.74
Trie ($b = 15$)	3.56	7.74	0.47	302.99

Table 3: Comparison of greedy decoding, top-k sampling, and trie-based beam search on MATH500 with CoT prompting. Memory usage is measured in MB, and $\bar{\ell}_{\text{out}}$ denotes the average output length in tokens.

Acknowledgments

We thank the reviewers and area chair for their valuable suggestions that strengthened the final version. This research was partially supported by the National Science and Technology Council (NSTC), Taiwan, under Grant No. 112-2221-E-001-016-MY3; by Academia Sinica under Grant No. 236d-1120205; and by the National Center for High-performance Computing, National Applied Research Laboratories, and NSTC under the "Trustworthy AI Dialog Engine (TAIDE)" project.

Limitations

While our trie-based decoding method significantly improves memory and decoding speed, it also has several limitations:

- **Interaction with Sparse Attention**: Sparse attention may modify the dependency structure, preventing strict equivalence. However, this effect did not manifest empirically in our evaluations. We therefore conclude that trie-based decoding offers a faithful and efficient substitute for conventional beam search in practice, while sparse-attention–aware extensions remain a promising avenue for future work.
- **Garbage Collection Overhead**: Acceleration primarily stems from reduced memory usage. In modern LLMs, the computational bottleneck often lies in memory bandwidth, so when memory usage is high, our implementation tends to be faster than the original. However, garbage collection introduces some overhead. As a result, when the model size is small, the beam width is narrow, and the total number of tokens is low, our method may actually be slightly slower. This can be observed in the case of the small 3.8B Phi-3.5-mini model in Table 1.
- **Task Dependency**: Our approach offers substantial gains primarily when beams share common prefixes, making it most beneficial for tasks where beams frequently converge. Its efficiency gain may diminish for tasks involving highly diverse or divergent outputs.
- **Evaluation Scope**: Our experiments focus on summarization (CNN/DailyMail) and code generation (HumanEval), evaluating three mainstream transformer architectures. While these results support the generalizability of our approach, its performance on other tasks or models, such as multimodal transformers or retrieval-augmented generation, requires further investigation.

Future work should address these limitations, exploring broader task applicability.

Ethical Considerations

This work exclusively utilizes publicly available datasets (CNN/DailyMail and HumanEval), which contain no personally identifiable or sensitive information. Additionally, we disclose that the manuscript underwent minor language refinement and polishing using ChatGPT. The authors retain full responsibility for all content presented.

References

Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, Alon Benhaim, Misha Bilenko, Johan Bjorck, Sébastien Bubeck, Martin Cai, Qin Cai, Vishrav Chaudhary, Dong Chen, Dongdong Chen, and 110 others. 2024. Phi-3 technical report: A highly capable language model locally on your phone. *Preprint*, arXiv:2404.14219.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2016. Neural machine translation by jointly learning to align and translate. *Preprint*, arXiv:1409.0473.

Rene De La Briandais. 1959. File searching using variable length keys. In *IRE-AIEE-ACM Computer Conference*.

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 *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

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.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. *Preprint*, arXiv:1904.10509.

Tri Dao. 2024. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations (ICLR)*.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Nouha Dziri, Andrea Madotto, Osmar Zaïane, and Avishek Joey Bose. 2021. Neural path hunter: Reducing hallucination in dialogue systems via path grounding. In *Proceedings of the 2021 Conference* *on Empirical Methods in Natural Language Processing*, pages 2197–2214, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Edward Fredkin. 1960. Trie memory. *Commun. ACM*, 3(9):490–499.

Igal Galperin and Ronald L Rivest. 1993. Scapegoat trees. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, pages 165–174.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, and 1 others. 2025. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638.

Horace He and Thinking Machines Lab. 2025. Defeating nondeterminism in llm inference. *Thinking Machines Lab: Connectionism*. Https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Daphne Ippolito, Reno Kriz, João Sedoc, Maria Kustikova, and Chris Callison-Burch. 2019. Comparison of diverse decoding methods from conditional language models. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3752–3762, Florence, Italy. Association for Computational Linguistics.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and 1 others. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

Jinming Li, Wentao Zhang, Tian Wang, Guanglei Xiong, Alan Lu, and Gerard Medioni. 2023. Gpt4rec: A generative framework for personalized recommendation and user interests interpretation. *Preprint*, arXiv:2304.03879.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Luca Massarelli, Fabio Petroni, Aleksandra Piktus, Myle Ott, Tim Rocktäschel, Vassilis Plachouras, Fabrizio Silvestri, and Sebastian Riedel. 2020. How decoding strategies affect the verifiability of generated text. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 223–235, Online. Association for Computational Linguistics.

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. ACM.

Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Çağlar Gulçehre, and Bing Xiang. 2016. Abstractive text summarization using sequence-to-sequence RNNs and beyond. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, pages 280–290, Berlin, Germany. Association for Computational Linguistics.

Duy Khoa Pham and Bao Quoc Vo. 2024. Towards reliable medical question answering: Techniques and challenges in mitigating hallucinations in language models. *Preprint*, arXiv:2408.13808.

Zongyue Qin, Zifan He, Neha Prakriya, Jason Cong, and Yizhou Sun. 2025. Dynamic-width speculative beam decoding for llm inference. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(23):25056–25064.

Benjamin Frederick Spector and Christopher Re. 2023. Accelerating LLM inference with staged speculative decoding. In *Workshop on Efficient Systems for Foundation Models @ ICML2023*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, and 12 others. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *Preprint*, arXiv:1609.08144.

A Empirical Analysis of Prefix Overlap

A key advantage of our trie-based beam search over conventional beam search is its ability to consolidate overlapping prefixes across beams, thereby reducing redundant KV cache storage. This naturally raises the question: how frequent is prefix overlap in practice, and are the observed efficiency gains justified?

In principle, certain prompts could yield little to no overlap, limiting the benefit of our method. However, extensive evaluations across multiple models, datasets, and tasks show that such cases are rare in practice.

To quantify overlap, we measured the ratio of memory usage between trie-based decoding (M_T) and conventional beam search (M_B), averaged across all samples ($\frac{M_T}{M_B}$).

Table 4 summarizes results over 21 settings spanning three models from our main experiments (Llama 3.1 8B, Mistral-Small 24B, and Phi-3.5-mini 3.8B). In addition, we report results for a much larger model, Llama 3.1 70B, on the HumanEval dataset with a beam width of 3. Due to computational limitations, this was the only feasible setting we could evaluate for the 70B model.

Across all settings, we observe an average ratio of 0.311 (median 0.274, standard deviation 0.155), consistently reflecting substantial memory savings. These findings indirectly confirm that significant prefix overlap is common in realistic scenarios, and that it underpins the efficiency gains of our approach. Importantly, at the 70B scale, trie-based decoding continued to deliver substantial savings, reinforcing its applicability to very large models.

Model	**Dataset**	b	$\frac{M_T}{M_B} \downarrow$
Llama 3.1 8B	CNN	3	0.385
		9	0.158
	HumanEval	3	0.669
		9	0.274
		15	0.212
Mistral-Small	CNN	3	0.398
		6	0.248
	HumanEval	3	0.053
		6	0.065
Phi-3.5-mini	CNN	3	0.386
		9	0.138
	HumanEval	3	0.512
		9	0.302
		15	0.216
	GSM8K	3	0.494
		9	0.274
		15	0.227
	WMT	3	0.472
		9	0.325
		15	0.270
Llama 3.1 70B	HumanEval	3	0.455

Table 4: Memory usage ratios of trie-based decoding relative to conventional beam search across models, datasets, and beam widths (b).

B Comparing Beam Search and Top-k Sampling

While our primary objective is to improve the efficiency of beam search decoding, it is also informative to compare beam search with alternative strategies to better understand its relative strengths and limitations.

We conducted additional experiments on the HumanEval benchmark using the Phi-3.5-mini model, comparing beam search with top-k sampling ($k = 50$). Top-k sampling achieved an accuracy of 64%, which is lower than greedy decoding (65%) and beam search (70% with $b = 15$). Although top-k sampling matches greedy decoding in speed, its stochastic nature introduces greater variability and increases susceptibility to errors.

As an illustration, consider generating a Python function to compute the maximum depth of nested parentheses for each space-separated substring (e.g., input "`(()()) ((())) ()`" → output `[2, 3, 1]`). As shown in Figure 5, with top-k sampling, the generated code erroneously measures the depth of each individual parenthesis pair, producing `[2, 2, 1, 3, 2, 1, 1]`. In contrast, the code produced via beam search correctly aggregates per substring and returns `[2, 3, 1]`.

```python
from typing import List

def parse_nested_parens(paren_string: str) -> List[int]:
    stack = []
    max_depths = []

    for char in paren_string:
        if char == '(':
            stack.append(len(stack))
        elif char == ')':
            if stack:
                max_depth = stack.pop()
                max_depths.append(max_depth + 1)

    return max_depths
```

```python
from typing import List

def parse_nested_parens(paren_string: str) -> List[int]:
    result = []
    for group in paren_string.split():
        depth = 0
        max_depth = 0
        for char in group:
            if char == '(':
                depth += 1
                max_depth = max(max_depth, depth)
            elif char == ')':
                depth -= 1
        result.append(max_depth)
    return result
```

Figure 5: Code generated with top-k sampling (upper) vs. beam search (lower)