SuffixDecoding: Extreme Speculative Decoding for Emerging AI Applications

Gabriele Oliaro, Zhihao Jia, Daniel Campos, Aurick Qiao

NeurIPS 2025 (Spotlight)

Agenda

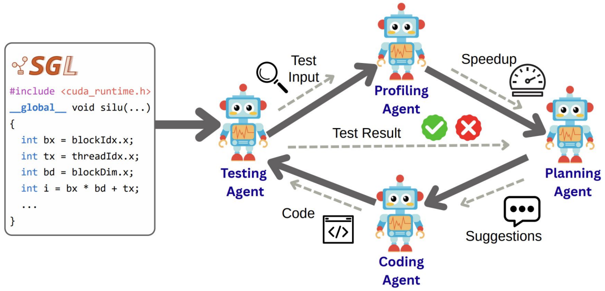
1. **The Problem:** Latency in Modern AI Workloads
2. **Background:** Speculative Decoding 101
3. **Our Solution:** SuffixDecoding
4. **Key Features:** Adaptive, Fast, and Hybrid
5. **Evaluation & Results**
   - End-to-End Speedups
   - Live vLLM Integration
6. **Deeper Dives & Ablations**
7. **Conclusion**

Agenda

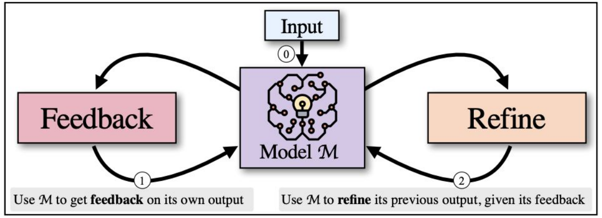
1. **The Problem:** Latency in Modern AI Workloads
2. **Background:** Speculative Decoding 101
3. **Our Solution:** SuffixDecoding
4. **Key Features:** Adaptive, Fast, and Hybrid
5. **Evaluation & Results**
   ○ End-to-End Speedups
   ○ Live vLLM Integration
6. **Deeper Dives & Ablations**
7. **Conclusion**

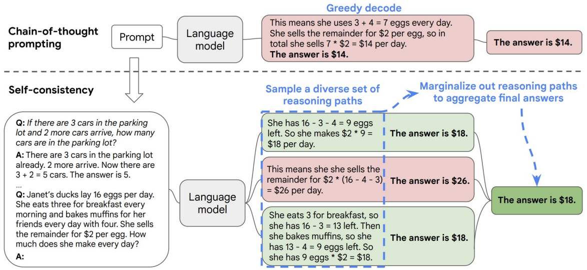
Challenge: Emerging AI Workloads Suffer from High Latency

✅ Inference-Time Scaling techniques can improve the output quality



```
#include <cuda_runtime.h>
__global__ void silu(...)
{
  int bx = blockIdx.x;
  int tx = threadIdx.x;
  int bd = blockDim.x;
  int i = bx * bd + tx;
  ...
}
```


Input
0
Feedback
Model $\mathcal{M}$
Refine
1
2
Use $\mathcal{M}$ to get **feedback** on its own output
Use $\mathcal{M}$ to **refine** its previous output, given its feedback


Chain-of-thought prompting
Prompt
Language model
Greedy decode
Self-consistency
Language model
Sample a diverse set of reasoning paths
Marginalize out reasoning paths to aggregate final answers

This means she uses 3 + 4 = 7 eggs every day. She sells the remainder for $2 per egg, so in total she sells 7 * $2 = $14 per day. **The answer is $14.**

The answer is $14.

Q: *If there are 3 cars in the parking lot and 2 more cars arrive, how many cars are in the parking lot?*
A: There are 3 cars in the parking lot already. 2 more arrive. Now there are 3 + 2 = 5 cars. The answer is 5.
...
Q: Janet's ducks lay 16 eggs per day. She eats three for breakfast every morning and bakes muffins for her friends every day with four. She sells the remainder for $2 per egg. How much does she make every day?
A:

She has 16 - 3 - 4 = 9 eggs left. So she makes $2 * 9 = $18 per day. **The answer is $18.**

This means she she sells the remainder for $2 * (16 - 4 - 3) = $26 per day. **The answer is $26.**

She eats 3 for breakfast, so she has 16 - 3 = 13 left. Then she bakes muffins, so she has 13 - 4 = 9 eggs left. So she has 9 eggs * $2 = $18. **The answer is $18.**

The answer is $18.

Multi-Agent Pipelines

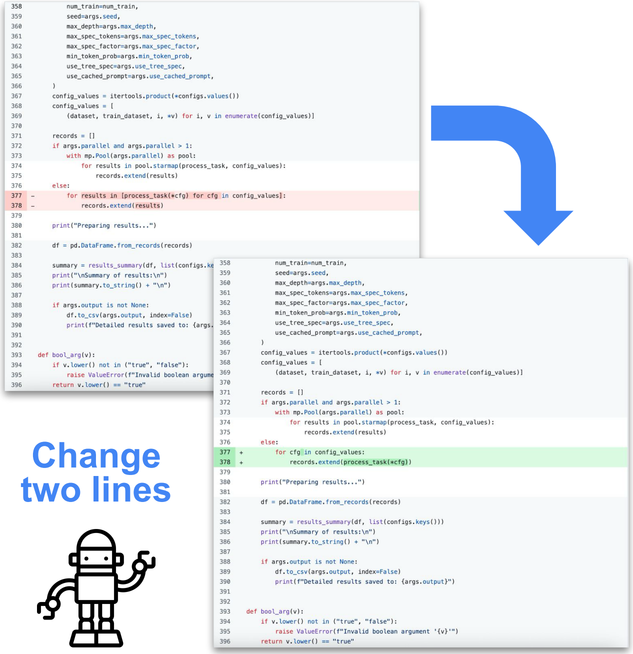
Iterative Refinement

Self-Consistency

⏳ But better accuracy comes **at the cost** of more generated tokens

Opportunity: Token Repetitions in Emerging AI Workloads

- ★ Self-reflection loops
  - ✓ Repeated code/text with minor edits
- ★ Multiple reasoning paths
  - ✓ Similar reasoning patterns
- ★ Multi-agent workflows
  - ✓ Shared context across agents
- ★ Iterative refinement
  - ✓ 100+ lines repeated, 2 modified



```
          num_train=num_train,
          seed=args.seed,
          max_depth=args.max_depth,
          max_spec_tokens=args.max_spec_tokens,
          max_spec_factor=args.max_spec_factor,
          min_token_prob=args.min_token_prob,
          use_tree_spec=args.use_tree_spec,
          use_cached_prompt=args.use_cached_prompt,
      )
      config_values = itertools.product(*configs.values())
      config_values = [
          (dataset, train_dataset, i, *v) for i, v in enumerate(config_values)]

      records = []
      if args.parallel and args.parallel > 1:
          with mp.Pool(args.parallel) as pool:
              for results in pool.starmap(process_task, config_values):
                  records.extend(results)
      else:
-         for results in [process_task(*cfg) for cfg in config_values]:
-             records.extend(results)

      print("Preparing results...")

      df = pd.DataFrame.from_records(records)

      summary = results_summary(df, list(configs.ke
      print("\nSummary of results:\n")
      print(summary.to_string() + "\n")

      if args.output is not None:
          df.to_csv(args.output, index=False)
          print(f"Detailed results saved to: {args.


  def bool_arg(v):
      if v.lower() not in ("true", "false"):
          raise ValueError(f"Invalid boolean argume
      return v.lower() == "true"
```

```
          num_train=num_train,
          seed=args.seed,
          max_depth=args.max_depth,
          max_spec_tokens=args.max_spec_tokens,
          max_spec_factor=args.max_spec_factor,
          min_token_prob=args.min_token_prob,
          use_tree_spec=args.use_tree_spec,
          use_cached_prompt=args.use_cached_prompt,
      )
      config_values = itertools.product(*configs.values())
      config_values = [
          (dataset, train_dataset, i, *v) for i, v in enumerate(config_values)]

      records = []
      if args.parallel and args.parallel > 1:
          with mp.Pool(args.parallel) as pool:
              for results in pool.starmap(process_task, config_values):
                  records.extend(results)
      else:
+         for cfg in config_values:
+             records.extend(process_task(*cfg))

      print("Preparing results...")

      df = pd.DataFrame.from_records(records)

      summary = results_summary(df, list(configs.keys()))
      print("\nSummary of results:\n")
      print(summary.to_string() + "\n")

      if args.output is not None:
          df.to_csv(args.output, index=False)
          print(f"Detailed results saved to: {args.output}")


  def bool_arg(v):
      if v.lower() not in ("true", "false"):
          raise ValueError(f"Invalid boolean argument '{v}'")
      return v.lower() == "true"
```

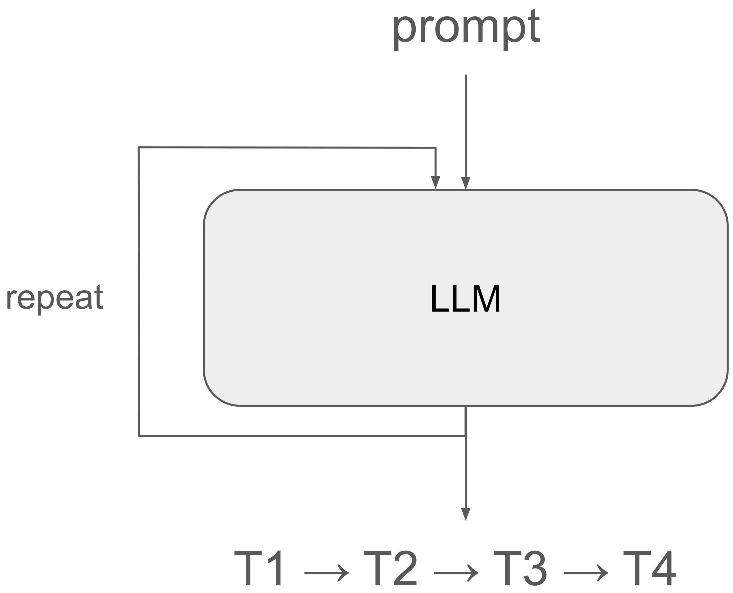
Change two lines

💡 **To leverage these repetitions, we designed SuffixDecoding!**

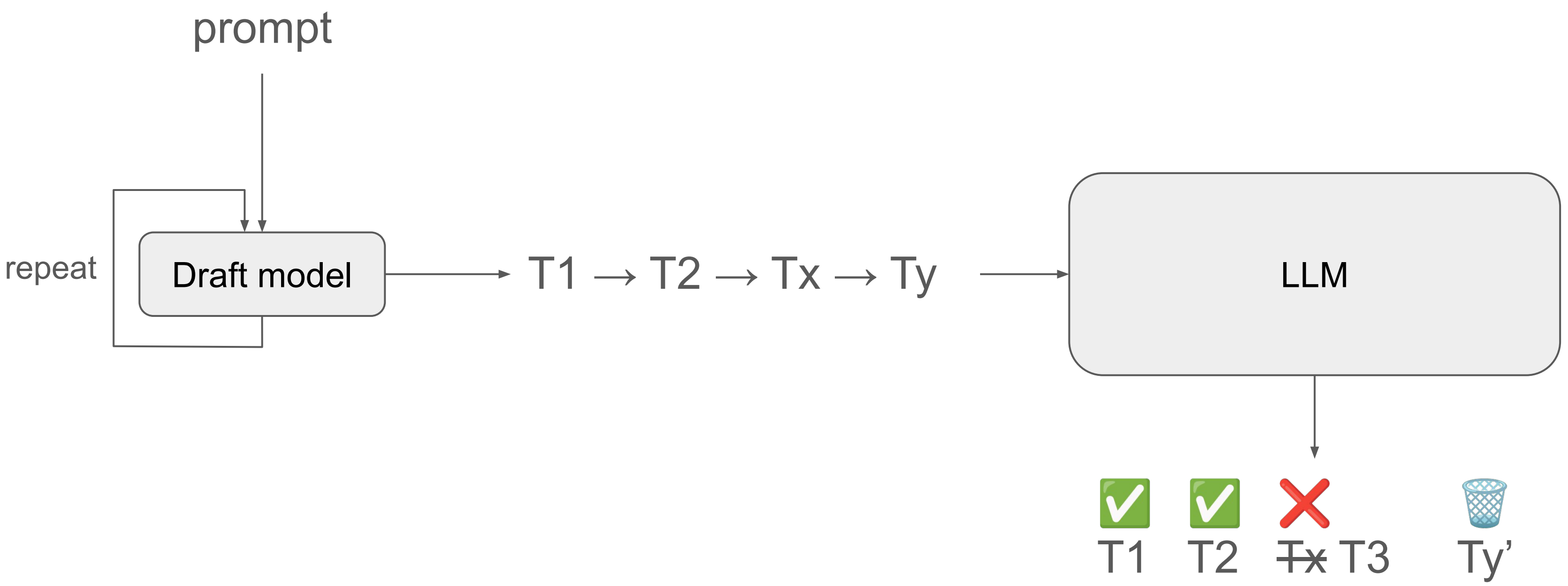
Agenda

1. **The Problem:** Latency in Modern AI Workloads
2. **Background:** Speculative Decoding 101
3. **Our Solution:** SuffixDecoding
4. **Key Features:** Adaptive, Fast, and Hybrid
5. **Evaluation & Results**
    - End-to-End Speedups
    - Live vLLM Integration
6. **Deeper Dives & Ablations**
7. **Conclusion**

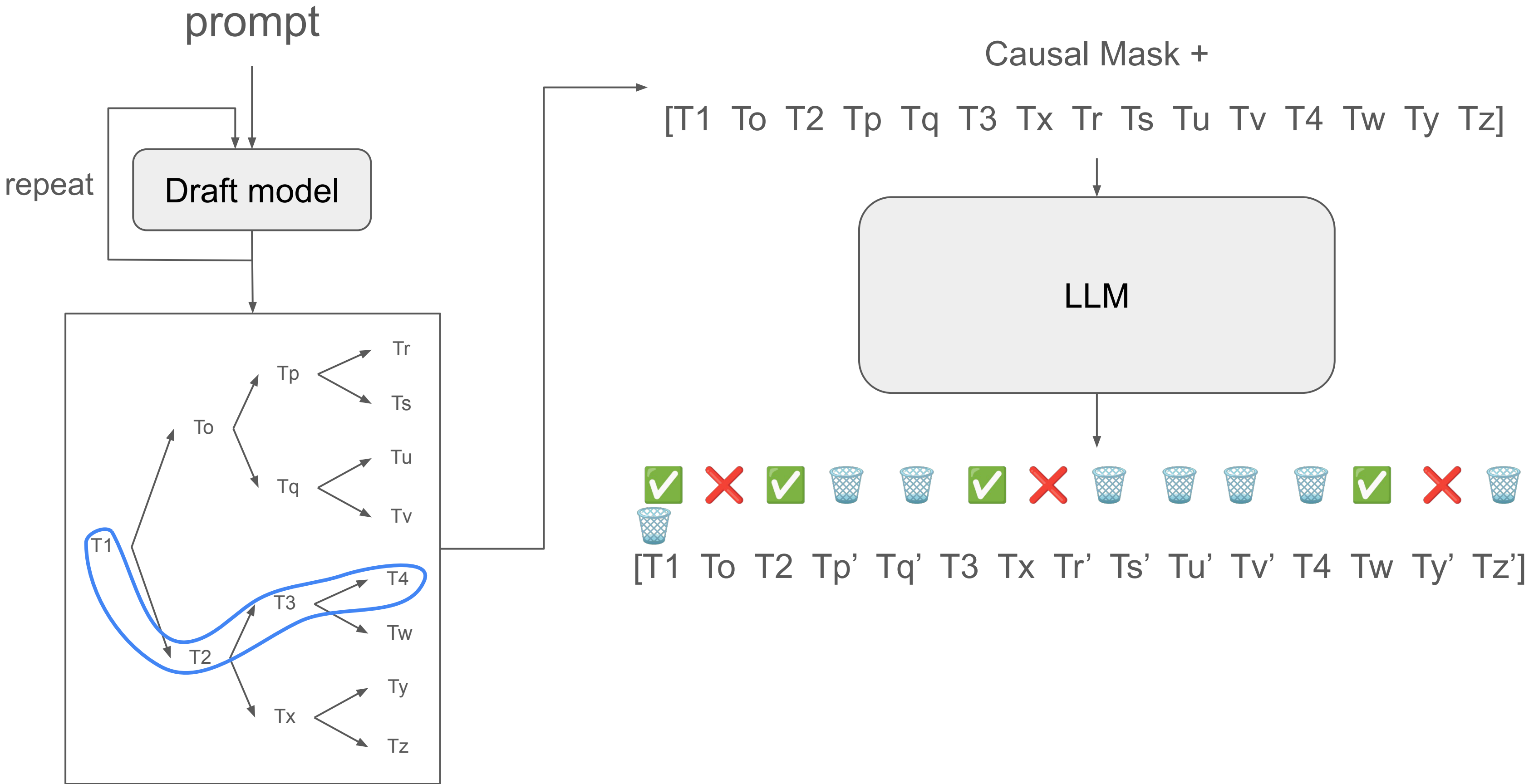
Speculative Decoding 101: Vanilla Decoding



Speculative Decoding 101: Sequence-Based Speculation



Speculative Decoding 101: Tree-Based Speculation



prompt

repeat

Draft model

T1

To

Tp

Tr

Ts

Tq

Tu

Tv

T2

T3

T4

Tw

Tx

Ty

Tz

Causal Mask +

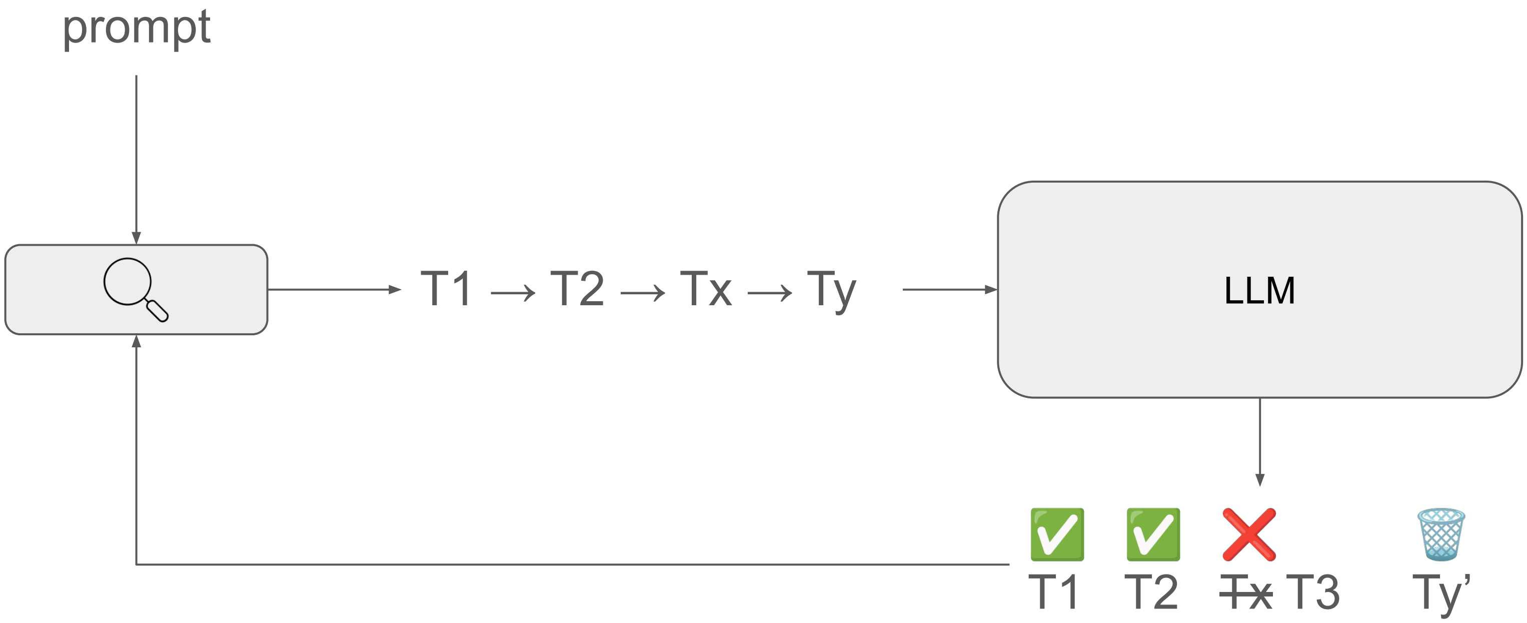
[T1 To T2 Tp Tq T3 Tx Tr Ts Tu Tv T4 Tw Ty Tz]

LLM

✅ ❌ ✅ 🗑️ 🗑️ ✅ ❌ 🗑️ 🗑️ 🗑️ 🗑️ ✅ ❌ 🗑️
🗑️

[T1 To T2 Tp’ Tq’ T3 Tx Tr’ Ts’ Tu’ Tv’ T4 Tw Ty’ Tz’]

Speculative Decoding 101: Model-Free Speculation



prompt

T1 → T2 → Tx → Ty

LLM

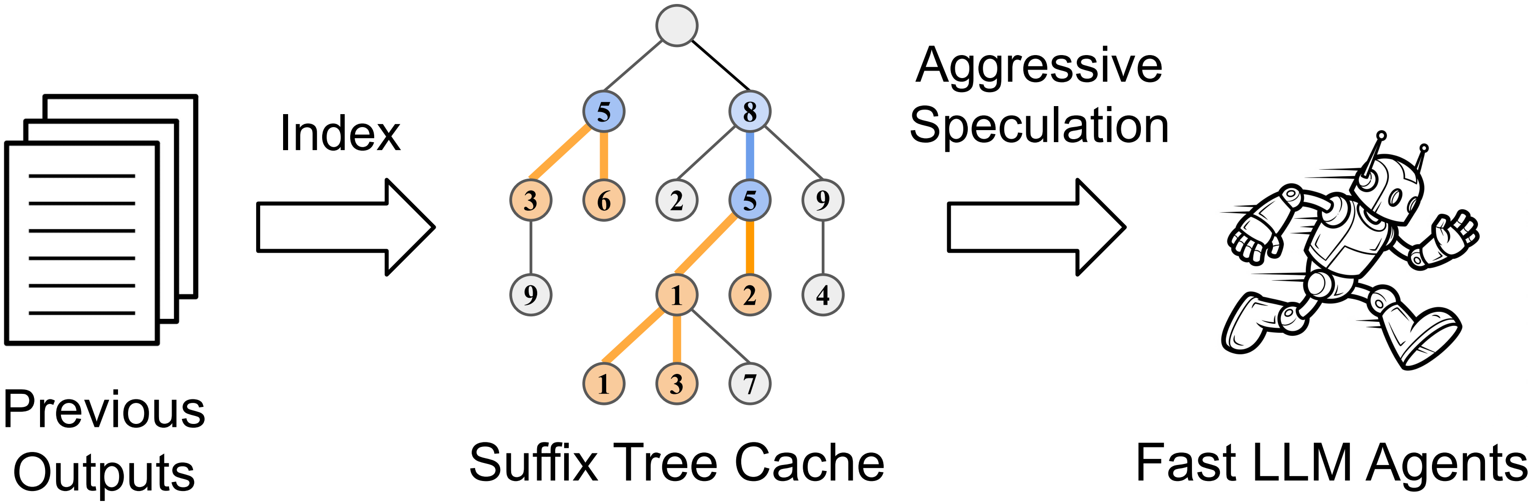
✅ ✅ ❌ 🗑️

T1 T2 ~~Tx~~ T3 Ty’

Agenda

1. **The Problem:** Latency in Modern AI Workloads
2. **Background:** Speculative Decoding 101
3. **Our Solution:** SuffixDecoding
4. **Key Features:** Adaptive, Fast, and Hybrid
5. **Evaluation & Results**
   - End-to-End Speedups
   - Live vLLM Integration
6. **Deeper Dives & Ablations**
7. **Conclusion**

🚀 SuffixDecoding: Up to **5.3x** end-to-end speedup



Previous Outputs

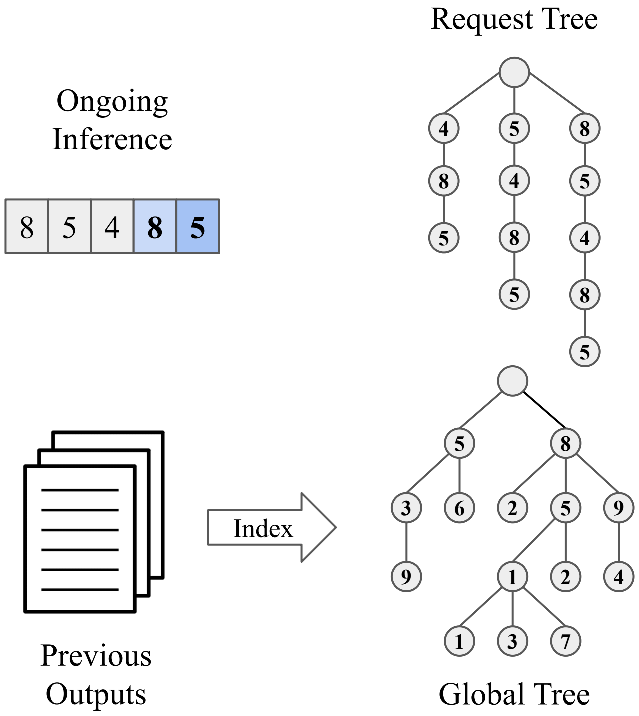
Index

Suffix Tree Cache

Aggressive Speculation

Fast LLM Agents

SuffixDecoding: A Tale of Two Trees 🌲🌳



Request Tree

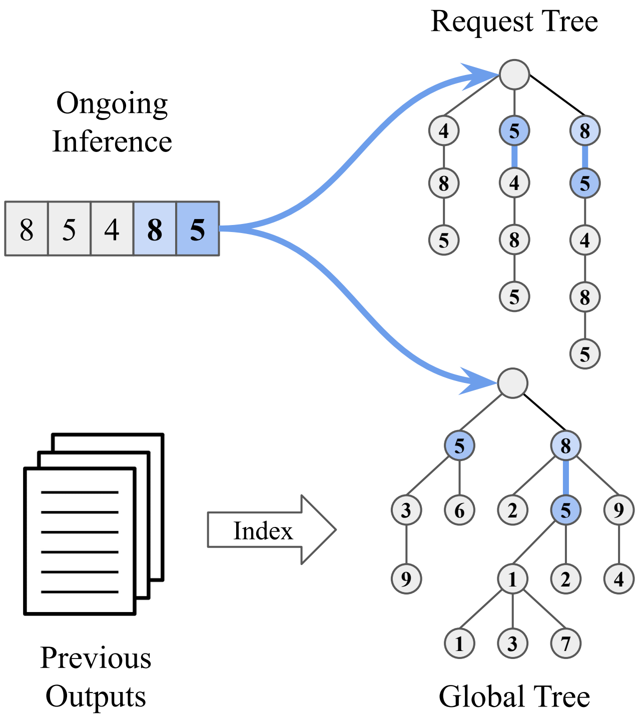
Ongoing Inference

Previous Outputs

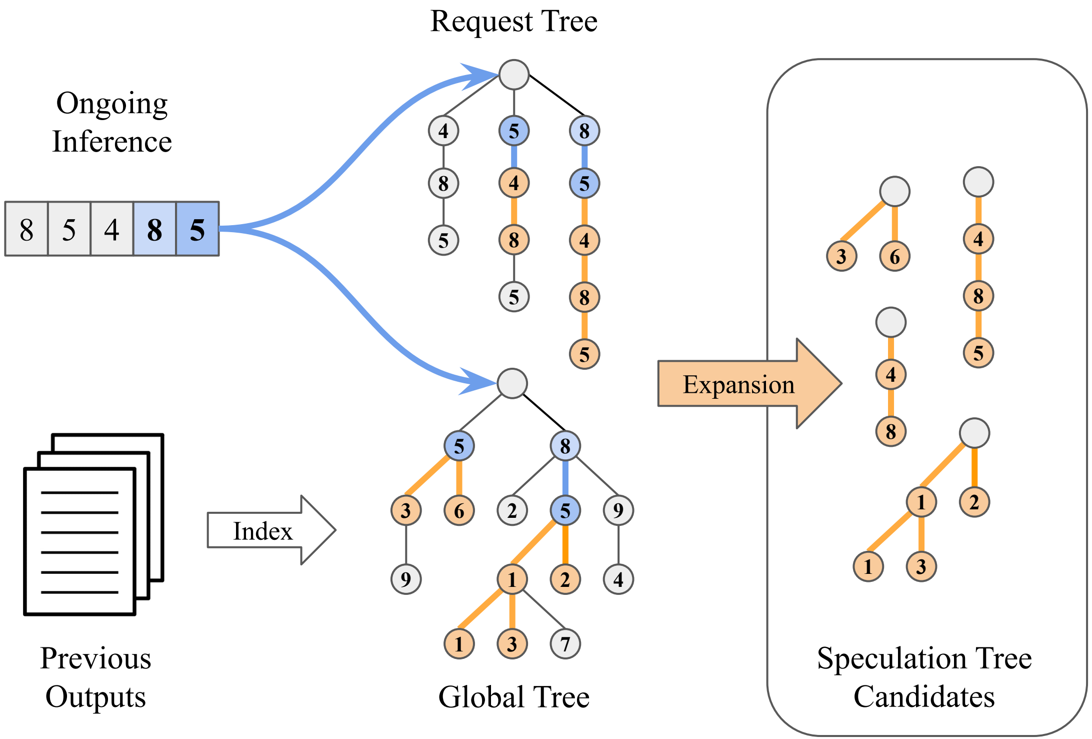
Index

Global Tree

SuffixDecoding: A Tale of Two Trees 🌲🌳



SuffixDecoding: A Tale of Two Trees 🌲🌳



Request Tree

Ongoing Inference

Previous Outputs

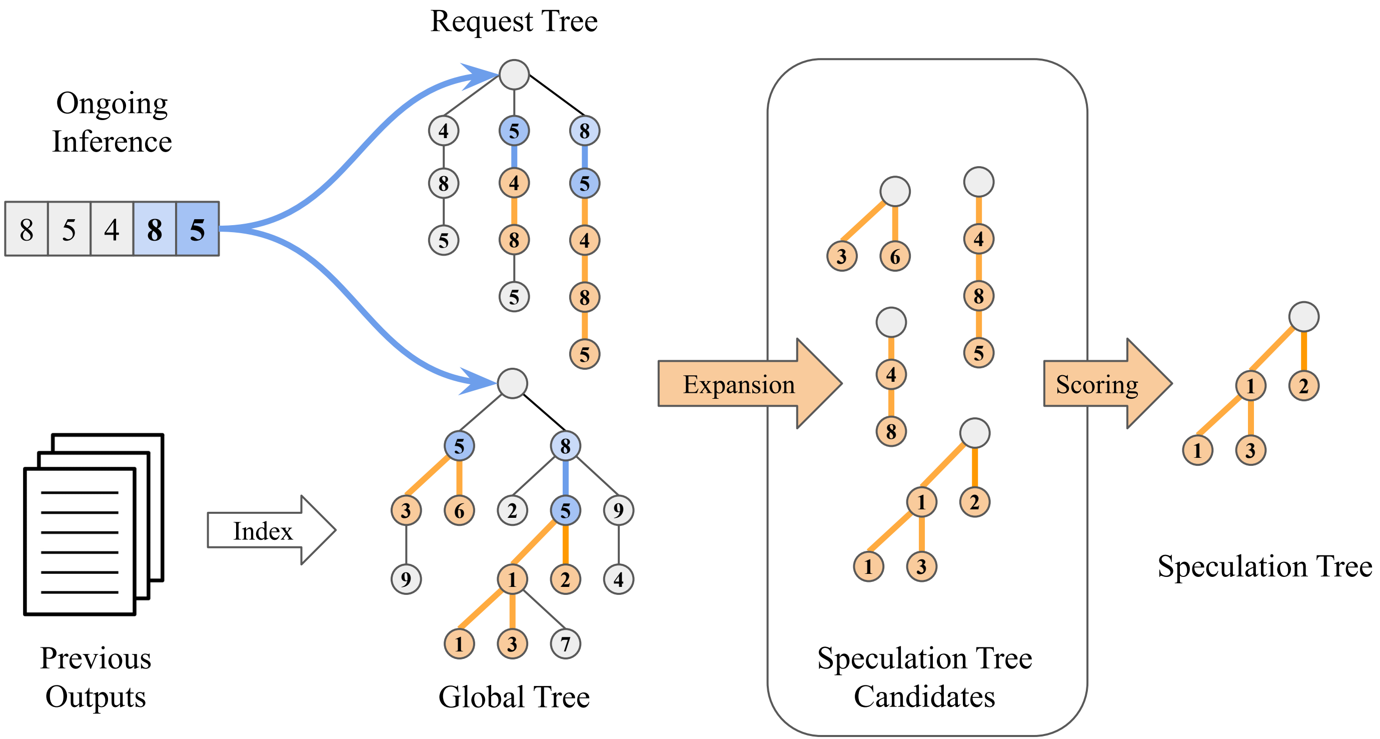
Index

Global Tree

Expansion

Speculation Tree Candidates

SuffixDecoding: A Tale of Two Trees 🌲🌳



Ongoing Inference

Request Tree

Previous Outputs

Index

Global Tree

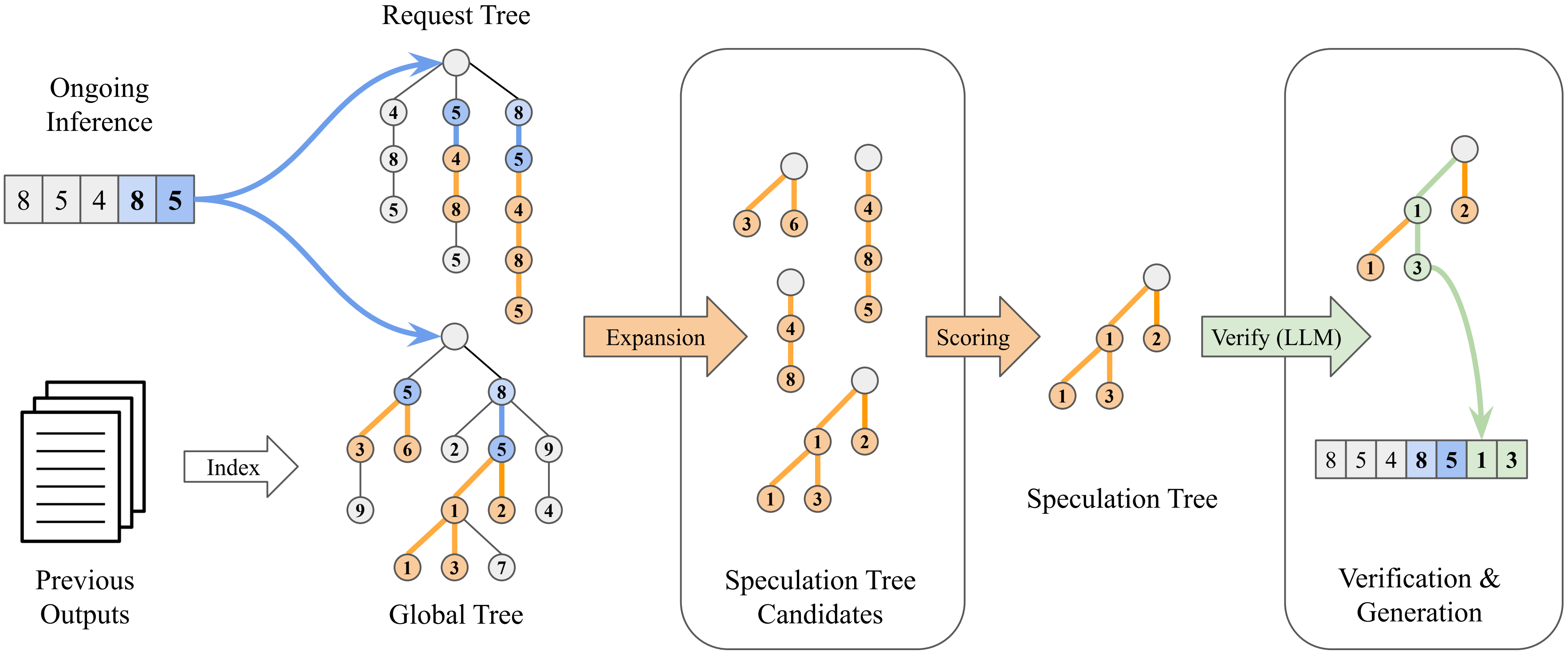
Expansion

Speculation Tree Candidates

Scoring

Speculation Tree

SuffixDecoding: A Tale of Two Trees 🌲🌳

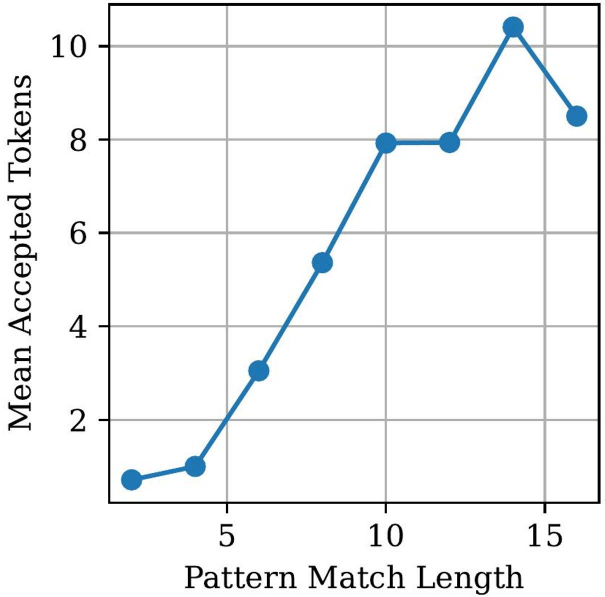


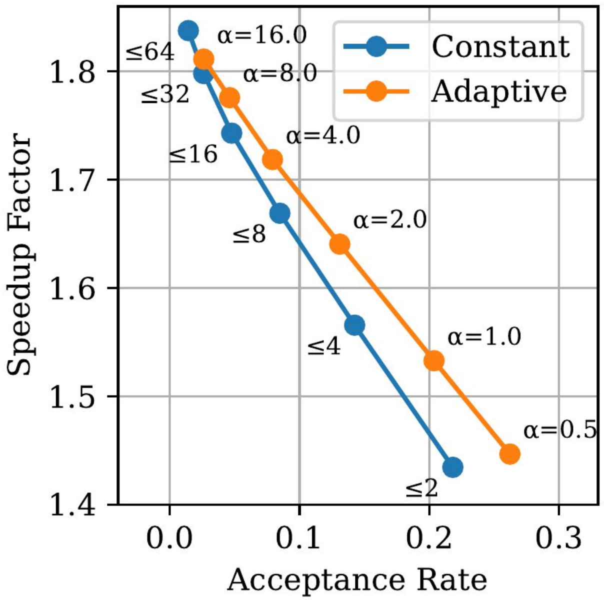
Agenda

1. **The Problem:** Latency in Modern AI Workloads
2. **Background:** Speculative Decoding 101
3. **Our Solution:** SuffixDecoding
4. **Key Features:** Adaptive, Fast, and Hybrid
5. **Evaluation & Results**
   - End-to-End Speedups
   - Live vLLM Integration
6. **Deeper Dives & Ablations**
7. **Conclusion**

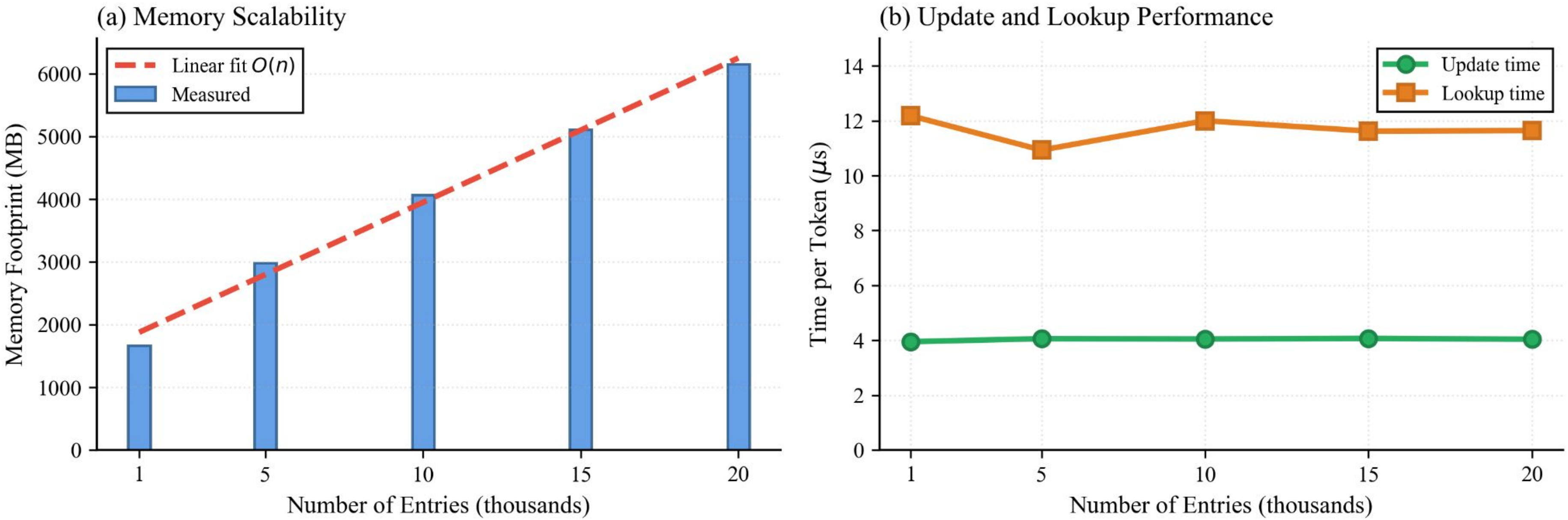
Efficient Verification via Adaptive Speculation

- Static speculation not viable because verification cost is substantial
- **Adaptive speculation** based on pattern match quality
- Formula: **MAX_SPEC = αp** (where p = pattern match length)

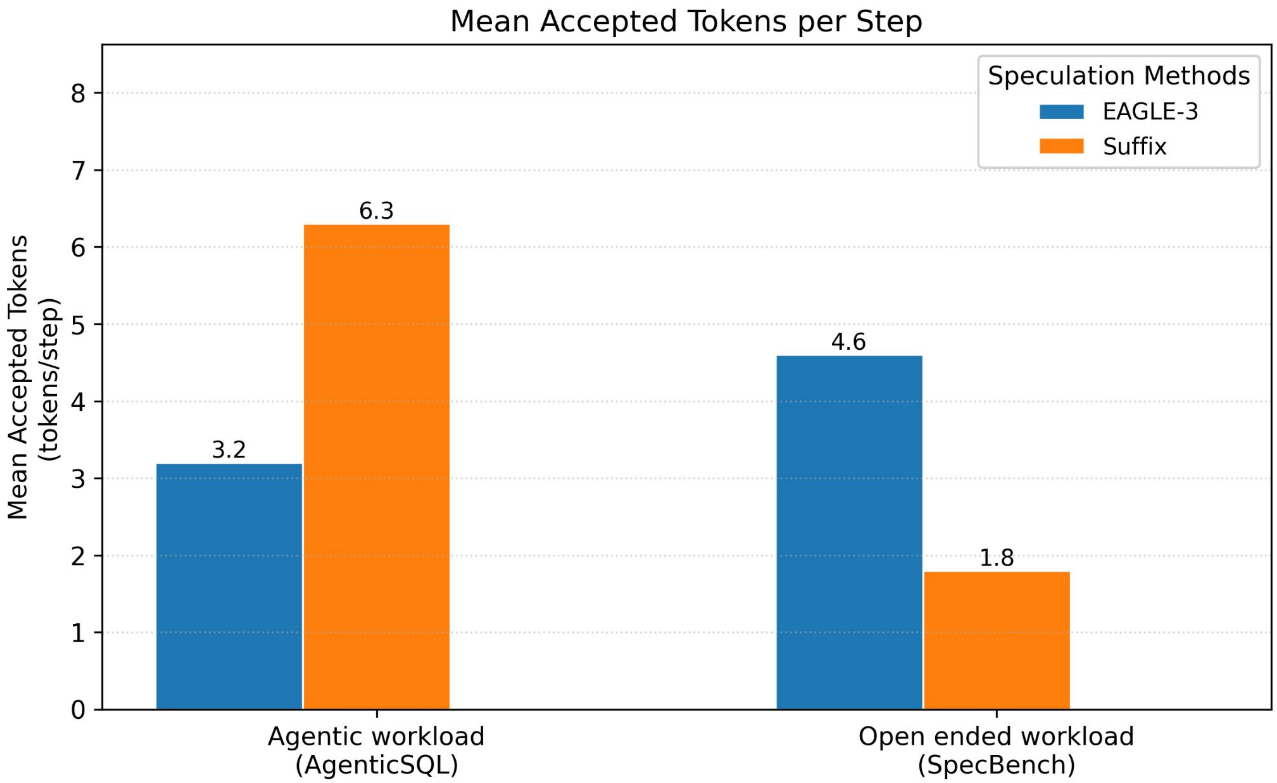




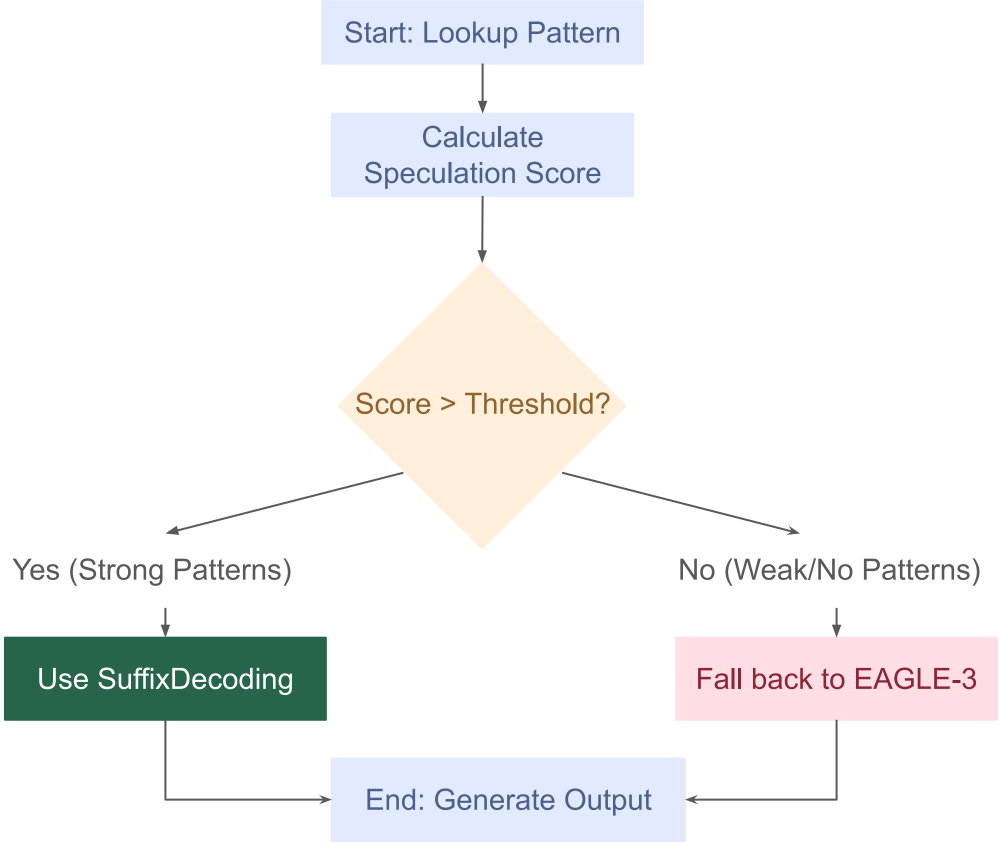
Blazingly Fast and Memory Efficient Speculation



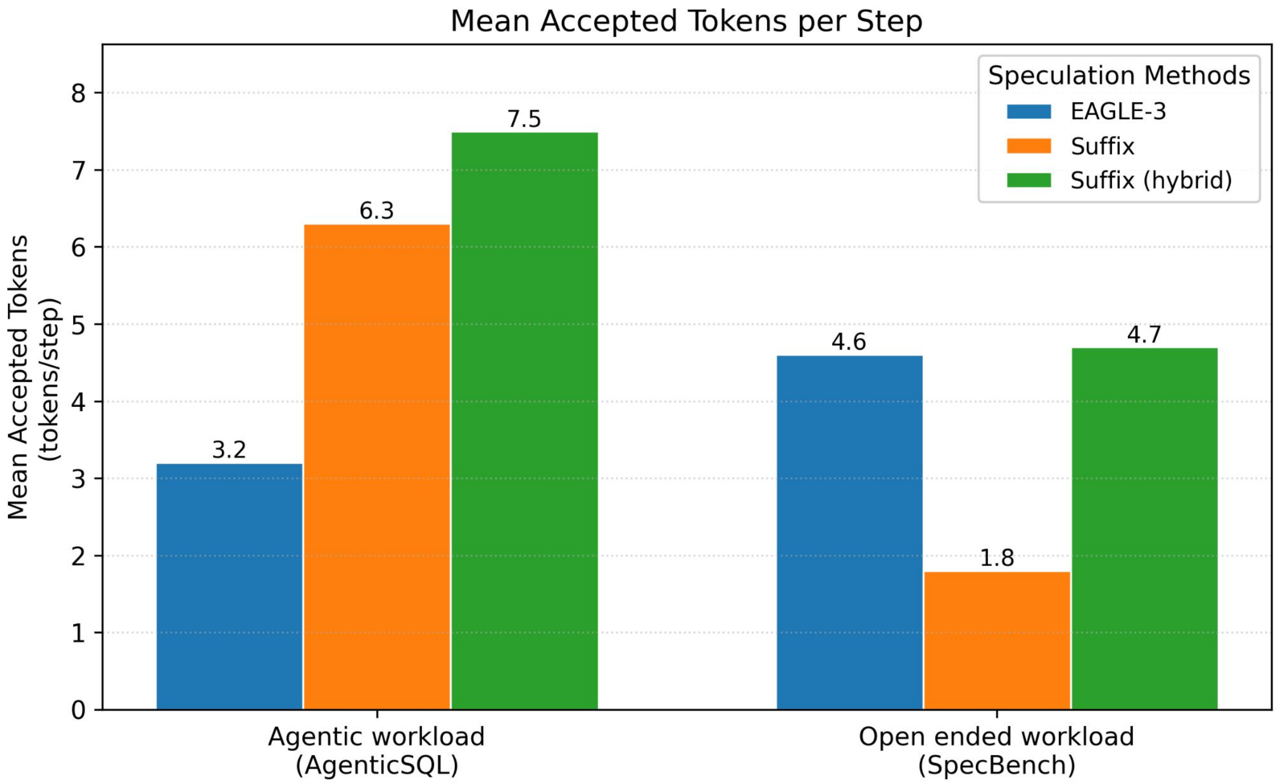
Robustness: SuffixDecoding + EAGLE-3 Hybrid



Robustness: SuffixDecoding + EAGLE-3 Hybrid



Robustness: SuffixDecoding + EAGLE-3 Hybrid

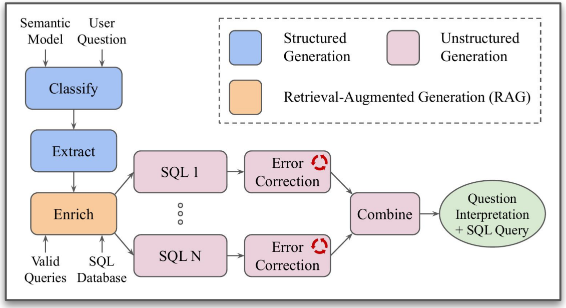


Agenda

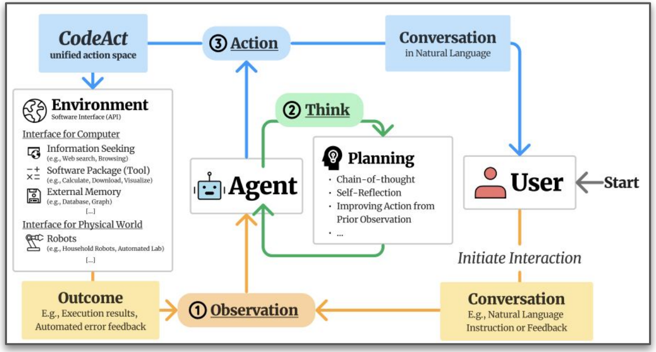
1. **The Problem:** Latency in Modern AI Workloads
2. **Background:** Speculative Decoding 101
3. **Our Solution:** SuffixDecoding
4. **Key Features:** Adaptive, Fast, and Hybrid
5. **Evaluation & Results**
    ○ End-to-End Speedups
    ○ Live vLLM Integration
6. **Deeper Dives & Ablations**
7. **Conclusion**

End–To-End Evaluation Benchmarks

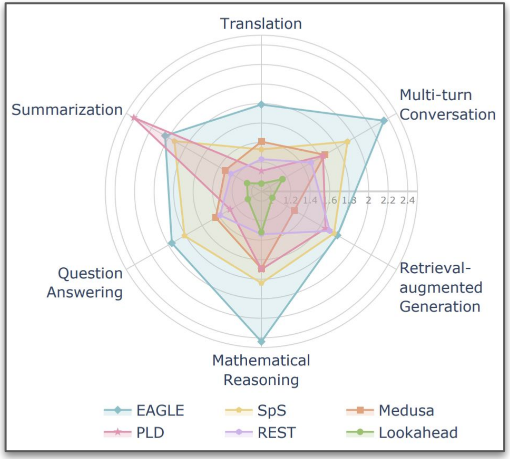
AgenticSQL



OpenHands CodeAct + SWE-Bench Verified



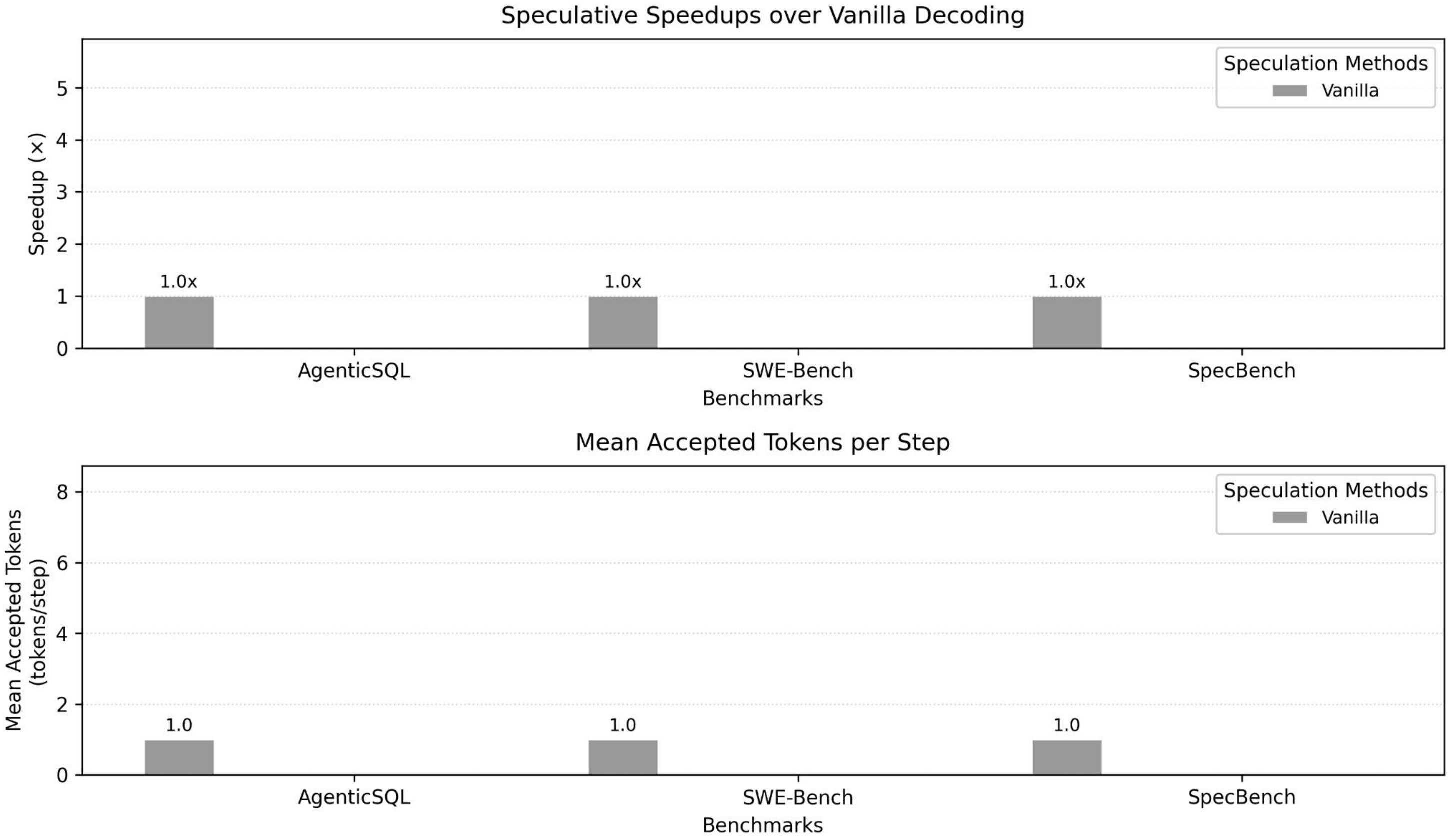
SpecBench



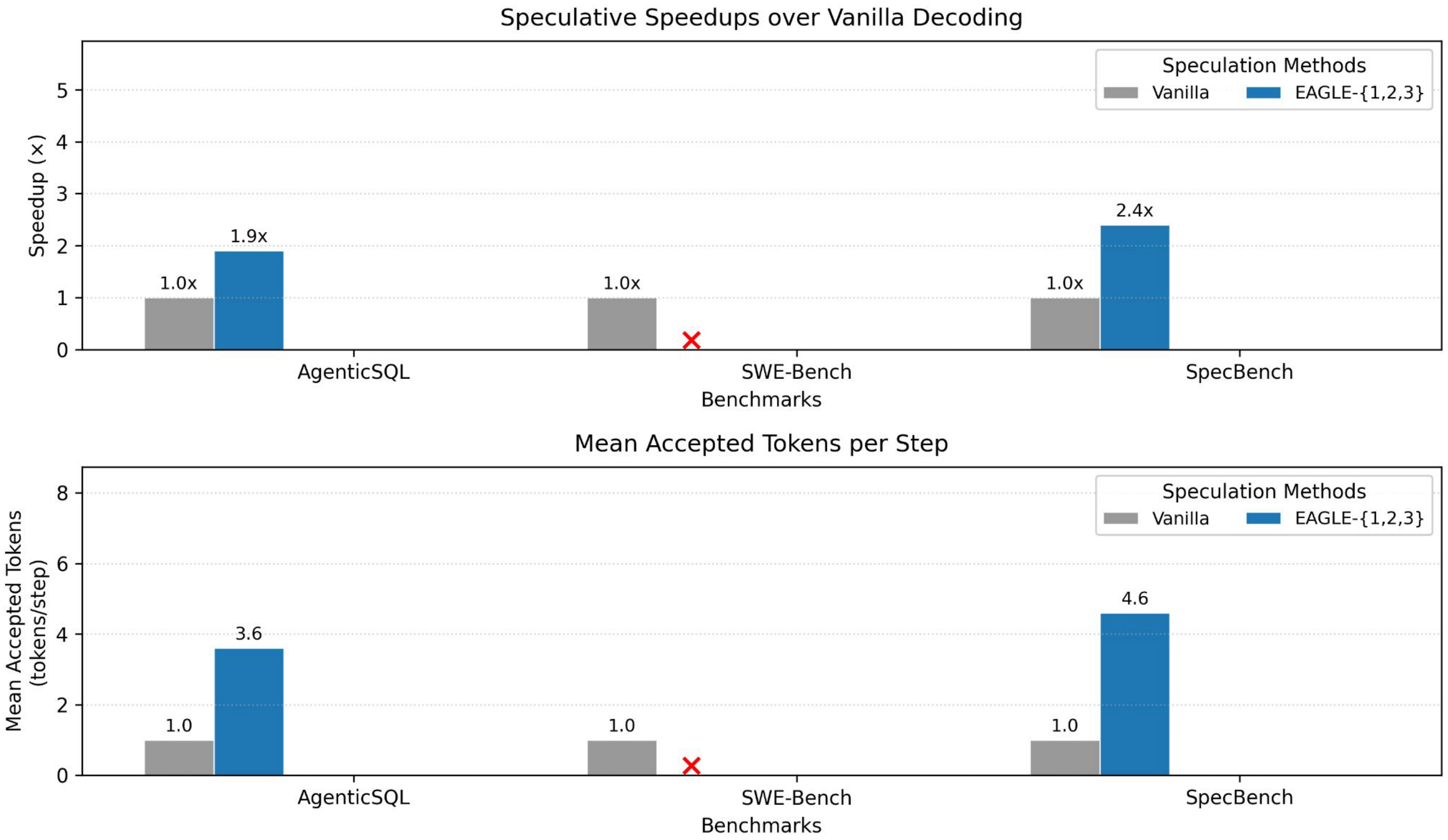
Agentic

Open-ended

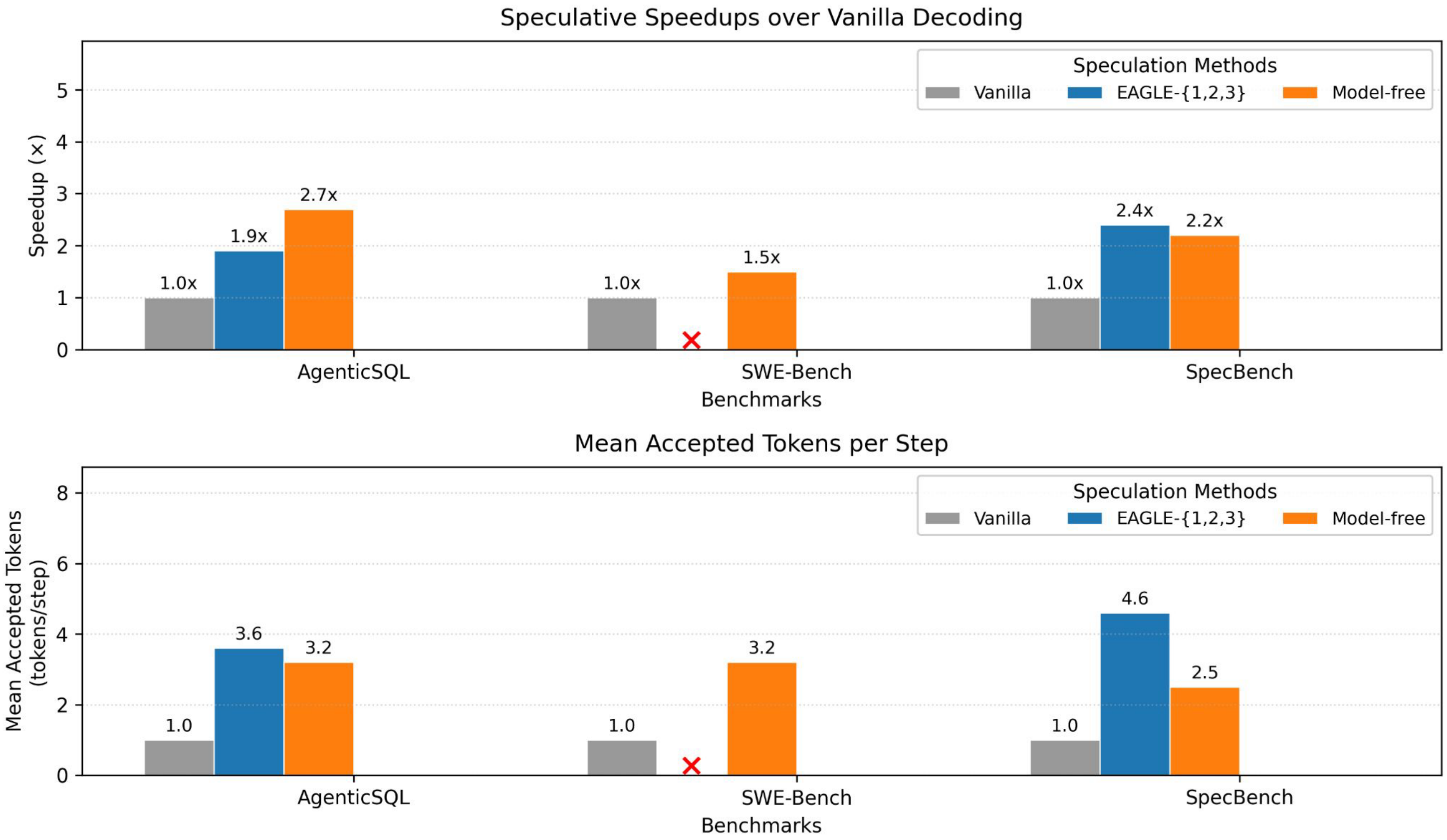
Up to 5.3x end-to-end speedup



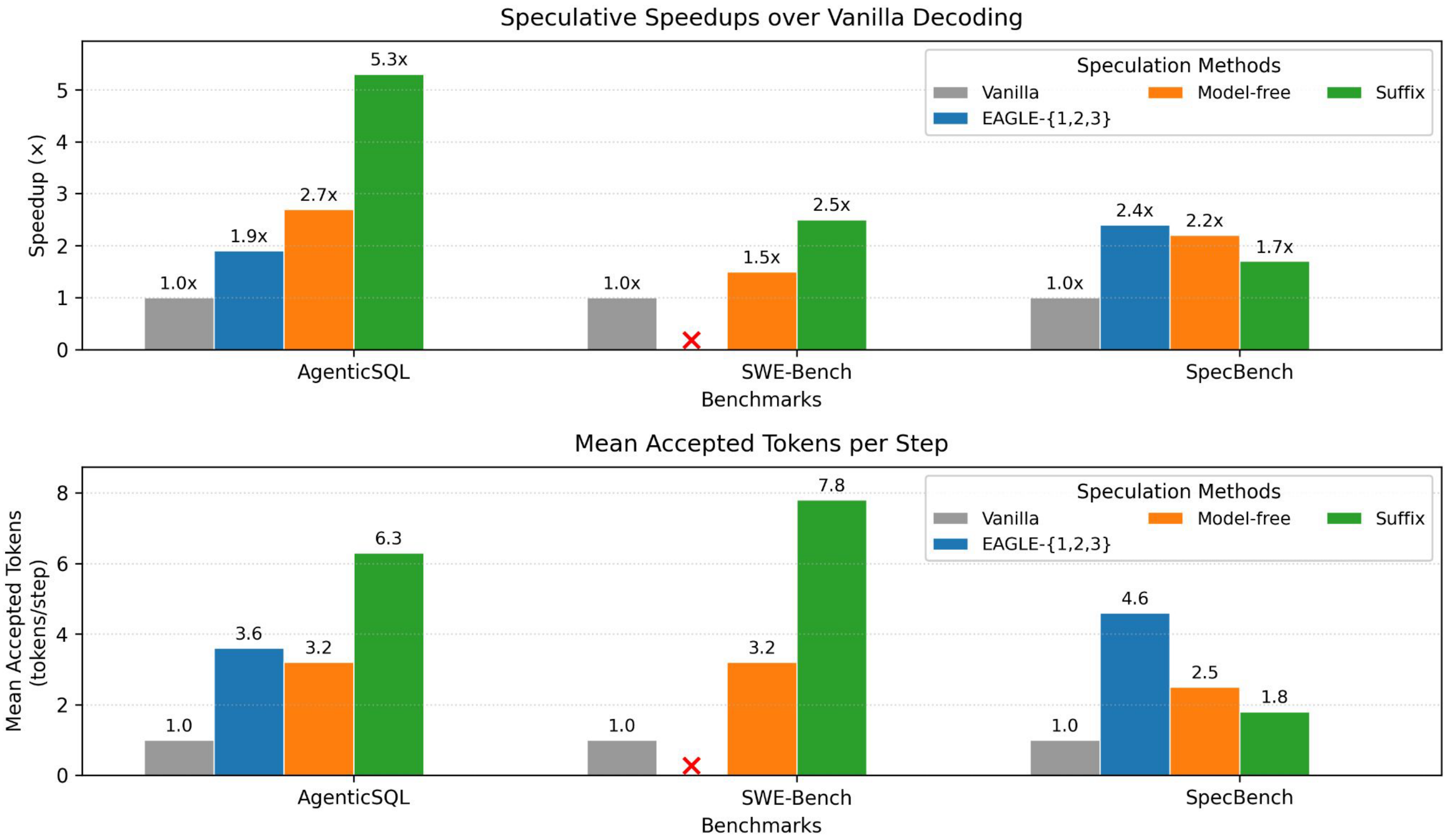
Up to 5.3x end-to-end speedup



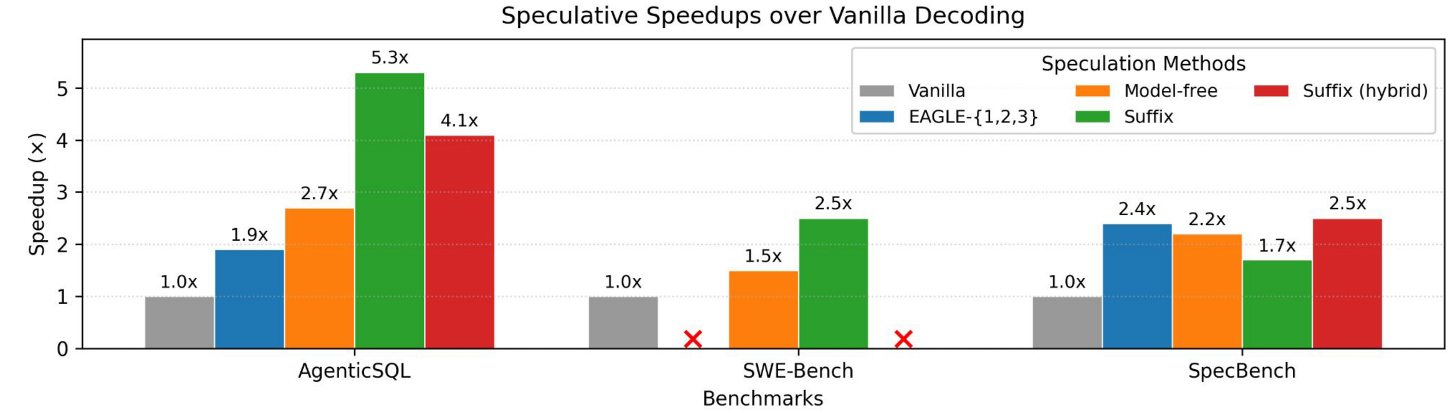
Up to 5.3x end-to-end speedup



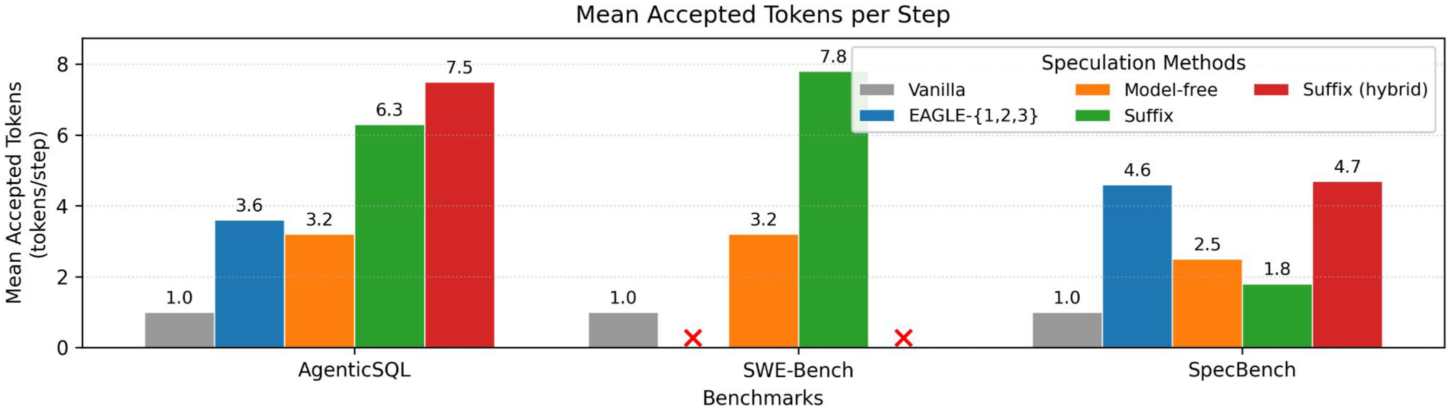
Up to 5.3x end-to-end speedup



Up to 5.3x end-to-end speedup

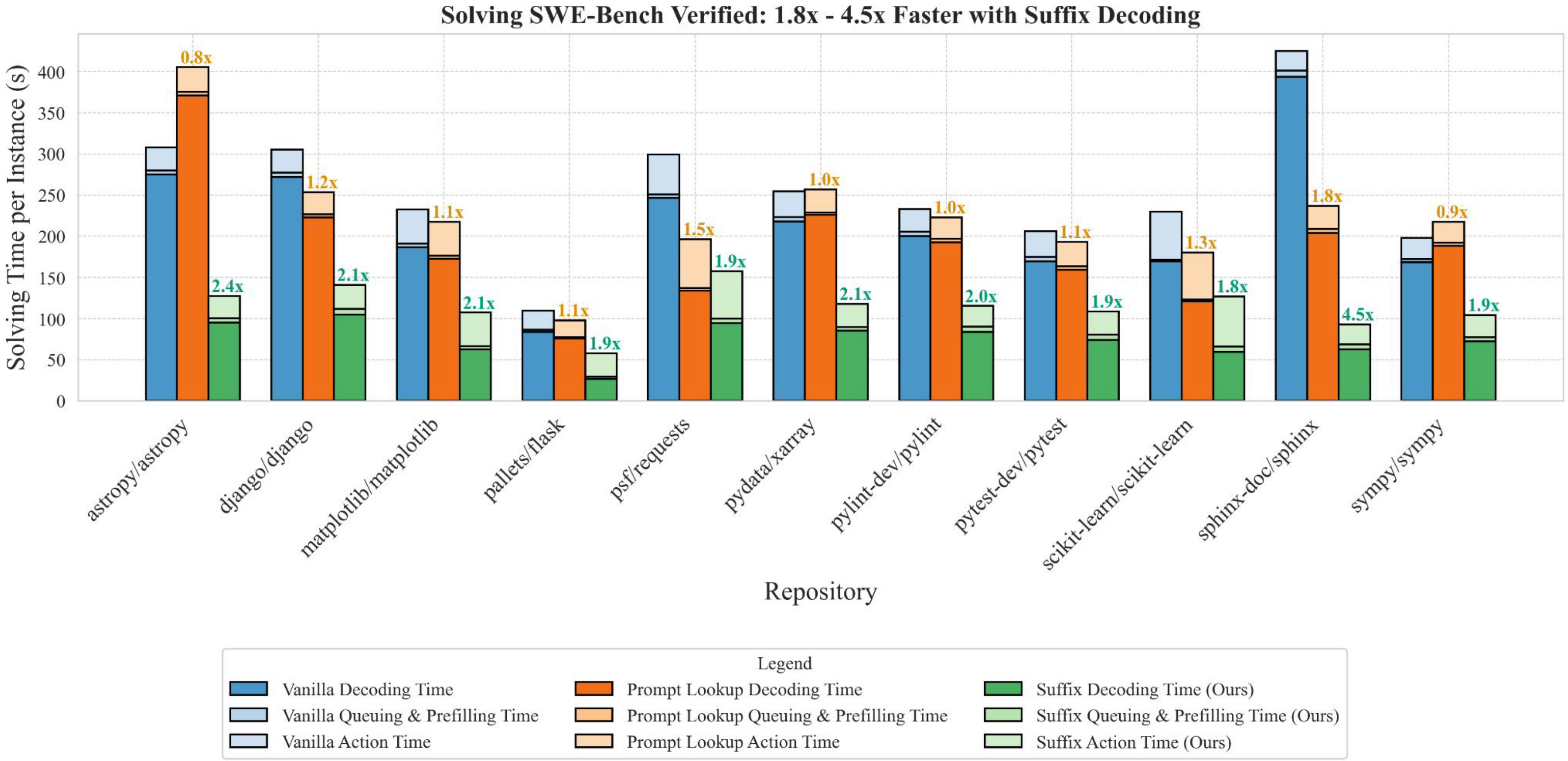


Speculative Speedups over Vanilla Decoding



Mean Accepted Tokens per Step

Up to 4.5x SWE-Bench Verified Task Completion Speedup



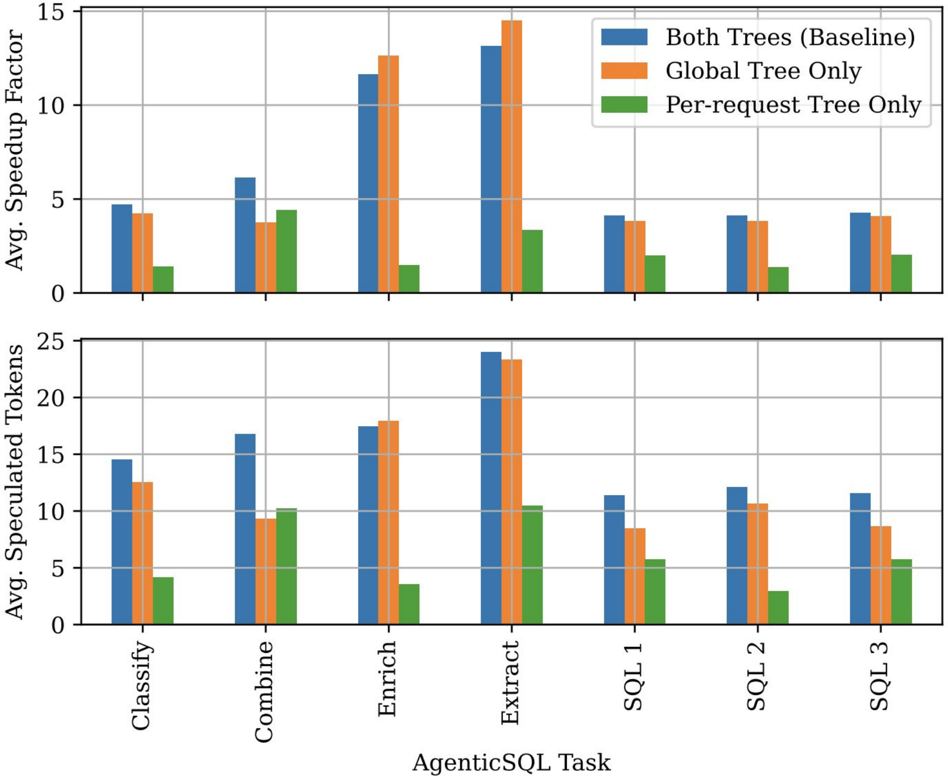
Agenda

1. **The Problem:** Latency in Modern AI Workloads
2. **Background:** Speculative Decoding 101
3. **Our Solution:** SuffixDecoding
4. **Key Features:** Adaptive, Fast, and Hybrid
5. **Evaluation & Results**
    - ○ Micro-Benchmarks
    - ○ End-to-End Speedups
6. **Deeper Dives & Ablations**
7. **Conclusion**

Why Two Trees?

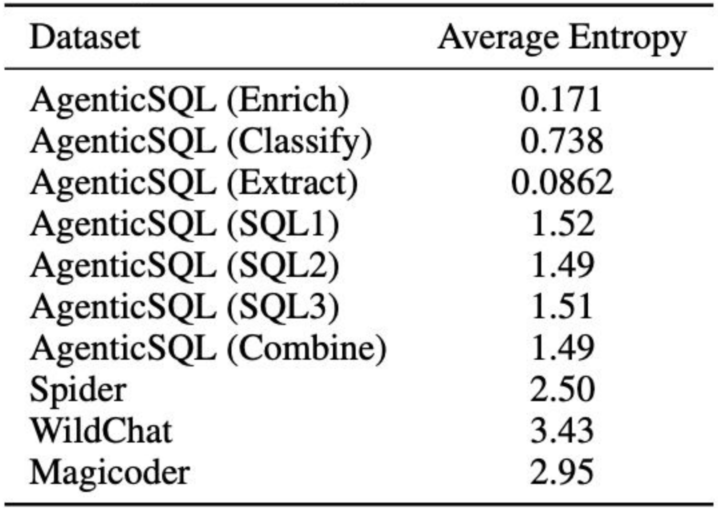
We ran an ablation to see what each suffix tree contributes.

- **Per-request Tree Only**: Decent speedup, but low.
- **Global Tree Only**: The long-term history provides most of the speedup.
- **Both Trees**: In almost every case, using both trees is the best.



How do we know if SuffixDecoding will work?

- Measure the empirical **entropy** (i.e. the "structuredness") of a workload with just 100 example outputs.
- Lower average entropy indicates more predictable outputs and better SuffixDecoding performance.
  - **AgenticSQL Enrich:** 0.171 entropy → 10.41× speedup
  - **SpiderSQL:** 2.50 entropy → 2.19× speedup
  - **WildChat (open-ended chat):** 3.43 entropy → modest speedup

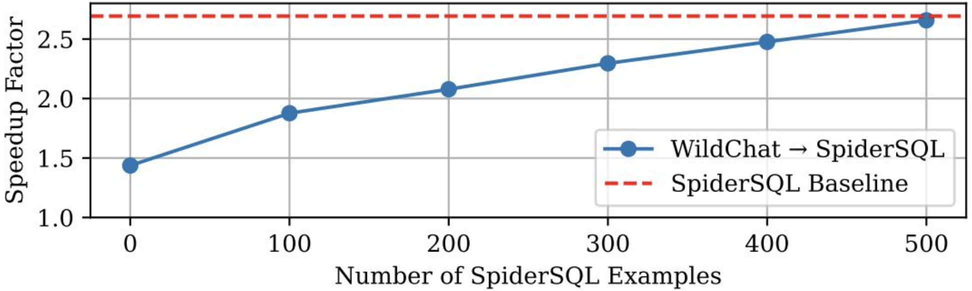


Dataset	Average Entropy
AgenticSQL (Enrich)	0.171
AgenticSQL (Classify)	0.738
AgenticSQL (Extract)	0.0862
AgenticSQL (SQL1)	1.52
AgenticSQL (SQL2)	1.49
AgenticSQL (SQL3)	1.51
AgenticSQL (Combine)	1.49
Spider	2.50
WildChat	3.43
Magicoder	2.95

What happens when the input distribution shifts?

🔎 Experiment:

1. **Train Cache** on **WildChat** (open-ended chat).
2. **Shift Workload:** Run inference on **SpiderSQL**
3. **Adapt:** We let SuffixDecoding *add* the new SpiderSQL outputs to the tree



📊 Results:

★ At 0 new examples, speedup is low (~1.5x) but *still better than vanilla*.

★ SuffixDecoding adapts *fast*. After 500 examples, already matching perfect cache

👉 **Takeaway:** The system is robust to distribution shift and adapts online *automatically*.

Agenda

1. **The Problem:** Latency in Modern AI Workloads
2. **Background:** Speculative Decoding 101
3. **Our Solution:** SuffixDecoding
4. **Key Features:** Adaptive, Fast, and Hybrid
5. **Evaluation & Results**
   - Micro-Benchmarks
   - End-to-End Speedups
6. **Deeper Dives & Ablations**
7. **Conclusion**

Conclusion

- ✓ SuffixDecoding achieves **5.3x end-to-end speedup** for agentic workloads.
- ✓ Requires **no model training.**
- ✓ Maintains **lossless** output quality.
- ✓ Can be **hybridized** with other speculation methods (e.g. EAGLE-3) for rapid open-ended generation.
- ✓ Available in vLLM

🏠 Project Page

Questions? Contact: goliaro@cs.cmu.edu