Monday, Feb 23rd, 2025

Decoding Acceleration

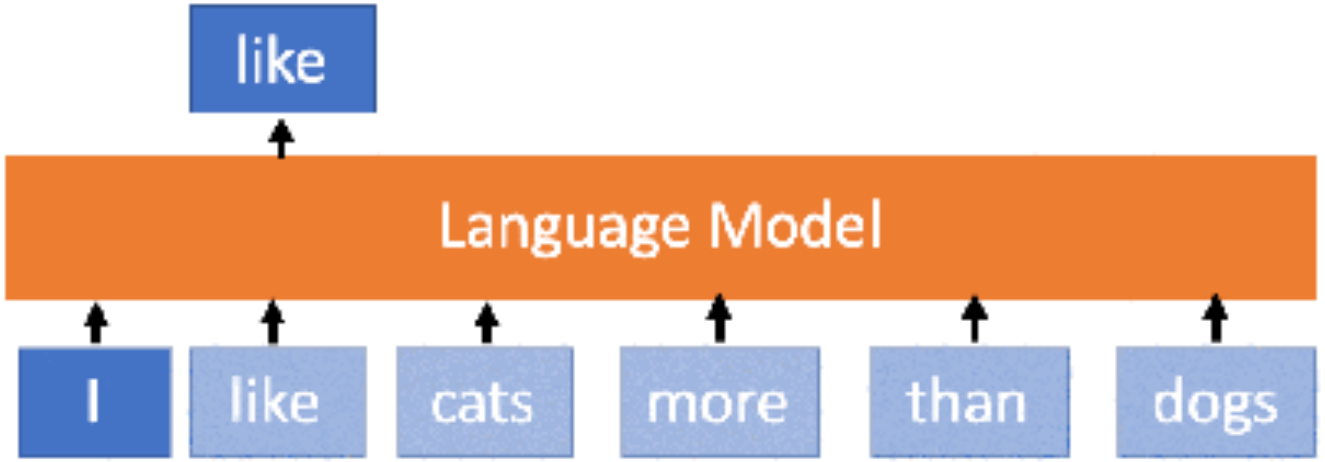
Huu Binh Ta
nzj6jt

Mark Do
mft6zc

Hunter Platt
kqr2pp

Overview

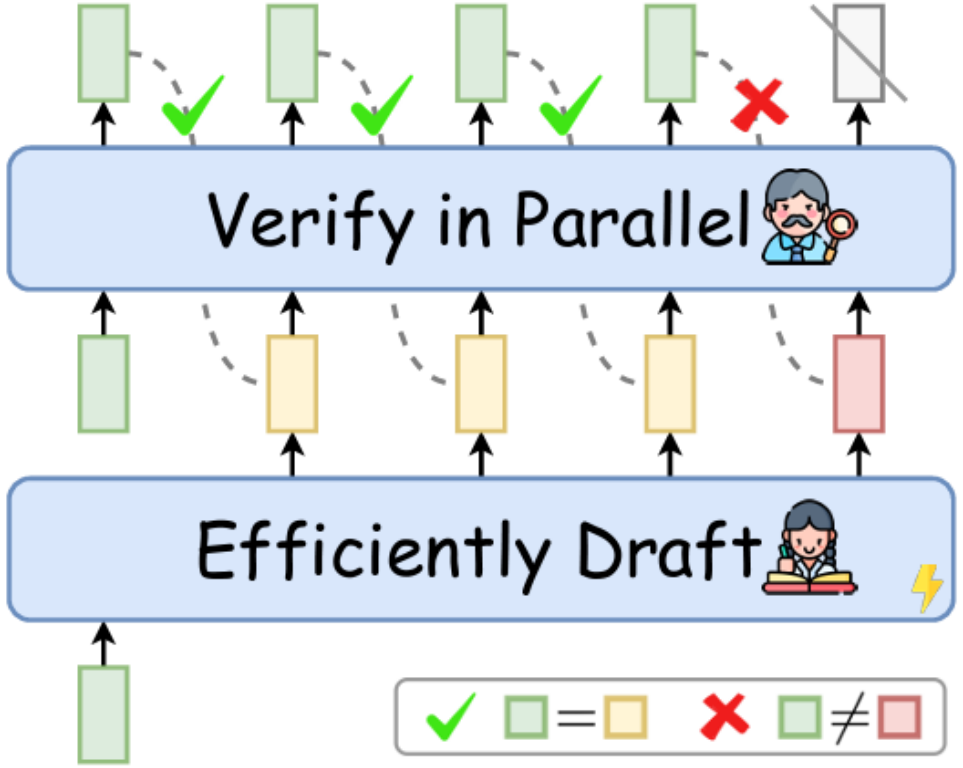
- **Problem with autoregressive language models:**
  - Tokens are generated one at a time, no parallelism across time steps.
  - One full forward pass per token → latency grows linearly with sequence length.



- **Question:** Can we accelerate generation process (or autoregressive decoding) without sacrificing output quality?

Overview

- **Core ideas:**
  - Use a lightweight predictor to propose several future tokens (draft tokens).
  - The original large model checks multiple proposed tokens in one forward pass.



Overview

- **Core ideas:**
  - Use a lightweight predictor to propose several future tokens (draft tokens).
  - The original large model checks multiple proposed tokens in one forward pass.
- **Papers:**
  1. Speculative Decoding: Use a lightweight draft model as the predictor.
  2. MEDUSA: Use multiple decoding heads to propose draft tokens.
  3. EAGLE: Model feature-level uncertainty to generate token proposals.

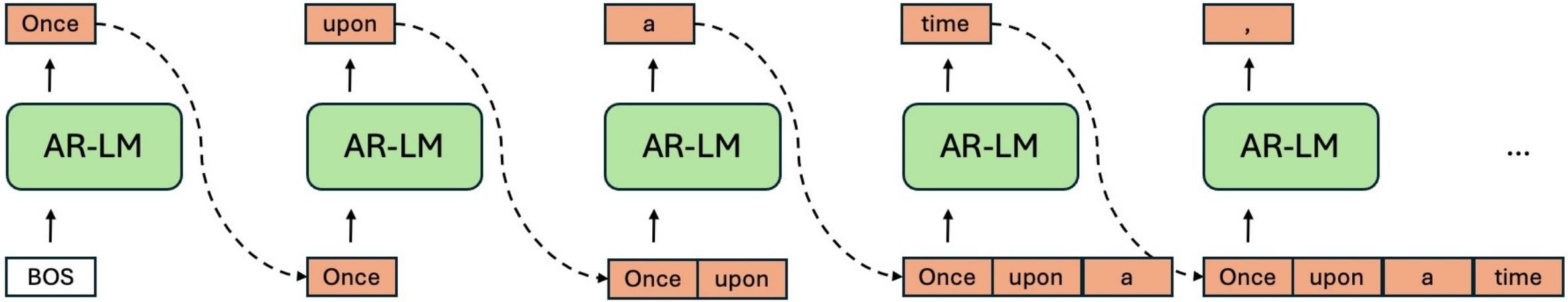
[1] Leviathan, Yaniv, Matan Kalman, and Yossi Matias. "Fast inference from transformers via speculative decoding." International Conference on Machine Learning. PMLR, 2023.
[2] Cai, Tianle, et al. "Medusa: Simple llm inference acceleration framework with multiple decoding heads." arXiv preprint arXiv:2401.10774 (2024).
[3] Li, Yuhui, et al. "Eagle: Speculative sampling requires rethinking feature uncertainty." arXiv preprint arXiv:2401.15077 (2024).

Fast inference from transformers via speculative decoding

ICML 2023

Problem & Motivation

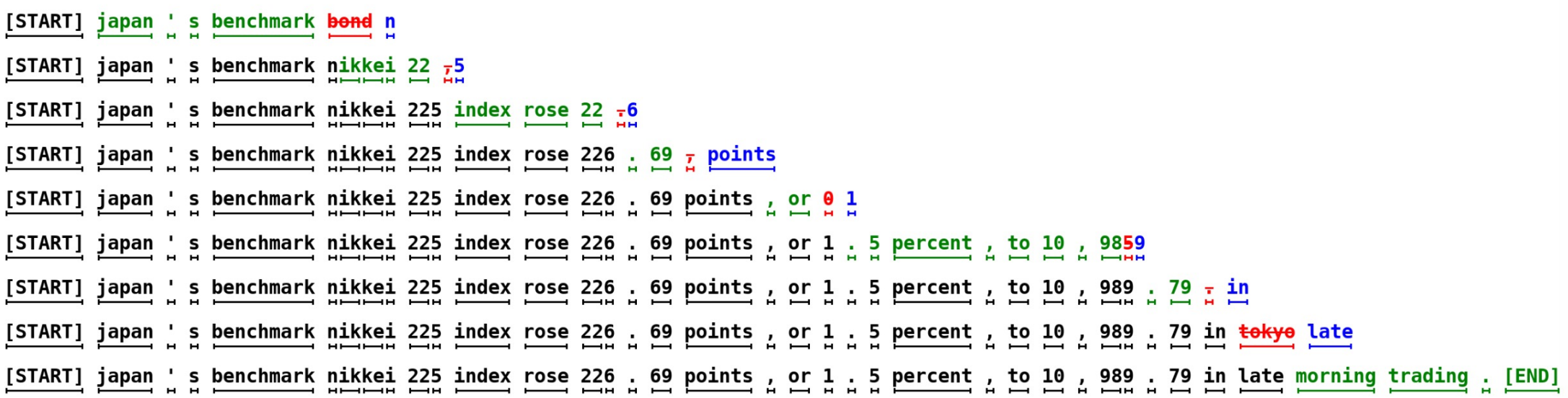
- Large autoregressive models are more capable than smaller models.
- HOWEVER, a single decode step from larger models is slower.



- This paper aims to achieve:
  - Inference time acceleration
  - Same output distribution
- Key observation: hard language-modeling tasks often include easier subtasks.

Small Models as Drafters

- **Ideas:**
  - Use a small model to propose draft tokens.
  - The target (base) model verifies parallelly and proposes tokens.



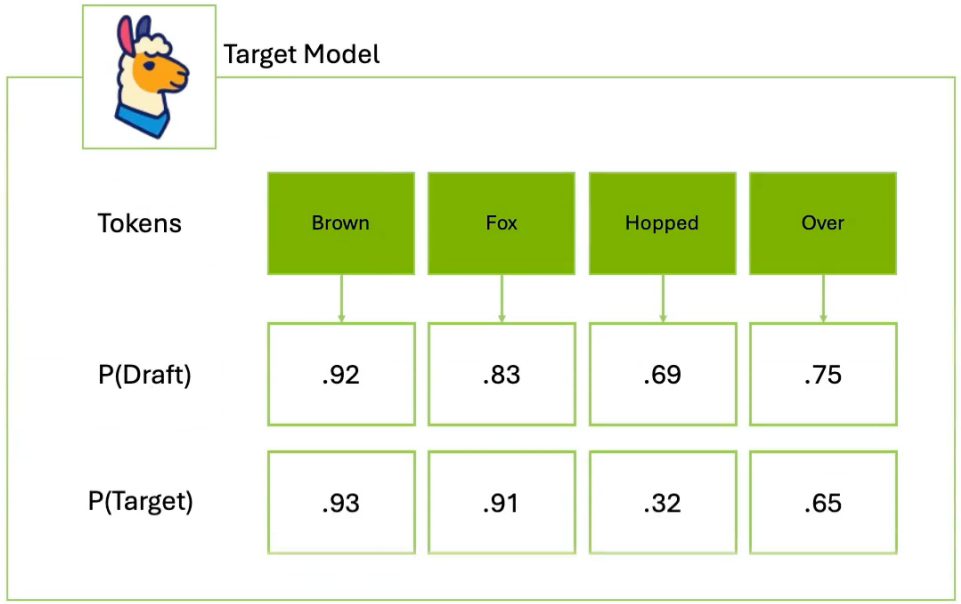
Small Models as Drafters

- **Ideas:**
  - Use a small model to propose draft tokens.
  - The target (base) model verifies parallelly and proposes tokens.
- **How to decide which draft tokens to be kept or discarded?**
  - Let $p(x)$: target model distribution
  - Let $q(x)$: draft model distribution
  - Sample $x \sim q(x)$
    - Accept with probabillity:

$$\min\left(1, \frac{p(x)}{q(x)}\right)$$

    - If reject, sample:

$$x \sim p'(x) = \text{norm}\big(\max\big(0, p(x) - q(x)\big)\big)$$



Pseudocode

Algorithm 1 SpeculativeDecodingStep

Inputs: $M_p, M_q, prefix$.
▷ Sample γ guesses $x_{1,\ldots,\gamma}$ from M_q autoregressively.
for $i = 1$ **to** γ **do**
  $q_i(x) \leftarrow M_q(prefix + [x_1, \ldots, x_{i-1}])$
  $x_i \sim q_i(x)$
end for
▷ Run M_p in parallel.
$p_1(x), \ldots, p_{\gamma+1}(x) \leftarrow$
  $M_p(prefix), \ldots, M_p(prefix + [x_1, \ldots, x_\gamma])$
▷ Determine the number of accepted guesses n.
$r_1 \sim U(0,1), \ldots, r_\gamma \sim U(0,1)$
$n \leftarrow \min(\{i - 1 \mid 1 \leq i \leq \gamma, r_i > \frac{p_i(x)}{q_i(x)}\} \cup \{\gamma\})$
▷ Adjust the distribution from M_p if needed.
$p'(x) \leftarrow p_{n+1}(x)$
if $n < \gamma$ **then**
  $p'(x) \leftarrow norm(max(0, p_{n+1}(x) - q_{n+1}(x)))$
end if
▷ Return one token from M_p, and n tokens from M_q.
$t \sim p'(x)$
return $prefix + [x_1, \ldots, x_n, t]$

Take the same time to iterate through multiple generated tokens as a single decode step from the target model.

Take the same time to create a new token as a single decode step from the target model.

Number of serial runs of the target model is **at most equal** to standard autoregressive decoding!

Analysis 1: Token Generation Efficiency

- **Definitions:**
  - Acceptance rate (β): probability that a draft token is accepted.
  - Expected value of acceptance rate $\alpha = \mathbb{E}(\beta)$ → How good is the draft model.
- **Core formula:**

$$E(\text{tokens per iteration}) = \frac{1 - \alpha^{\gamma+1}}{1 - \alpha}$$

- **Insights:**
  - Even if $\alpha = 0$, the expected tokens per iteration is 1.
  - Theoretical Bound: If $\gamma \to \infty$, the expected speedup factor approaches $\frac{1}{1-\alpha}$.

Analysis 2: Quantitative Metric for Approximation Quality

- **Definitions:**
  - $D_{LK}(p, q) = \sum_x |p(x) - M(x)|$, where M is the average of p and q.
- **Theorem:** probability of accepting a draft token is related to this divergence:

$$\beta = 1 - D_{LK}(p, q)$$

- **Corollary:** Expected value of acceptance rate is:

$$\alpha = 1 - E\big(D_{LK}(p, q)\big) = E(\min(p, q))$$

- **Insight:** Show how the similarity between two models affect acceptance rate.

Analysis 3: Walltime Improvement

- **Definitions:**
  - Cost coefficient (c): Ratio between time for a run of the draft and target model.
- **Theorem:** Expected improvement factor in total walltime (num. tokens per time) is:

$$\text{Improvement Factor} = \frac{1 - \alpha^{\gamma+1}}{(1-\alpha)(c\gamma+1)}$$

  - Numerator: Average number of tokens produced per iteration.
  - Denominator: Total time cost per iteration
- **Insights:**
  - Existence of Improvement: If $\alpha > c$, there exists a value for γ that results in a faster decoding time.
  - Minimum Gain: When $\alpha > c$, improvement factor is at least $\frac{1+\alpha}{1+c}$.

Analysis 4: Arithmetic Operations vs. Latency

- **Increased Computational Workload:**
  - Parallel Execution Cost: run parallel evaluations using the target.
  - Wasted Computation: If a draft token is rejected, arithmetic operations performed on all subsequent tokens has no contribution.
- **Theorem:** Expected factor of increase in total operations is:

$$\text{Factor} = \frac{(1-\alpha)(\hat{c}\gamma+\gamma+1)}{1-\alpha^{\gamma+1}}$$

  - $\hat{c}$: ratio of arithmetic operations per token of the draft related to target model.
- **Insights:** If α is low, the increase in the number of arithmetic operations is high.
- **Weight Reuse:** weights and the KV cache are read only once per iteration

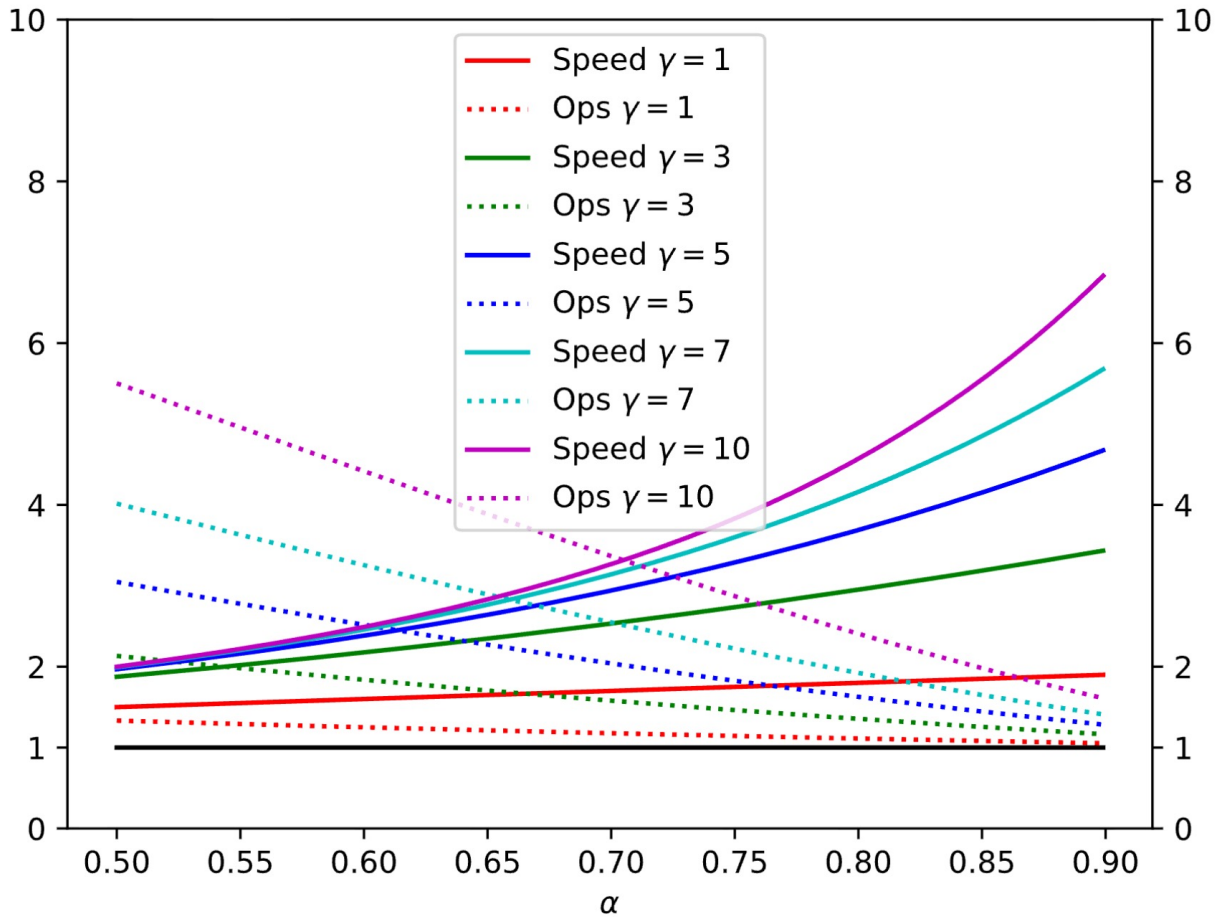
  → the total number of memory accesses can go down.

Analysis 5: Tuning Number of Draft Tokens

- **Optimal number of draft tokens depends on walltime improvement:**

$$\text{Improvement Factor} = \frac{1-\alpha^{\gamma+1}}{(1-\alpha)(c\gamma+1)}$$

  - As α increases (better draft model), the optimal γ also increases.
  - As c increases (slower draft model), the optimal γ decreases.



Analysis 6: Draft Models

- Speculative decoding maintains the exact output distribution of the target model.
- **Proof:**
  - We have: $P(x = x') = P(\text{accepted}, x = x') + P(\text{rejected}, x = x'))$.
  - Path 1 (accepted): $P(\text{accepted}, x = x') = q(x') \cdot \min\left(1, \frac{p(x')}{q(x')}\right) = \min\left(q(x'), p(x')\right)$
  - Path 2 (rejected): $P(\text{rejected}, x = x') = (1 - \beta) \cdot p'(x')$
  - We also have: $p'(x) = \frac{\max\left(0, p(x) - q(x)\right)}{1-\beta} = \frac{p(x) - \min\left(q(x), p(x)\right)}{1-\beta}$
  - Then: $P(\text{rejected}, x = x') = p(x') - \min\left(q(x'), p(x')\right)$
  - As a result: $P(x = x') = \min\left(q(x'), p(x')\right) + p(x') - \min\left(q(x'), p(x')\right) = p(x')$

→ The lossless guarantee holds for any choice of draft model

Experimental Results

- ENDE: English-to-German translation task.
- CNNDM: Text summarization task.
- Target model: T5-XXL (11B).
- Draft models:
  - T5-large (800M)
  - T5-base (250M)
  - T5-small (77M)
- All approaches improve speed.
- T5-small is the best draft model.

Table 2. Empirical results for speeding up inference from a T5-XXL 11B model.

TASK	M_q	TEMP	γ	α	SPEED
ENDE	T5-SMALL ★	0	7	0.75	**3.4X**
ENDE	T5-BASE	0	7	0.8	2.8X
ENDE	T5-LARGE	0	7	0.82	1.7X
ENDE	T5-SMALL ★	1	7	0.62	**2.6X**
ENDE	T5-BASE	1	5	0.68	2.4X
ENDE	T5-LARGE	1	3	0.71	1.4X
CNNDM	T5-SMALL ★	0	5	0.65	**3.1X**
CNNDM	T5-BASE	0	5	0.73	3.0X
CNNDM	T5-LARGE	0	3	0.74	2.2X
CNNDM	T5-SMALL ★	1	5	0.53	**2.3X**
CNNDM	T5-BASE	1	3	0.55	2.2X
CNNDM	T5-LARGE	1	3	0.56	1.7X

Experimental Results

- The theoretical predictions mostly match the measured runtimes.
- Larger differences are due to:
  - Implementation latency: No parallel execution due to system-level optimizations.
  - The simplifying assumption that β is independent and identically distributed.

Table 4. Expected improvement factor (EXP) vs. empirically measured improvement factor (EMP).

TASK	M_q	TEMP	γ	α	c	EXP	EMP
ENDE	T5-SMALL	0	7	0.75	0.02	3.2	3.4
ENDE	T5-BASE	0	7	0.8	0.04	3.3	2.8
ENDE	T5-LARGE	0	7	0.82	0.11	2.5	1.7
ENDE	T5-SMALL	1	7	0.62	0.02	2.3	2.6
ENDE	T5-BASE	1	5	0.68	0.04	2.4	2.4
ENDE	T5-LARGE	1	3	0.71	0.11	2.0	1.4
CNNDM	T5-SMALL	0	5	0.65	0.02	2.4	3.1
CNNDM	T5-BASE	0	5	0.73	0.04	2.6	3.0
CNNDM	T5-LARGE	0	3	0.74	0.11	2.0	2.2
CNNDM	T5-SMALL	1	5	0.53	0.02	1.9	2.3
CNNDM	T5-BASE	1	3	0.55	0.04	1.8	2.2
CNNDM	T5-LARGE	1	3	0.56	0.11	1.6	1.7

Limitations and Future Directions

Limitation

- Infrastructure Overhead: Requires loading and managing two separate models.
- Variable Efficiency: Speedup is highly dependent on the "guessability" of γ.
- Computation Waste: Increases total FLOPs; may not be ideal for energy-constrained application.

Potential Improvement

- Self-Speculation: Modify and retrain target models to draft tokens.
- Adaptive Speculation: Implementing a dynamic γ that adjusts in real-time.
- Specialized Kernels: Develop GPU kernels optimized for the asymmetric workload of speculative decoding

MEDUSA: Simple LLM inference acceleration framework with multiple decoding heads

ICML 2024

Problems

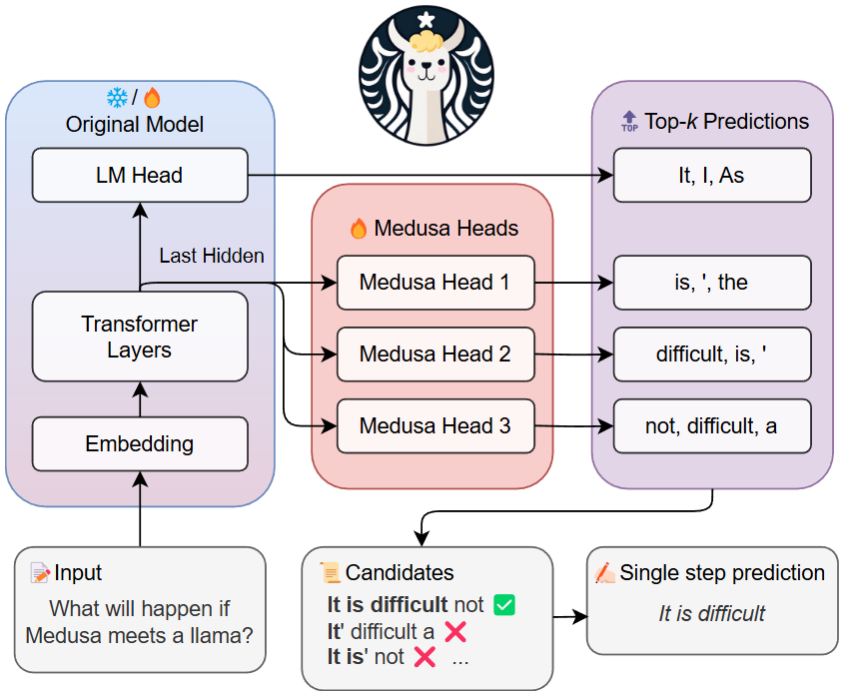
- **Autoregressive Decoding:**
  - **Sequential decoding**: Tokens generated one at a time; no parallelism
  - **Memory bottleneck**: Full model weights loaded for every token
  - **Low compute utilization**: GPU often idle waiting on memory
  - **Underutilized hardware**: Performance limited by memory, not compute
- **Speculative Decoding:**
  - **Draft model tax**: Requires training and serving an extra model
  - **Alignment difficulty**: Draft and target models must closely match
  - **System complexity**: Synchronizing two models increases overhead
  - **Wasted compute**: Mismatches cause rejected tokens and inefficiency

Problem & Motivation

- **Goal:**
  - Break the “memory wall” to significantly speed up LLM inference
- **Question:**
  - Can we get speculative decoding’s speed benefits without training/serving a second model?
- **Medusa’s direction:**
  - Use the target model’s own hidden states to generate speculative candidates (via added heads).

Solution: The Medusa Heads

- Add lightweight Medusa heads on top of a pre-trained LLM to propose multiple future tokens in parallel.



- Each head predicts a different future offset
- Outputs top-k candidate continuations
- Candidates verified in a single parallel pass

Generating Candidates

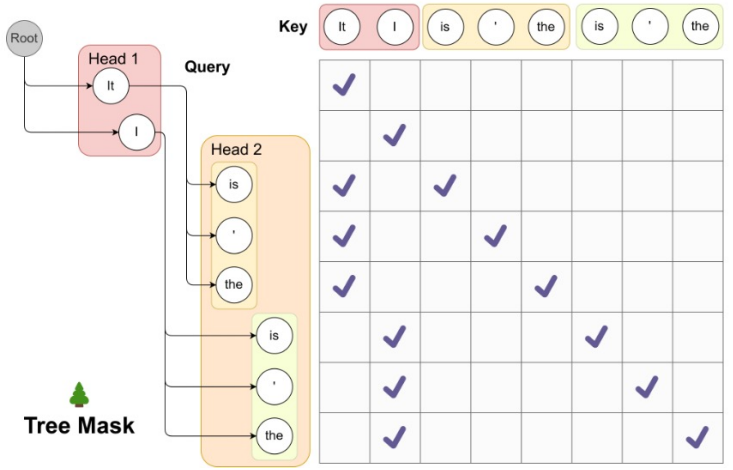
- Run the backbone LLM once on the current prefix to get the final hidden state h_t
- For each Medusa head k, compute a token distribution from the same h_t:

$$p_t^{(k)} = \text{softmax}\left(W_2^{(k)} \cdot \left(\text{SiLU}(W_1^{(k)} \cdot h_t) + h_t\right)\right),$$
$$\text{where } W_2^{(k)} \in \mathbb{R}^{d \times V}, W_1^{(k)} \in \mathbb{R}^{d \times d}.$$

- From each $p_t^{(k)}$, take the top-s_k tokens as that head's candidate options
- Combine candidates across heads to form candidate continuations
- Verify all candidate continuations in parallel using tree attention and accept the longest valid prefix

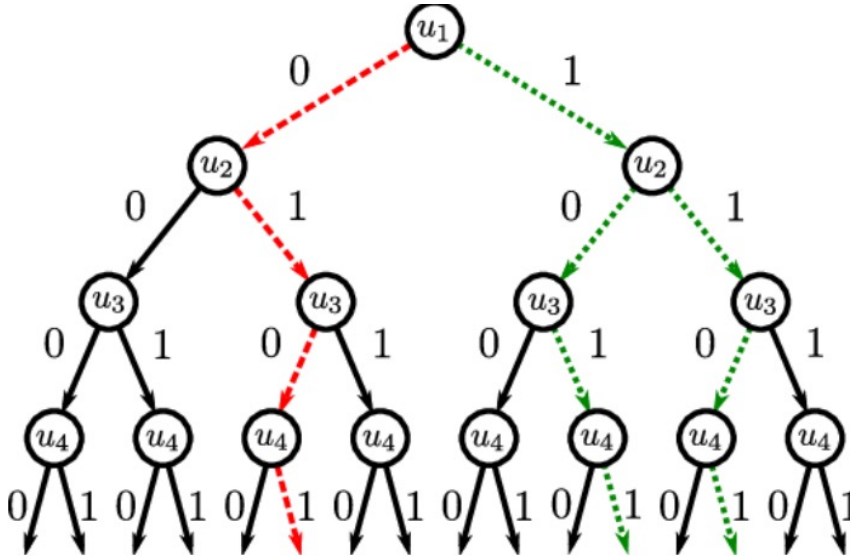
Tree-Based Verification

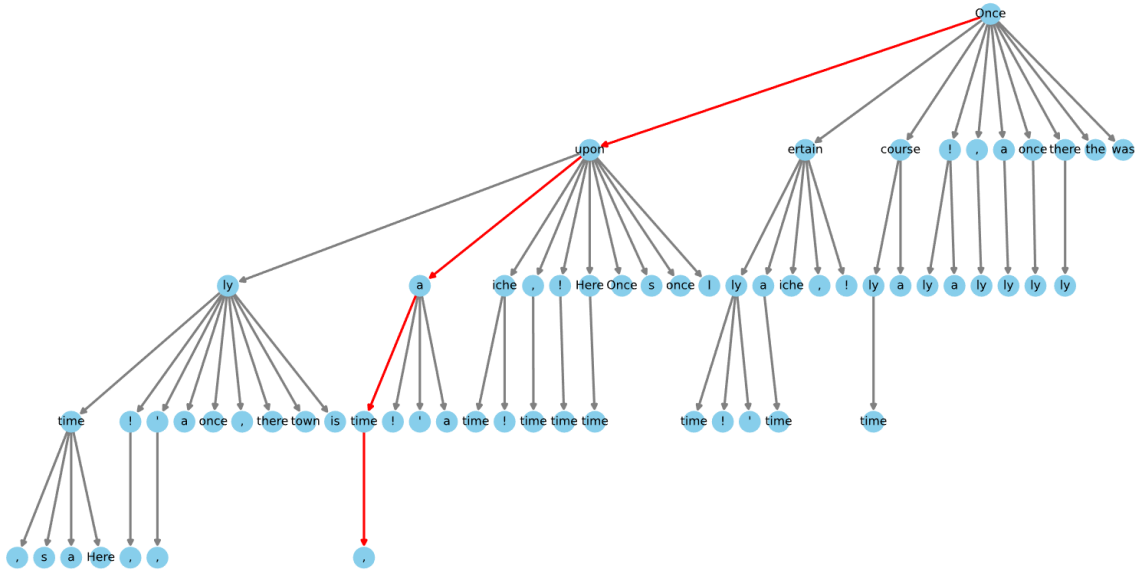
- **Problem:**
  - Multiple heads generate many candidate continuations that would take too long to check one by one



- **Solution:**
  - Candidate Tree: MEDUSA constructs a "tree" where each node is a potential future token
  - Tree Mask: A specialized attention mask is applied so that each node only "sees" its actual ancestors in the tree
  - One-pass Verification: Flatten the tree and verify all candidates in a single backbone forward pass
- **Acceptance:**
  - The longest path that matches the original model's distribution is accepted.

Dense vs Sparse Trees

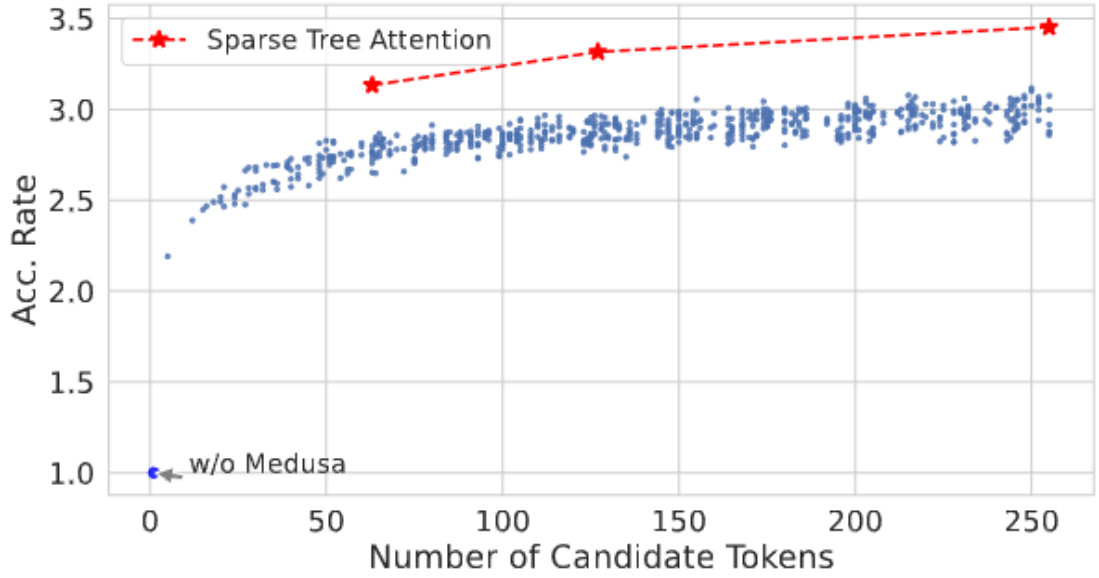


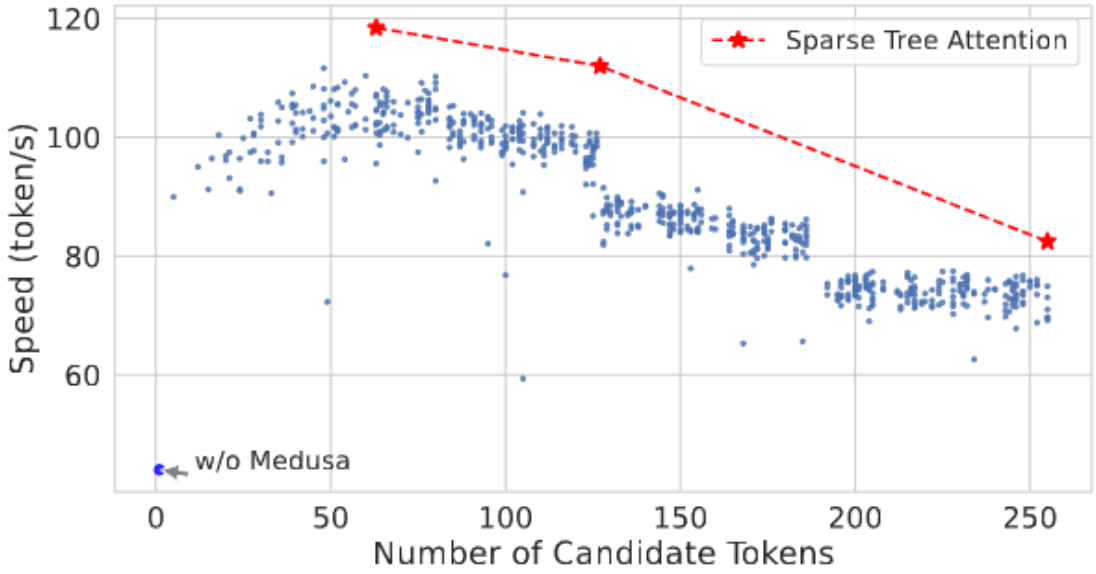


- Combines top-k tokens from each head into all possible paths
- Explores more sequences
- May increase acceptance, but adds many low-probability paths
- Verification becomes very expensive

- Keeps only high-value branches
- Unlikely paths are pruned early on
- Focuses compute on high-probability sequences
- Achieves similar acceleration with fewer candidates

Dense vs Sparse Trees Tradeoffs





- Acceleration rate increases with more candidates, but diminishing returns
- Actual speed peaks at moderate tree sizes, then drops as overhead grows
- optimized sparse trees achieve higher acceleration + higher speed at the same, or smaller, candidate budget
- Best performance comes from a small/optimized tree

Typical Acceptance

- **Strict Verification Limits**:
  - Most faithful to baseline decoding, but can be overly strict
  - High entropy leads to more rejections and weaker speedups
  - Can reject tokens that are plausible but not strongly supported
- **Typical Acceptance Idea**:
  - Accept tokens that are typical under the backbone and plausible given uncertainty
  - Acceptance rule uses an entropy-adaptive threshold:
    - $p_{\text{original}}(x) > \min\left(\epsilon,\ \delta e^{-H(p)}\right)$
  - Benefit: Higher acceptance gives more tokens per step and faster decoding
  - Tradeoff: Sampling is approximate and not exactly distribution matching

MEDUSA-1: Frozen Backbone

- **Core Concept**
  - Freeze the backbone LLM and train only the lightweight Medusa heads
  - Head k predicts the token at offset k+1 ahead using the same hidden state
  - Train with cross-entropy and downweigh farther-ahead heads with λ_k

$$\mathcal{L}_{\text{MEDUSA-1}} = \sum_{k=1}^{K} -\lambda_k \log p_t^{(k)}(y_{t+k+1})$$

- **Outcomes & Trade-offs**
  - Preserves original model quality because backbone weights stay fixed
  - Fast to train since only small heads are updated
  - Larger K increases speculation depth but reduces per-head accuracy and acceptance
  - Example: ~5 hours on a single A100 for Vicuna-7B with ~60k ShareGPT samples

MEDUSA-2: Joint Training

- **Core Concept**
  - Jointly train backbone and Medusa heads so the backbone becomes Medusa-aware and head accuracy improves
  - Reduces mismatch between head proposals and backbone verification

$$\mathcal{L}_{\mathcal{LM}} = -\log p_t^{(0)}(y_{t+1})$$

$$\mathcal{L}_{\text{MEDUSA-2}} = \mathcal{L}_{\text{LM}} + \lambda_0 \mathcal{L}_{\text{MEDUSA-1}}$$

- **Three Joint Training Methods**
  - **Combined loss:** Add backbone cross-entropy loss to head loss, with λ_0 balancing backbone accuracy and head training.
  - **Differential learning rates**: Use a smaller learning rate for the backbone and a larger learning rate for the heads.
  - **Heads warmup**: Train only the heads first, then jointly train backbone and heads to prevent early large head gradients from distorting the backbone.

Self-Distillation

- **Goal**: Train Medusa heads even when you don't have the model's original fine-tuning data/recipe
- Start with a small set of seed prompts
- Use the target model itself to generate responses to those prompts
- Collect prompt-response pairs to form a synthetic dataset
- For MEDUSA-1: Train the Medusa heads on the synthetic data while keeping the backbone frozen
- For MEDUSA-2: Jointly train the heads and the backbone using a special distillation loss (KL divergence).
- This allows Medusa to be added to Reinforcement Learning from Human Feedback (RLHF) chat models or models where the original training data is private or unavailable.

Experiments

- **Models Tested**
  - Vicuna-7B and Vicuna-13B (public datasets), Vicuna-33B (private dataset), Zephyr-7B (RLHF trained)
- **Benchmark**
  - MT-Bench for multi-turn conversational quality, using GPT-4 as an AI judge
- **Comparisons**
  - Compared MEDUSA-1 and MEDUSA-2 against standard auto-regressive decoding and speculative decoding
- **Metrics**
  - Evaluated for both actual decoding speed (tokens per second) and generation quality (0-10 score)

Speedup Across Models and Tasks

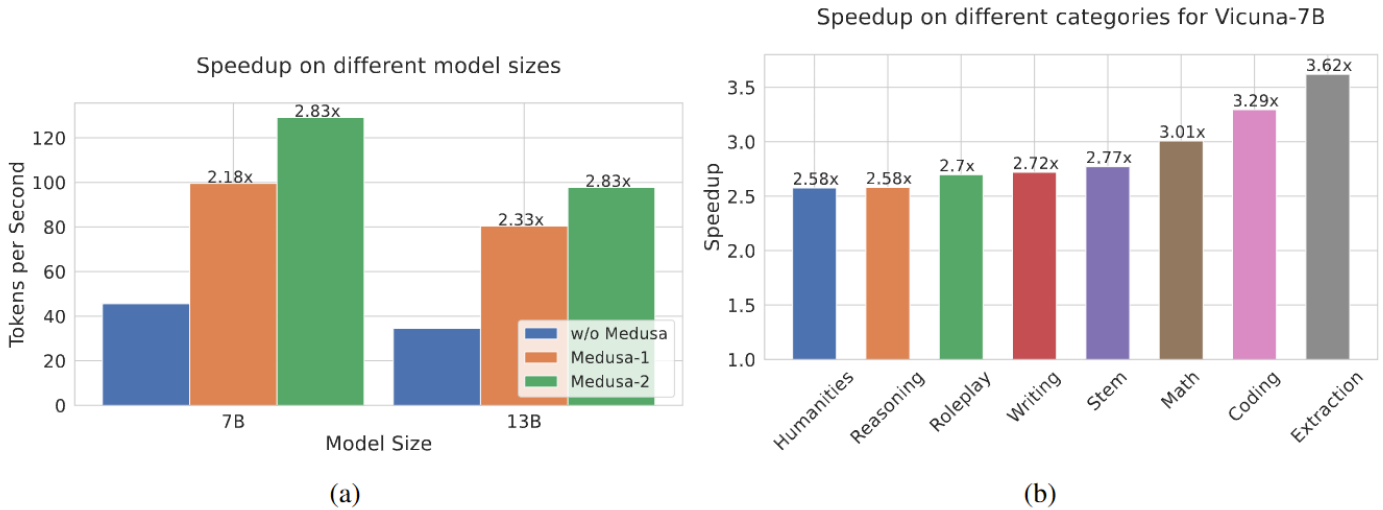


Figure 3. Left: Speed comparison of baseline, MEDUSA-1 and MEDUSA-2 on Vicuna-7B/13B. MEDUSA-1 achieves more than 2× wall-time speedup compared to the baseline implementation while MEDUSA-2 further improves the speedup by a significant margin. Right: Detailed speedup performance of Vicuna-7B with MEDUSA-2 on 8 categories from MT-Bench.

- MEDUSA-1 achieves a lossless speedup of 2.18x and 2.33x on Vicuna-7B/13B
- Using MEDUSA-2 instead, speedup jumps to 2.83x
- Due to MEDUSA-2 joint training, prediction accuracy increases, leading to better speculations
- For task specific tests, highly structured tasks see the biggest speedups

Results Summary

Model Name	Vicuna-7B	Zephyr-7B	Vicuna-13B	Vicuna-33B
Acc. rate	3.47	3.14	3.51	3.01
Overhead	1.22	1.18	1.23	1.27
Quality	6.18 (+0.01)	7.25 (-0.07)	6.43 (-0.14)	7.18 (+0.05)
$S_{\text{SpecDecoding}}$	1.47	-	1.56	1.60
S_{MEDUSA}	2.83	2.66	2.83	2.35

- **Acceleration Rate:** average tokens generated per decoding step
- **Overhead:** extra computation and memory cost from speculation and verification
- **Quality:** fidelity of outputs compared to standard decoding
- **Speedup:** actual wall-clock improvement in tokens/sec or latency
  - Computed by acceleration rate / overhead

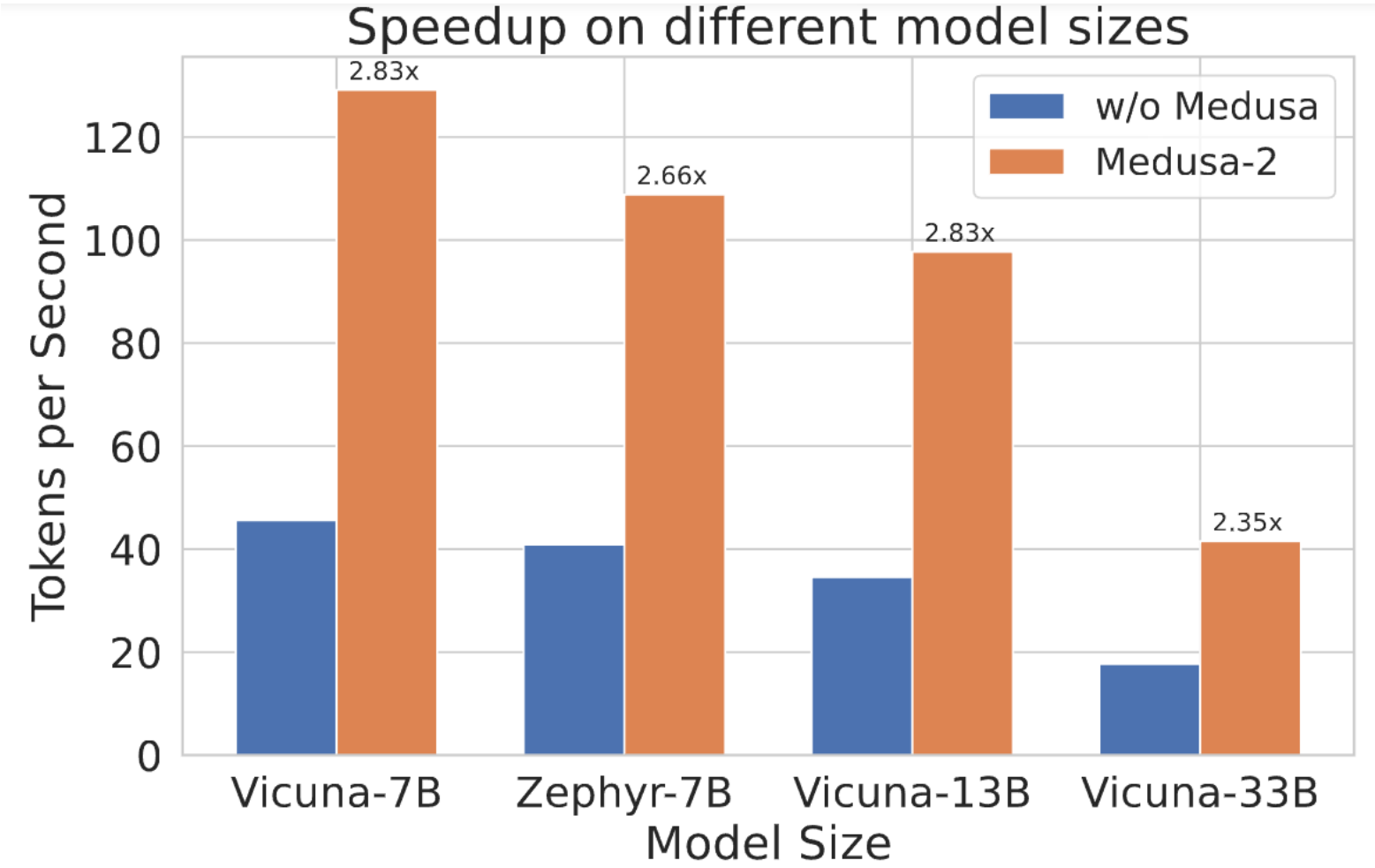
Results Summary

Table 3. Impact of Techniques on Speedup

Technique	Speedup
Medusa-1 heads without tree attention	∼1.5x
Adding tree attention	∼1.9x
Using optimized tree configuration	∼2.2x
Training heads with Medusa-2	∼2.8x

- **Takeaways:**
  - Adding extra heads to predict the future results in a decent speedup baseline
  - Evaluating multiple candidate branches simultaneously increases the number of tokens accepted per step
  - Optimized sparse trees focus on high-value paths, meaning less compute is wasted on unlikely sequences
  - Aligning the base model to support the heads (MEDUSA-2) maximizes predictive accuracy and overall acceleration

Results: Speedup



Limitations and Future Directions

Limitations

- Accuracy degrades for later Medusa heads due to missing context
- More heads give diminishing speedup gains
- Single hidden-state guessing hurts long-range coherence
- MEDUSA-2 joint training risks degrading base model quality

Future Directions

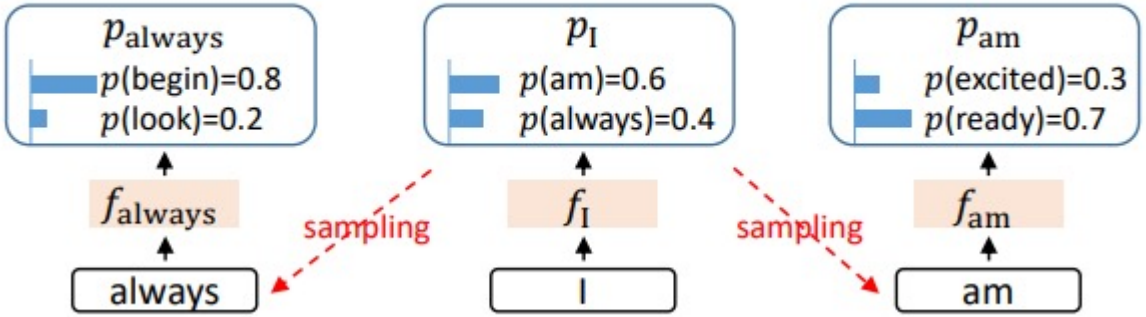
- Supply extra context beyond the single last hidden state to reduce uncertainty and improve the accuracy of deeper speculations
- Optimize for Large Batch Sizes
- Improve the attention mechanism to keep processing speeds high even on very long documents

EAGLE: speculative sampling requires rethinking feature uncertainty

ICML 2024

Problem and Motivation

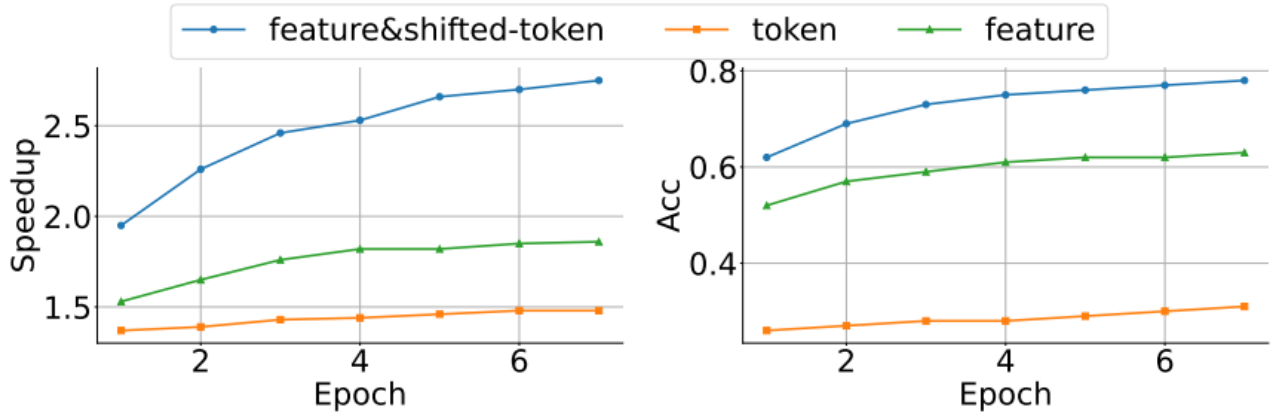
- **Problem with Current Solutions:**
  - Difficulty in finding sufficiently small models to act as draft models for certain target LLMs that still mirror the larger model
- **Problem with Medusa**
  - Not truly auto-regressive, later tokens don't have the context of earlier tokens, lowering accuracy for later tokens proposed by draft model



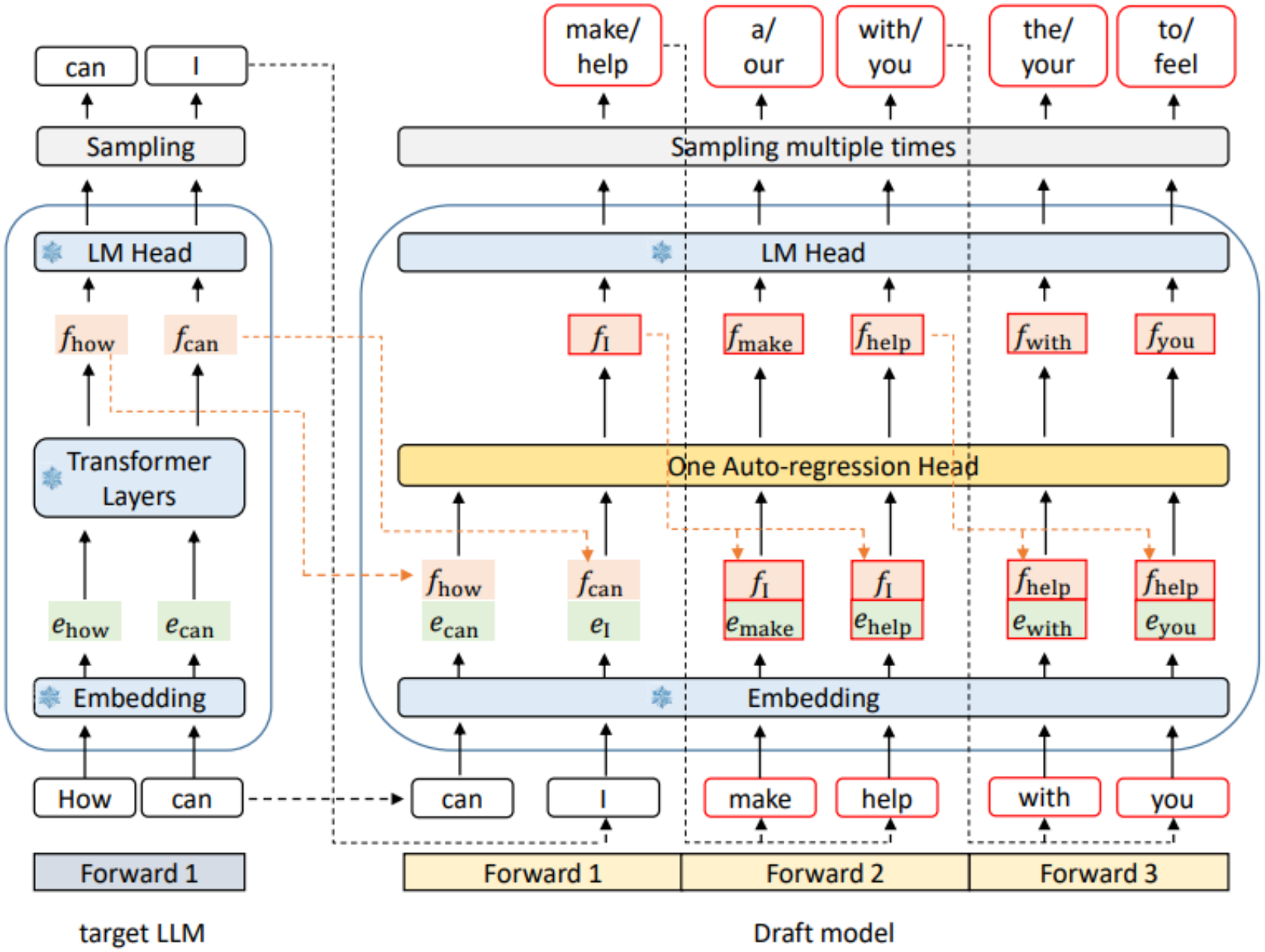
- **Question:** Can we increase the accuracy of the draft model in order to have more tokens accepted during each target model step?

Solution

- **Autoregression at the feature level is simpler than the token level:**
  - When choosing a token you collapse the probability distribution, and possibly diverge away from what the target would want
- **Providing the next token as an embedding to reduce ambiguity:**
  - Feature level distributions have uncertainty in what token should be sampled
  - Provide word embeddings of the sampled token in order to reduce ambiguity in finding the next feature level representation



Drafting Phase



Key

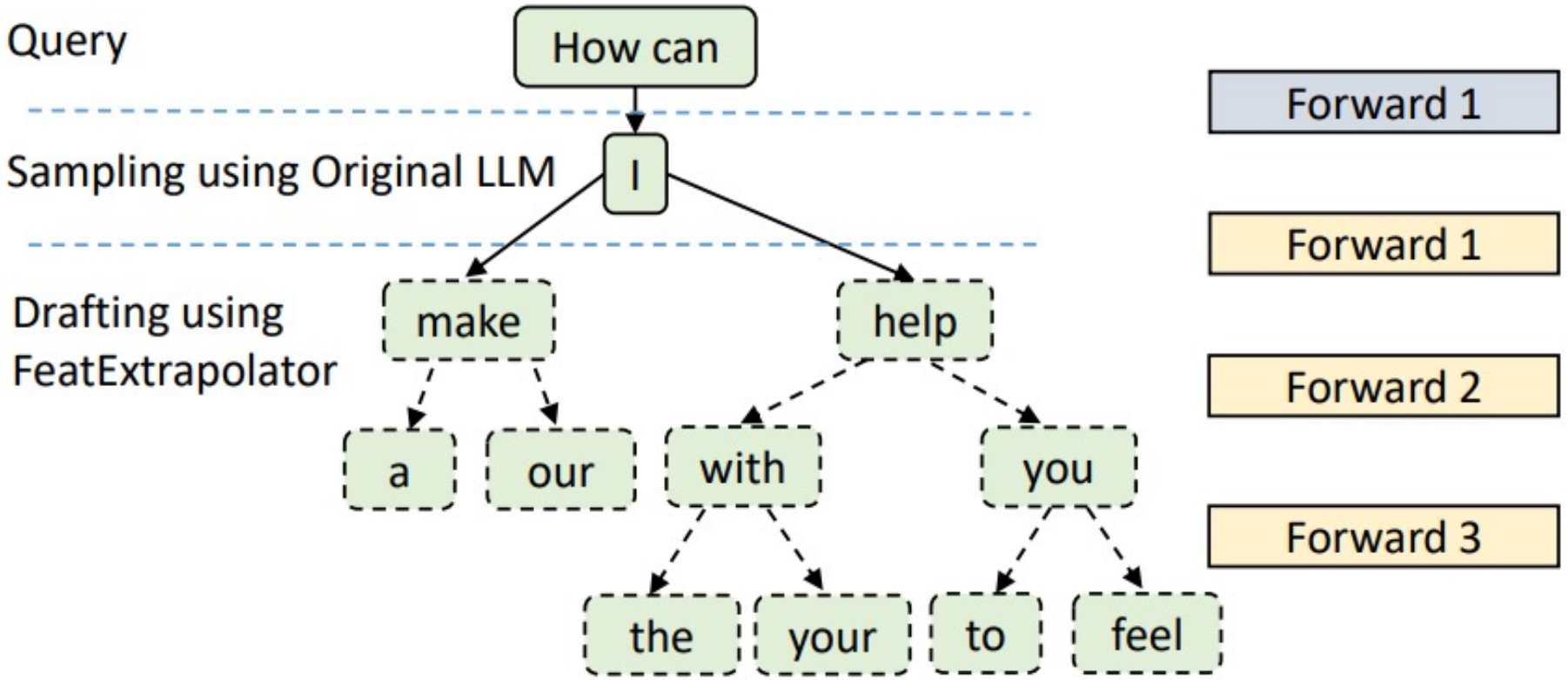
Green->token embedding

Orange->features

Red box -> predictions

Blue boxes -> taken from target model

Drafting Phase Result



Training

- **Embedding and LM Head are reused, no training required**
- **Training Auto Regressive Head:**
  - Ideally trained with generated text from the target LLM this has high cost
  - Ablation Study finds that sensitivity is low and a fixed dataset can achieve solid results with lower overhead
  - Trains against feature representations from target model, adds random noise to prevent inaccuracies from the target model

$$L_{reg} = \text{Smooth L1}(f_{i+1}, \text{Draft\_Model}(T_{2:i+1}, F_{1:i})).$$

$$p_{i+2} = \text{Softmax}(\text{LM\_Head}(f_{i+1})),$$

$$\hat{p}_{i+2} = \text{Softmax}(\text{LM\_Head}(\hat{f}_{i+1})),$$

$$L_{cls} = \text{Cross\_Entropy}(p_{i+2}, \hat{p}_{i+2}).$$

$$L = L_{reg} + w_{cls} L_{cls}.$$

Training data	Speedup	τ
Fixed dataset	2.78x	3.62
Data generated by target LLM	2.88x	3.75

Verification

- **Tree Attention**
  - Leveraging tree attention, the target computes probability of each token in the tree through a single forward pass
  - It verifies the distribution given by the draft model and ensures it matches with the target
  - Concurrently document accepted tokens and their features to leverage in next drafting phase

Experiments

- **Models Used**
  - Vicuna (7B, 13B and 33B)
  - LLaMA2-chat (7B, 13B and 70B)
  - Mixtral 8x7B Instruct
- **Tasks**
  - Multi-turn dialogue, code generation, mathematical reasoning nad instruction following
- **Datasets**
  - Used MT-Bench, HumanEval, GSM8K and Alpaca datasets

Experiment Metrics and Training

- **Metrics**
  - Raw speedup ratio -> How much faster is Eagle relative to base model?
  - Average Acceptance Length -> How many tokens were accepted by target?
  - Acceptance rate -> What percent of tokens were accepted?
- **Tasks**
  - Multi-turn dialogue, code generation, mathematical reasoning nad instruction following
- **Datasets**
  - Used MT-Bench, HumanEval, GSM8K and Alpaca datasets
- **Draft Model Sizes and Training time**
  - 7B -> .24B, 13B -> .37B, 33B -> .56B, 70B -> .99B, the autoregression head is trainable within 1-2 days on an A100 40G server for the 70B model

Results

		HumanEval		GSM8K		Alpaca	
	Model	Speedup	τ	Speedup	τ	Speedup	τ
T=0	V 7B	3.33x	4.29	3.01x	4.00	2.79x	3.86
	V13B	3.58x	4.39	3.08x	3.97	3.03x	3.95
	V 33B	3.67x	4.28	3.25x	3.94	2.97x	3.61
	LC 7B	3.17x	4.24	2.91x	3.82	2.78x	3.71
	LC 13B	3.76x	4.52	3.20x	4.03	3.01x	3.83
	LC 70B	3.52x	4.42	3.03x	3.93	2.97x	3.77
T=1	V 7B	2.39x	3.43	2.34x	3.29	2.21x	3.30
	V13B	2.65x	3.63	2.57x	3.60	2.45x	3.57
	V 33B	2.76x	3.62	2.77x	3.60	2.52x	3.32
	LC 7B	2.61x	3.79	2.40x	3.52	2.29x	3.33
	LC 13B	2.89x	3.78	2.82x	3.67	2.66x	3.55
	LC 70B	2.92x	3.76	2.74x	3.58	2.65x	3.47

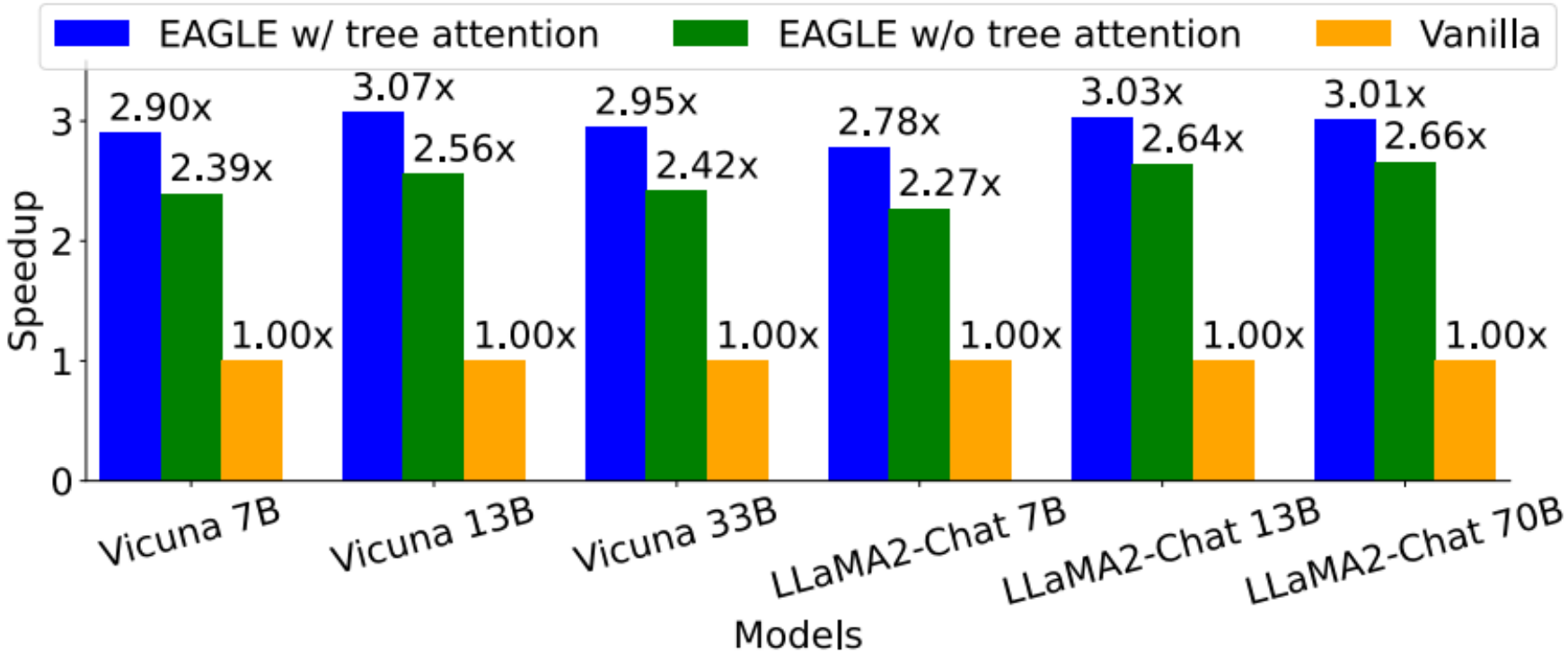
Results on MT-Bench

	Model	τ	0-α	1-α	2-α	3-α	4-α
T=0	Vicuna 7B	3.94	0.79	0.74	0.72	0.73	0.67
	Vicuna 13B	3.98	0.79	0.74	0.72	0.74	0.70
	Vicuna 33B	3.68	0.74	0.69	0.67	0.67	0.66
	LLaMA2-Chat 7B	3.62	0.76	0.69	0.67	0.68	0.68
	LLaMA2-Chat 13B	3.90	0.77	0.69	0.69	0.70	0.71
	LLaMA2-Chat 70B	3.81	0.75	0.69	0.65	0.64	0.64
T=1	Vicuna 7B	3.17	0.71	0.68	0.66	0.66	0.65
	Vicuna 13B	3.20	0.73	0.68	0.68	0.67	0.69
	Vicuna 33B	3.22	0.71	0.67	0.64	0.64	0.64
	LLaMA2-Chat 7B	3.30	0.71	0.66	0.66	0.66	0.64
	LLaMA2-Chat 13B	3.45	0.73	0.69	0.66	0.67	0.67
	LLaMA2-Chat 70B	3.46	0.73	0.67	0.64	0.66	0.65

Mixtral 8x7B,
temperature = 0

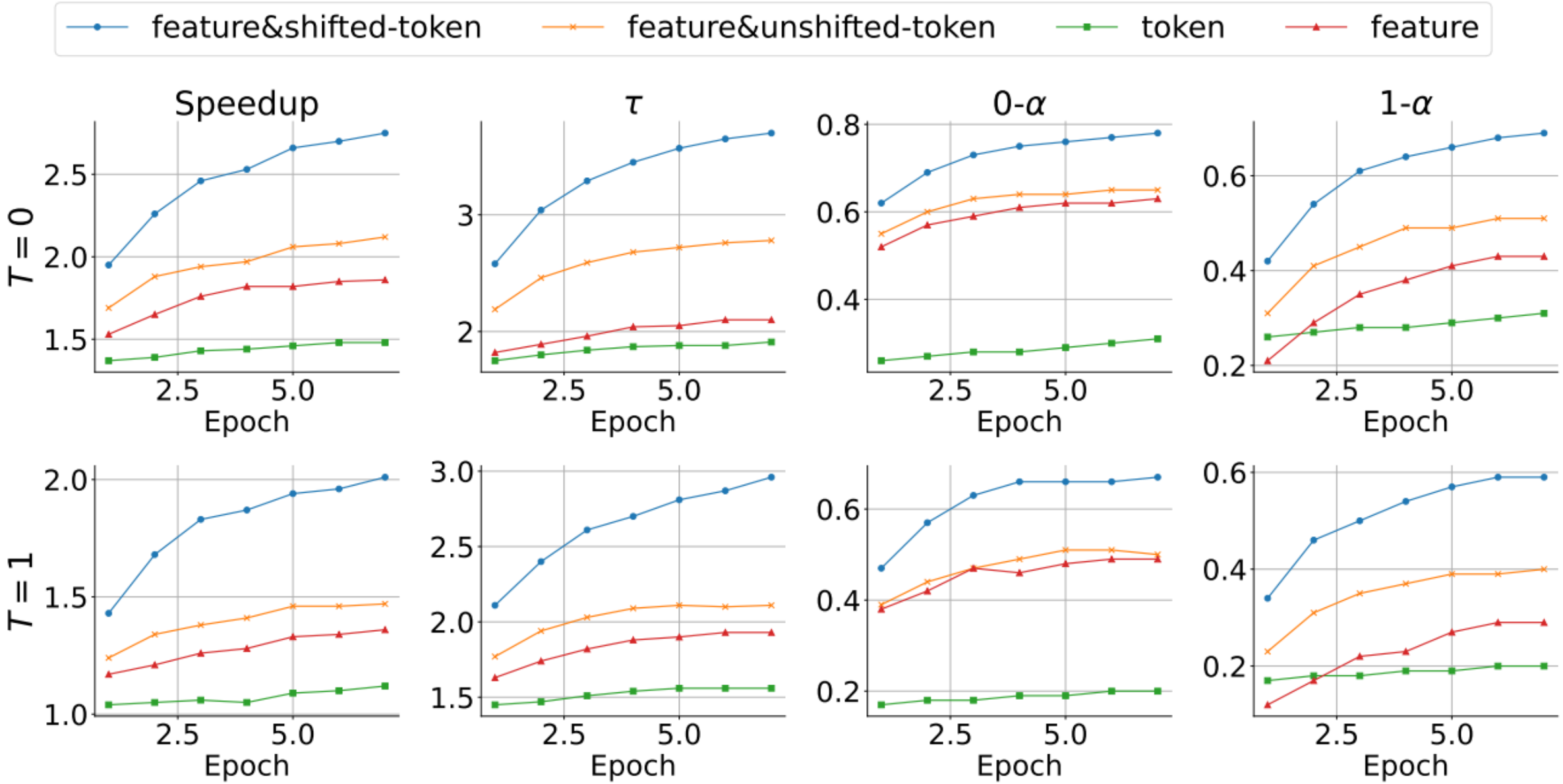
Speedup	τ	0-α	1-α	2-α	3-α	4-α
1.50x	3.25	0.67	0.62	0.61	0.64	0.63

Results Tree vs Non-Tree Attention



Vicuna			LLaMA2-Chat		
Size	Chain	Tree	Size	Chain	Tree
7B	3.20	3.94 (+0.74)	7B	3.00	3.62 (+0.62)
13B	3.23	3.98 (+0.75)	13B	3.18	3.90 (+0.68)
33B	2.97	3.68 (+0.71)	70B	3.12	3.81 (+0.69)

Comparing Draft Model Input



Batch Size Comparisons

Batch size	1	2	3	4	Throughput
Vicuna 7B	2.90x	2.87x	2.65x	2.76x	1.97x
LLaMA2-Chat 70B	3.01x	2.81x	2.50x	2.40x	1.99x

Speedup ratios at different batch sizes. MT-Bench dataset, temperature = 0

Limitations and Future Directions

- **Limitations**
  - Certain Tasks enjoy higher temperature, EAGLE may perform poorly at those
  - LM Head trained on target model features, as draft features drift away accuracy lowers due to unexpected feature representations
  - Must be retrained for different models
  - Model Size/Type, As models get larger their feature representations may get more complex, will EAGLE still work?
- **Future Directions**
  - Testing with Target Models larger than 70B
  - Target Model Architecture Changes --> Train LLM alongside feature predictor to more effectively use LLM hidden features
  - Add branch confidence and stop branches with low confidence early

Conclusion

Approaches	Speculative Decoding	MEDUSA	EAGLE
System Architecture	Target model + draft model	LLM + added heads	LLM + lightweight engine
Drafting Mechanism	Draft model generates draft tokens	Multiple heads predict tokens simultaneously	Predicts next-step hidden states
Training Requirement	Optional	Mandatory	Mandatory
Key Strength	Flexible, no training required, work with any draft models	Enables speculative decoding without a second draft model	Increases accuracy of predictions via previous features and shifted token
Use Case	When have a ready-to-use, strong small model, and do not want additional training	When training is allowed and you want speedups without an additional model	When training is allowed and you can afford to train a separate draft model