Speculative Decoding and Efficient Inference

Amin Saberi

MS&E 319

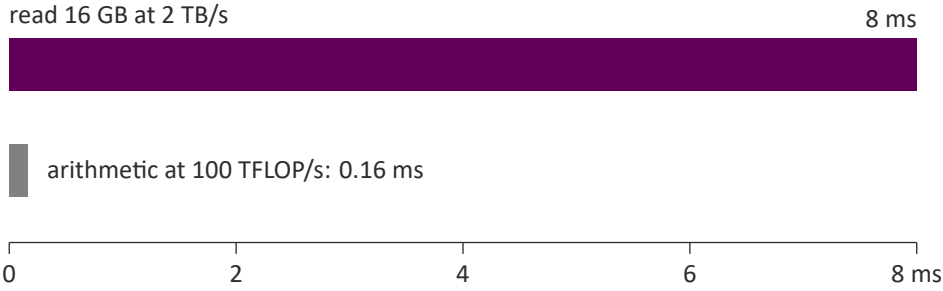
Autumn 2026



One Token per Pass

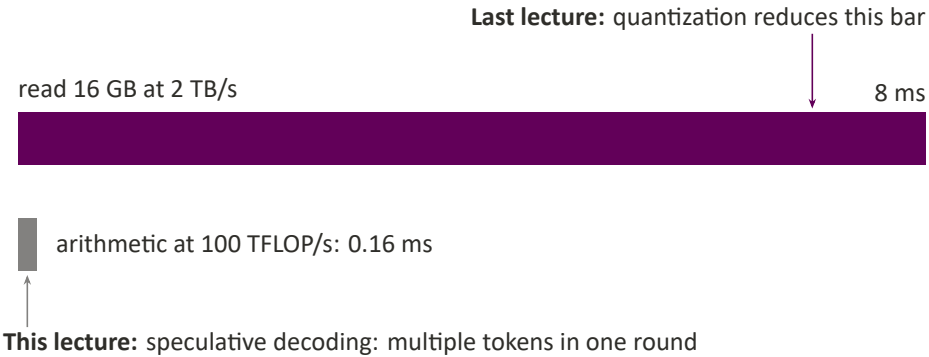
An 8-billion-weight model at 2 bytes per weight holds 16 GB.

Each decoding step reads all of it to produce one token.



The arithmetic units mostly wait.

One Token per Pass



Part I:
The Computational Opportunity



Sampling Waits; Checking Does Not

Target: the large model; $p(\omega \mid \text{prefix})$ is its probability that the next token is $\omega \in \mathcal{V}$.

Generating

$$Y_1 \sim p(\cdot \mid \text{prefix})$$

wait for Y_1

$$Y_2 \sim p(\cdot \mid \text{prefix}, Y_1)$$

wait for Y_2

$$Y_3 \sim p(\cdot \mid \text{prefix}, Y_1, Y_2)$$



Sampling Waits; Checking Does Not

Target: the large model; $p(\omega \mid \text{prefix})$ is its probability that the next token is $\omega \in \mathcal{V}$.

Generating

$$Y_1 \sim p(\cdot \mid \text{prefix})$$

wait for Y_1

$$Y_2 \sim p(\cdot \mid \text{prefix}, Y_1)$$

wait for Y_2

$$Y_3 \sim p(\cdot \mid \text{prefix}, Y_1, Y_2)$$

Checking a supplied A, B, C

$$p(\cdot \mid \text{prefix})$$

$$p(\cdot \mid \text{prefix}, \text{A})$$

$$p(\cdot \mid \text{prefix}, \text{A}, \text{B})$$

$$p(\cdot \mid \text{prefix}, \text{A}, \text{B}, \text{C})$$

one pass



Sampling Waits; Checking Does Not

Target: the large model; $p(\omega \mid \text{prefix})$ is its probability that the next token is $\omega \in \mathcal{V}$.

Generating

$$Y_1 \sim p(\cdot \mid \text{prefix})$$

wait for Y_1

$$Y_2 \sim p(\cdot \mid \text{prefix}, Y_1)$$

wait for Y_2

$$Y_3 \sim p(\cdot \mid \text{prefix}, Y_1, Y_2)$$

Checking a supplied A, B, C

$$p(\cdot \mid \text{prefix})$$
$$p(\cdot \mid \text{prefix}, \text{A})$$
$$p(\cdot \mid \text{prefix}, \text{A}, \text{B})$$
$$p(\cdot \mid \text{prefix}, \text{A}, \text{B}, \text{C})$$

one pass

Supplied tokens: every conditional in one pass.



A Cheap Drafter Guesses Ahead

Drafter: a cheaper predictor with next-token distribution $q(\omega \mid \text{prefix})$; its guesses are the **draft**.

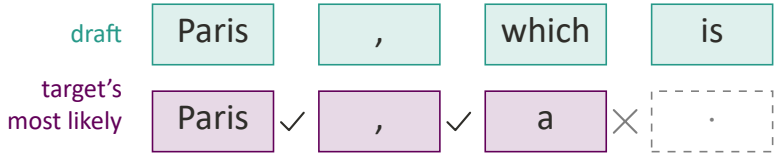


The target checks the whole draft in one pass.



Greedy: Keep the Matching Prefix

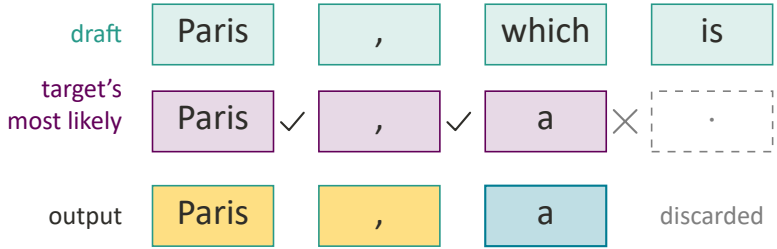
Greedy decoding: the target always emits its most likely token.





Greedy: Keep the Matching Prefix

Greedy decoding: the target always emits its most likely token.

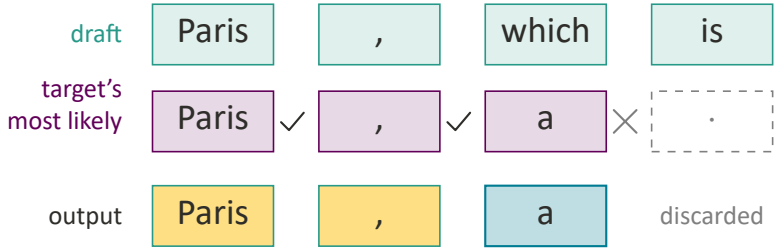


Keep the draft up to the first mismatch, emit the target's token there, discard the rest.



Greedy: Keep the Matching Prefix

Greedy decoding: the target always emits its most likely token.



Keep the draft up to the first mismatch, emit the target's token there, discard the rest.

The problem with greedy: it finds the most likely sample, but does not reproduce the exact distribution.



Part II:
Exact Sampling from a Guess



A Sampling Puzzle

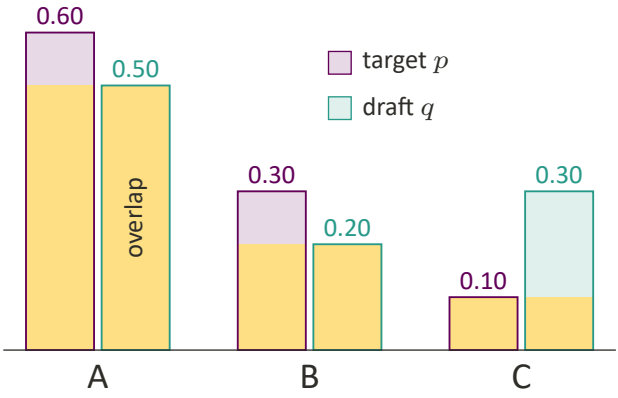
The drafter proposes a token from q. We may keep it or replace it; the output must follow p.

Token	Target p	Draft q	Proposed in 100	Wanted in 100
A	0.60	0.50	50	60
B	0.30	0.20	20	30
C	0.10	0.30	30	10

Which proposals can we keep, and what fraction of each, to keep as many as possible?



Keep All A and B, a Third of C



Keep every A and every B:
there are fewer
than wanted.

Keep one C in three:
10 of 30.

Kept mass: $0.50 + 0.20 + 0.10 = 0.80$.



The Acceptance Rule

Leviathan et al. (2023); Chen et al. (2023)

Keep a proposal ω with probability $\beta(\omega)$:

$$\beta(\omega) := \min\left\{1, \frac{p(\omega)}{q(\omega)}\right\},$$



The Acceptance Rule

Leviathan et al. (2023); Chen et al. (2023)

Keep a proposal ω with probability $\beta(\omega)$:

$$\beta(\omega) := \min\left\{1, \frac{p(\omega)}{q(\omega)}\right\}, \qquad \alpha := \sum_{\omega} \min\{p(\omega), q(\omega)\}.$$

The overlap α is the probability of acceptance.



The Acceptance Rule

Leviathan et al. (2023); Chen et al. (2023)

Keep a proposal ω with probability $\beta(\omega)$:

$$\beta(\omega) := \min\left\{1, \frac{p(\omega)}{q(\omega)}\right\}, \qquad \alpha := \sum_{\omega} \min\{p(\omega), q(\omega)\}.$$

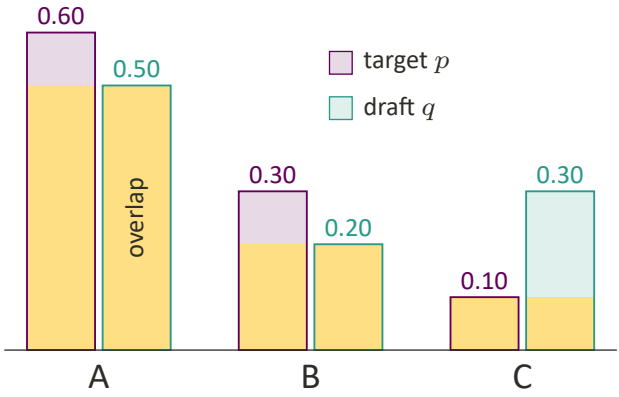
The overlap α is the probability of acceptance.

Note: $\alpha = 1 - \mathrm{TV}(p, q)$ (total variation distance).

$$\mathrm{TV}(p, q) := \frac{1}{2} \sum_{\omega} |p(\omega) - q(\omega)|.$$



What Is Still Missing?



Kept: $(0.50, 0.20, 0.10)$, total 0.80.

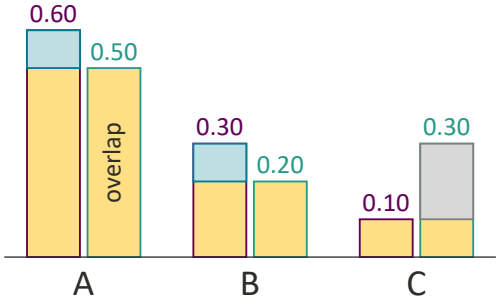
Wanted: $(0.60, 0.30, 0.10)$.

What distribution to sample from in case of rejection to match the target distribution?



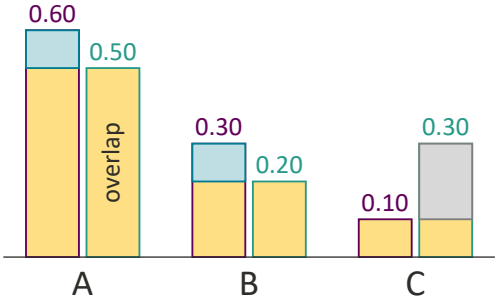
The Residual Fills the Deficit

Missing: $(0.10, 0.10, 0)$, in total $0.20 = 1 - \alpha$.





The Residual Fills the Deficit



Missing: $(0.10,\ 0.10,\ 0)$, in total $0.20 = 1 - \alpha$.

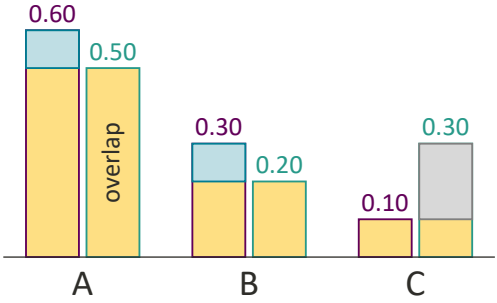
Residual, with $[\,\cdot\,]_+ := \max\{\cdot, 0\}$:

$$p_{\text{res}}(\omega) := \frac{[p(\omega) - q(\omega)]_+}{1 - \alpha}\,.$$

$$p_{\text{res}} = (0.10,\ 0.10,\ 0)/0.20\,.$$



The Residual Fills the Deficit



Missing: $(0.10,\ 0.10,\ 0)$, in total $0.20 = 1 - \alpha$.

Residual, with $[\,\cdot\,]_+ := \max\{\cdot, 0\}$:

$$p_{\text{res}}(\omega) := \frac{[p(\omega) - q(\omega)]_+}{1 - \alpha}\,.$$

$$p_{\text{res}} = (0.10,\ 0.10,\ 0)/0.20\,.$$

Accepted proposals supply the overlap; residual draws supply the deficit.



Output Distribution Matches the Target

Claim. Draw a proposal from q; keep it with probability β, otherwise draw from p_{res}. The output has law p.



Output Distribution Matches the Target

Claim. Draw a proposal from q; keep it with probability β, otherwise draw from p_{res}. The output has law p.

Proof. Two disjoint ways to output ω: propose and keep it, or reject and draw it.

$$\mathbb{P}(\text{reject}) = \sum_{\omega} \big(q(\omega) - \min\{p(\omega), q(\omega)\}\big) = 1 - \alpha\,,$$



Output Distribution Matches the Target

Claim. Draw a proposal from q; keep it with probability β, otherwise draw from p_{res}. The output has law p.

Proof. Two disjoint ways to output ω: propose and keep it, or reject and draw it.

$$\mathbb{P}(\text{reject}) = \sum_{\omega} \big(q(\omega) - \min\{p(\omega), q(\omega)\}\big) = 1 - \alpha\,,$$

$$\begin{aligned}\mathbb{P}(\text{output} = \omega) &= \min\{p(\omega), q(\omega)\} + (1-\alpha)\, p_{\text{res}}(\omega) \\ &= \min\{p(\omega), q(\omega)\} + [p(\omega) - q(\omega)]_+ = p(\omega)\,.\end{aligned}$$

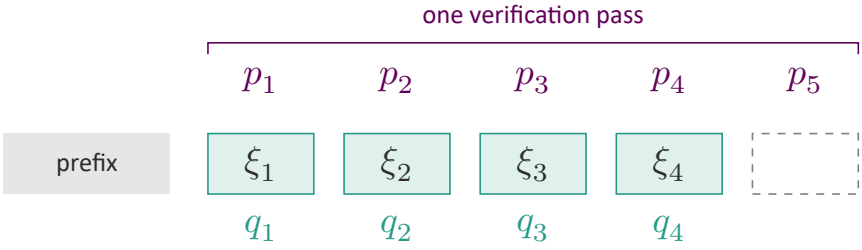


Part III:
The Block Algorithm



The Objects of a Round

Draft length K: the drafter samples $\xi_1, \dots, \xi_K$ one at a time.



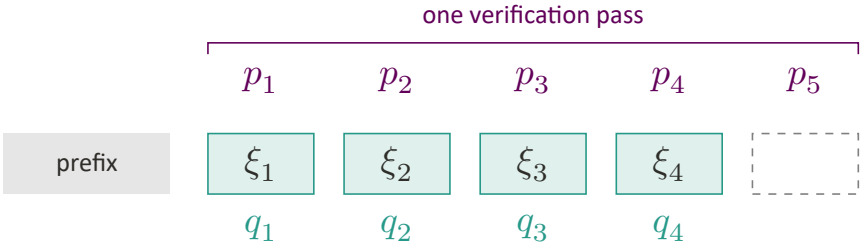
$$q_i := q(\cdot \mid \text{prefix}, \xi_{<i}), \qquad p_i := p(\cdot \mid \text{prefix}, \xi_{<i}).$$

One verification pass over the draft gives $p_1, \dots, p_{K+1}$.



Assemble a Round

You have the draft, both conditionals q_i and p_i at each position, and the one-token rule.



Put them together into one round. What should the round return?



One Speculative Round

1 Input: $\xi_1, \ldots, \xi_K$ from the draft; $p_1, \ldots, p_{K+1}$ from verification.



One Speculative Round

1. Input: $\xi_1, \dots, \xi_K$ from the draft; $p_1, \dots, p_{K+1}$ from verification.

2. For $i = 1, \dots, K$:

   draw $U \sim \text{Uniform}[0,1]$; if $U \leq \min\{1, p_i(\xi_i)/q_i(\xi_i)\}$, accept ξ_i,

   else draw ζ from the residual of p_i, q_i and return $\xi_{<i}, \zeta$.



One Speculative Round

1. Input: $\xi_1, \dots, \xi_K$ from the draft; $p_1, \dots, p_{K+1}$ from verification.

2. For $i = 1, \dots, K$:

   draw $U \sim \text{Uniform}[0,1]$; if $U \leq \min\{1, p_i(\xi_i)/q_i(\xi_i)\}$, accept ξ_i,

   else draw ζ from the residual of p_i, q_i and return $\xi_{<i}, \zeta$.

3. All accepted: draw the bonus token $\zeta \sim p_{K+1}$; return $\xi_1, \dots, \xi_K, \zeta$.



Three of the Possible Outcomes

Verification result	Committed output
Reject the first proposal	ζ
Accept A, B; reject C	A, B, ζ
Accept all four	A, B, C, D, ζ

A round commits between 1 and $K + 1$ tokens, never zero.



Why the Round Stops

Why stop after the first rejection, or after the bonus token?



Why the Round Stops

Why stop after the first rejection, or after the bonus token?

After a rejection, the later conditionals assumed the rejected token.

The replacement never equals it: rejection needs $q_i > p_i$ there, the residual needs $p_i > q_i$.

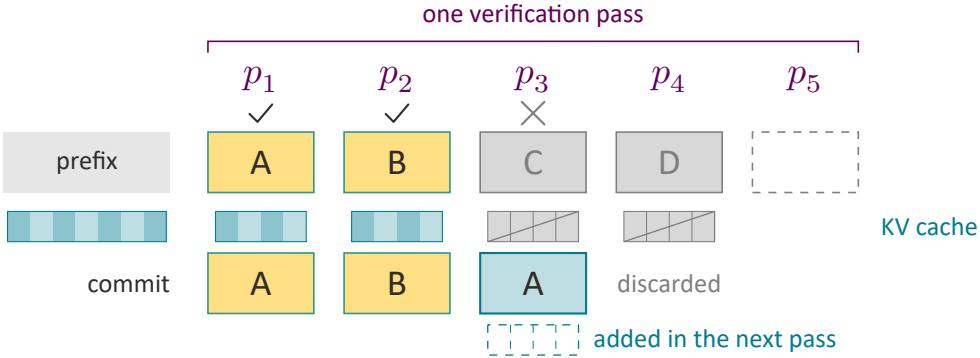
After the bonus token, its next conditional is not computed yet.



The KV Cache After a Rejection



The KV Cache After a Rejection



Keep the cache of the accepted prefix; discard that of C and D; the fresh token's entries come with the next pass.

The Whole Sequence Is Exact

Theorem. With exact arithmetic and exact sampling, the concatenated rounds generate from the target: for every continuation $\omega_1, \dots, \omega_{n_{\text{out}}}$,

$$\mathbb{P}(\omega_1, \dots, \omega_{n_{\text{out}}} \mid \text{prefix}) = \prod_{t=1}^{n_{\text{out}}} p(\omega_t \mid \text{prefix}, \omega_{<t}) \,.$$

Proof idea. The chain rule and induction.

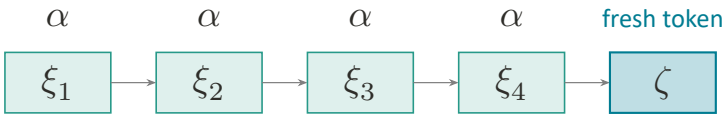


Part IV:
How Much Does a Round Buy?



Tokens per Round

Assume: each reached proposal is accepted with probability α.



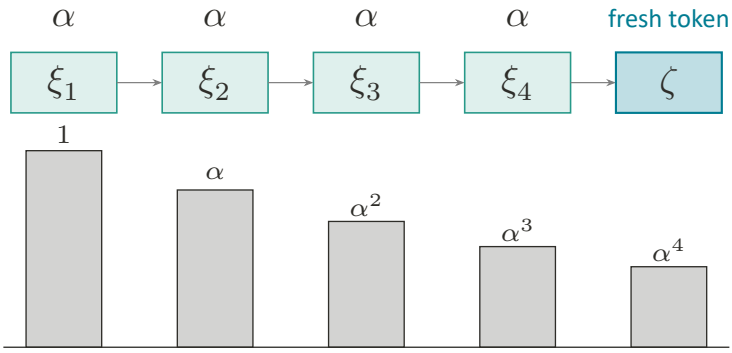
reached: every earlier proposal of the round was accepted

With K proposals, how many tokens does a round commit on average? Count the fresh token.



The Tail-Sum Answer

$$\mathbb{E}[\text{tokens per round}] = 1 + \alpha + \cdots + \alpha^K = \frac{1 - \alpha^{K+1}}{1 - \alpha}.$$



probability that the round fills this position: every earlier proposal accepted



From Tokens to Time

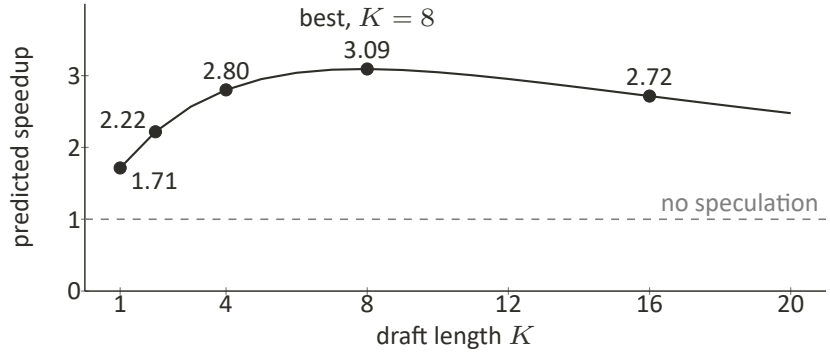
one ordinary target step

$$\text{speedup} \approx \frac{\text{step time} \times \mathbb{E}[\text{tokens per round}]}{\text{draft time} + \text{verification time} + \text{other overhead}}.$$



Longer Drafts, Diminishing Returns

Assumed: $\alpha = 0.8$; drafting 0.05 step per token; verification one step.



Each extra proposal adds only α^{K+1} tokens, while drafting cost keeps growing.



When Speculation Pays

One request: arithmetic waits on weight traffic, and speculation can use it.

Many requests: batching, serving them together, already reuses W; speculation competes for compute and memory.

Long contexts: every candidate position attends to the whole KV cache, and that traffic can dominate even with perfect weight reuse.

The gain depends on the regime, not only on acceptance.



Optimality (for One Token)

A coupling of q and p: any joint construction of a proposal with law q and an output with law p.

Theorem. For every coupling, $\mathbb{P}(\text{proposal} = \text{output}) \leq \alpha$. Acceptance with the residual attains α.



Optimality (for One Token)

A coupling of q and p: any joint construction of a proposal with law q and an output with law p.

Theorem. For every coupling, $\mathbb{P}(\text{proposal} = \text{output}) \leq \alpha$. Acceptance with the residual attains α.

Proof. $\mathbb{P}(\text{proposal} = \text{output} = \omega) \leq \min\{q(\omega), p(\omega)\}$; sum over ω. On rejection, the residual never returns the rejected token.



Optimal per Token, Not per Block

Sun et al. (2025)

The theorem covers one token. And a whole draft?

Example: tokens A, B; at every position $p = (\frac{1}{3}, \frac{2}{3})$, $q = (\frac{2}{3}, \frac{1}{3})$; drafts of length 2.

Tokenwise: $\alpha = \frac{2}{3}$; accepted draft tokens per round: $\alpha + \alpha^2 = \frac{10}{9}$.



Optimal per Token, Not per Block

Sun et al. (2025)

The theorem covers one token. And a whole draft?

Example: tokens A, B; at every position $p = (\frac{1}{3}, \frac{2}{3})$, $q = (\frac{2}{3}, \frac{1}{3})$; drafts of length 2.

Tokenwise: $\alpha = \frac{2}{3}$; accepted draft tokens per round: $\alpha + \alpha^2 = \frac{10}{9}$.

The draft A B is equally likely under q and p ($\frac{2}{9}$), yet tokenwise keeps A only half the time.

Can a verifier that judges the whole draft do better?



Judging the Draft Jointly

Draft	q(draft)	p(draft)	Tokenwise	Block
A A	4/9	1/9	3/4	1/2
A B	2/9	2/9	1	2
B A	2/9	2/9	3/2	3/2
B B	1/9	4/9	2	2
Average over q			10/9	11/9

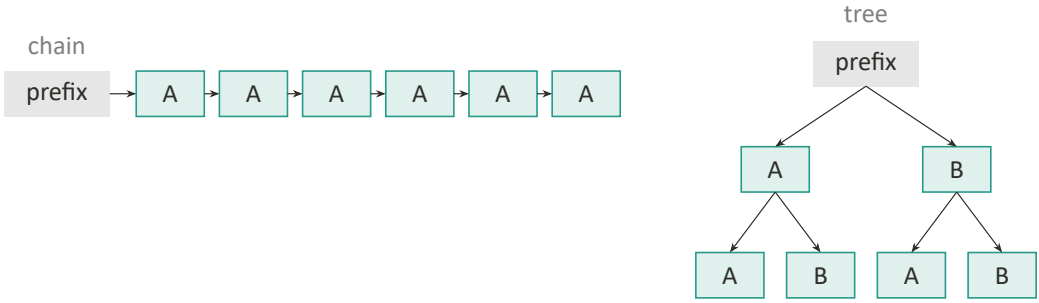
Block verification keeps A B whole and pays for it on A A; the output still has law p.

Joint decisions can use a draft better than tokenwise ones.



One Chain or Many Futures

A chain bets on one continuation: an early mismatch wastes the rest.
A tree keeps alternatives: each node is a candidate prefix.



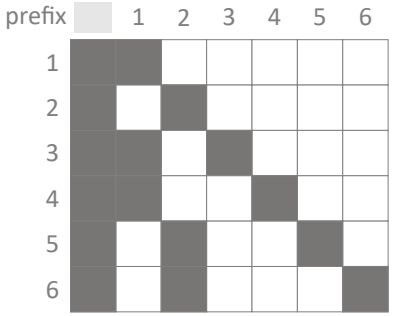
How can the target check every node in one pass?



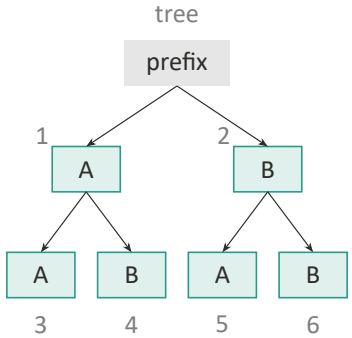
Tree Attention

To check a tree in one pass, the verifier uses **tree attention**: its own attention, with a tree mask.

Each node attends to the prefix, its ancestors and itself; never to another branch.

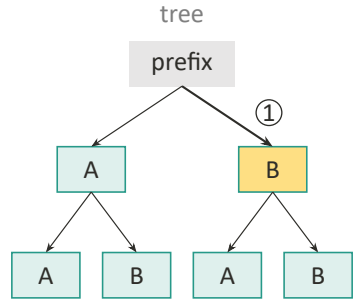


rows: nodes;
columns: what each may see



An Exact Sampler on a Fixed Tree

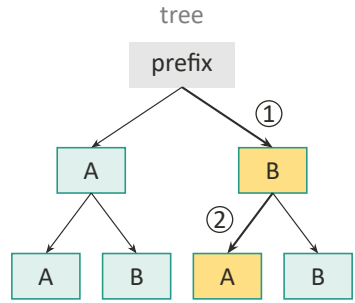
① At the current node, draw a token from the target.



An Exact Sampler on a Fixed Tree

① At the current node, draw a token from the target.

② If it is a child, emit it and move there; repeat.





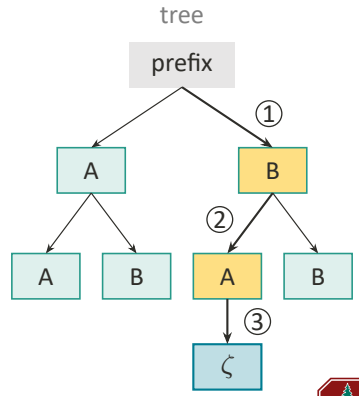
An Exact Sampler on a Fixed Tree

① At the current node, draw a token from the target.

② If it is a child, emit it and move there; repeat.

③ Otherwise emit it and stop; at a leaf, draw once more and stop.

Exact: every token is drawn from the target at its actual history.



Finding the Best Tree

Chen et al. (2024), Sequoia

Each accepted node adds a token; one fresh token ends the round:

$$\mathbb{E}[\text{tokens per round}] = 1 + \sum_{\text{nodes}} \mathbb{P}(\text{the node is accepted}) .$$

Assume the k-th child of an accepted node is accepted with probability ψ_k, whatever the context. With $V(n, b)$ the best $\mathbb{E}[\text{tokens per round}]$ of n nodes, root included, whose root has b children, and $V(1, 0) = 1$:

$$V(n, b) = \max_{1 \le m < n} \left[V(n - m,\, b - 1) + \psi_b \max_j V(m, j) \right] .$$

Offline dynamic programming; optimal under the assumption.

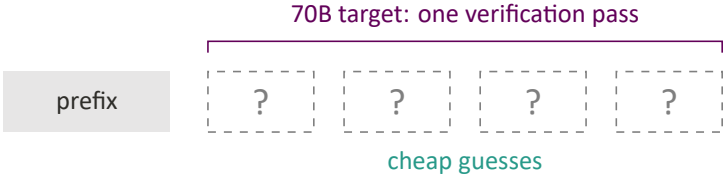


Part V:
Where the Draft Comes From



Where Could a Guess Come From?

You serve a 70-billion-weight model; the verifier of Part III keeps its law exactly.

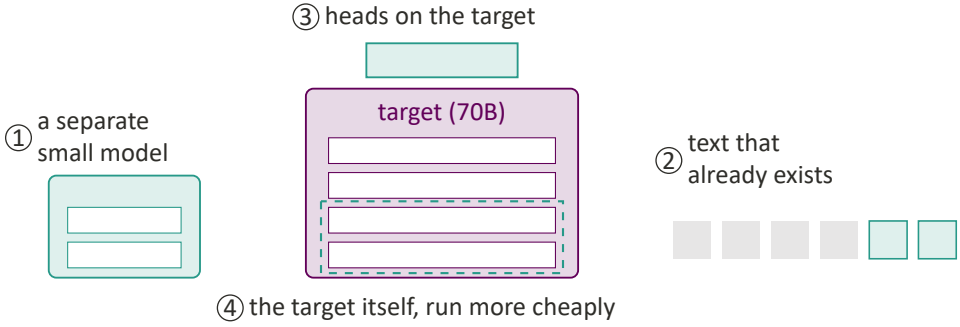


Where could cheap guesses of its next tokens come from?



Four Places to Find a Guess

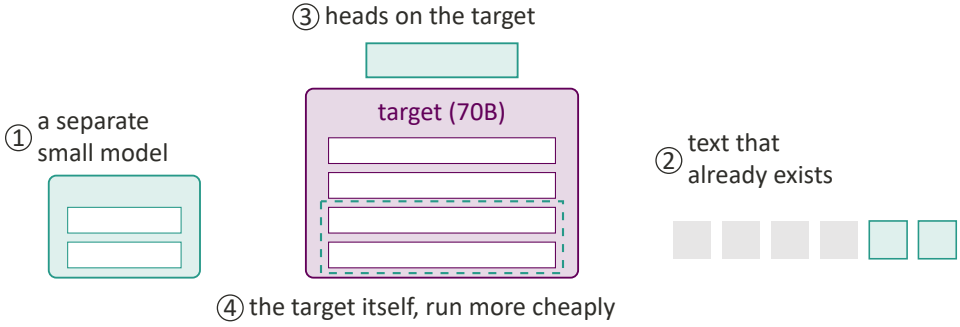
Any guess is allowed: the verifier needs only the law q it was drawn from. A fixed guess A is accepted with probability $p(\mathrm{A})$.





Four Places to Find a Guess

Any guess is allowed: the verifier needs only the law q it was drawn from. A fixed guess A is accepted with probability $p(\mathrm{A})$.



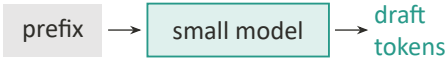
The four are not equal, and not equally supported.



Sources of Draft Tokens

A draft can come from a model or from text we already have.

Smaller language model



T5-small drafts for T5-XXL
4B drafts for 70B

Reuse a compatible model.
Extra weights, cache, and draft time.

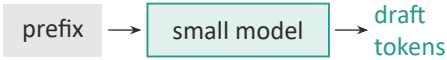
Train a draft model to match the target: DistillSpec (Zhou et al., 2024); online adaptation (Liu et al., 2024).



Sources of Draft Tokens

A draft can come from a model or from text we already have.

Smaller language model

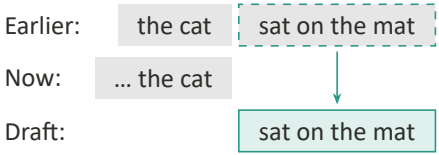


T5-small drafts for T5-XXL
4B drafts for 70B

Reuse a compatible model.
Extra weights, cache, and draft time.

Lookup in existing text

Prompt, previous output, or corpus



No draft model to run.
Works best when text repeats.

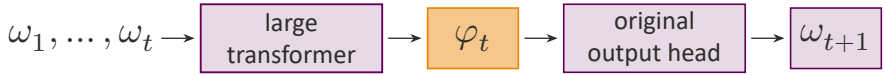
Train a draft model to match the target: DistillSpec (Zhou et al., 2024); online adaptation (Liu et al., 2024).



Medusa: Several Heads, One Hidden State

Cai et al. (2024)

Why another model, when the target already has clues about future tokens?

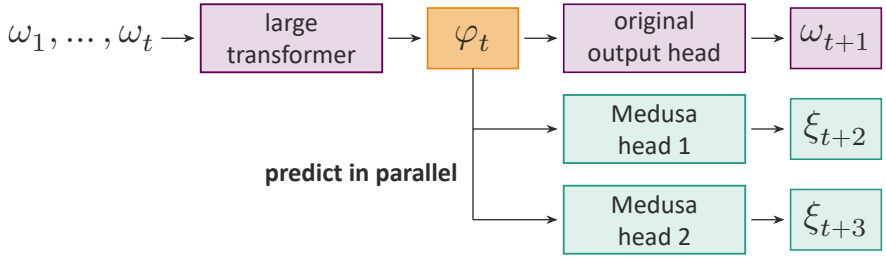




Medusa: Several Heads, One Hidden State

Cai et al. (2024)

Why another model, when the target already has clues about future tokens?



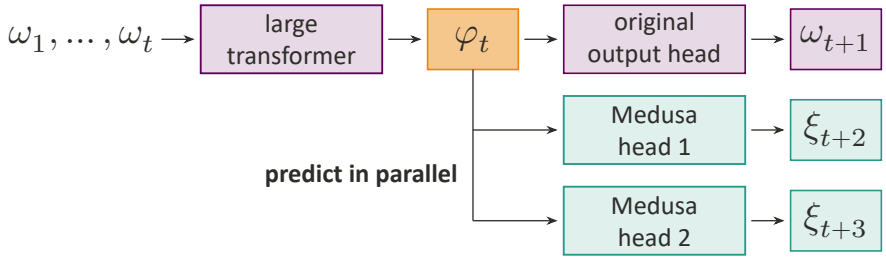
Predict several positions in parallel. Verify a candidate tree.



Medusa: Several Heads, One Hidden State

Cai et al. (2024)

Why another model, when the target already has clues about future tokens?



Predict several positions in parallel. Verify a candidate tree.

Later heads do not see earlier candidate choices.

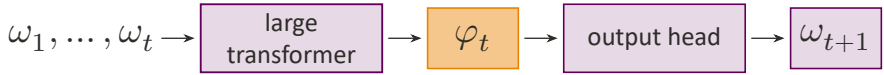


EAGLE-1

Li et al. (2024)

Can we advance the hidden state after each chosen token?

target model



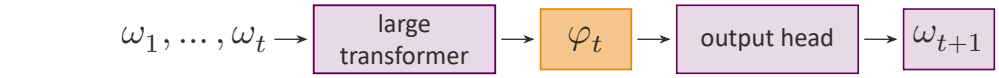


EAGLE-1

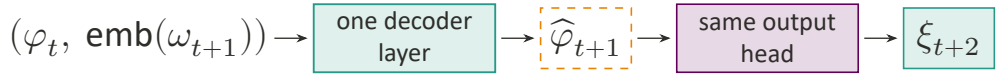
Li et al. (2024)

Can we advance the hidden state after each chosen token?

target model



EAGLE drafter



Train once to match target features and token probabilities.

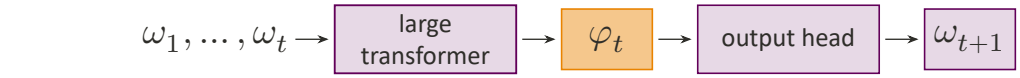


EAGLE-1

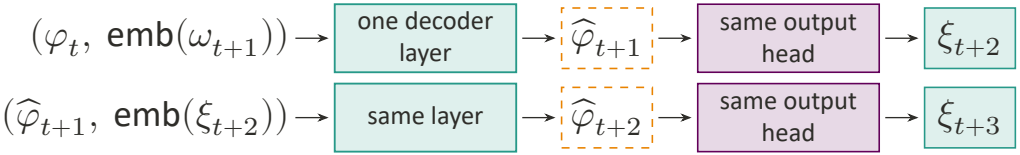
Li et al. (2024)

Can we advance the hidden state after each chosen token?

target model



EAGLE drafter



Train once to match target features and token probabilities.

Reuse the layer sequentially. Target verifies candidates together.



EAGLE-2 and EAGLE-3

Li et al. (2024, 2025)

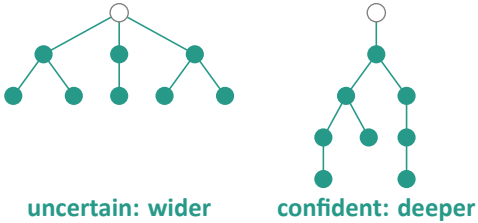


EAGLE-2 and EAGLE-3

Li et al. (2024, 2025)

EAGLE-2

Let confidence shape the tree



Same verification budget

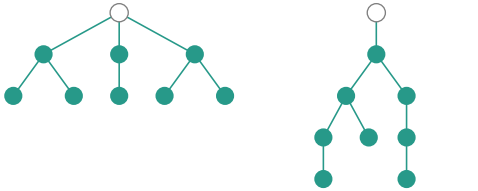
Favor paths likely to survive verification.

EAGLE-2 and EAGLE-3

Li et al. (2024, 2025)

EAGLE-2

Let confidence shape the tree

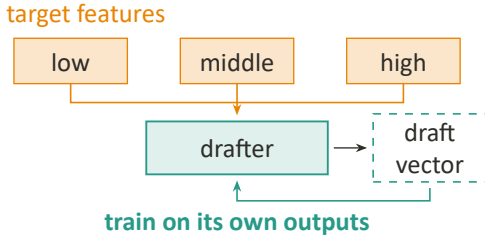


Same verification budget

Favor paths likely to survive verification.

EAGLE-3

A better trained drafter

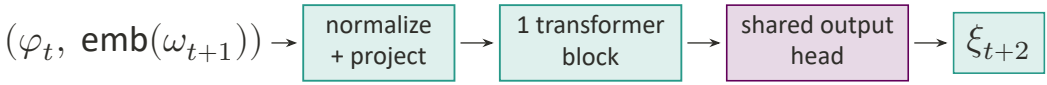


Token prediction loss only

Drop hidden state matching.

DeepSeek-V3: Multi-Token Prediction (MTP)

DeepSeek-AI (2024)

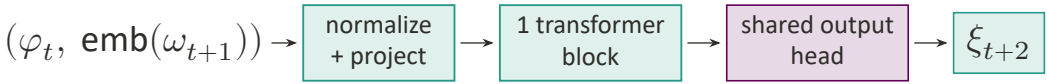


Like EAGLE: hidden state + preceding token drive each prediction.
Unlike Medusa: predictions follow a causal chain.



DeepSeek-V3: Multi-Token Prediction (MTP)

DeepSeek-AI (2024)



Like EAGLE: hidden state + preceding token drive each prediction.
Unlike Medusa: predictions follow a causal chain.

Training
Jointly train backbone + module.

Inference
Reuse the module as a drafter.



What Serving Systems Support

	vLLM	SGLang	TensorRT-LLM
separate draft model	yes	yes	yes
n-gram (prompt lookup)	yes	yes	yes
Medusa	–	–	legacy
EAGLE-1, EAGLE-2	yes	yes	legacy
EAGLE-3	yes	yes	yes
MTP (DeepSeek-V3)	yes	yes	yes

All three verify one chain by default; two also offer trees.

Source: documentation, October 2026: vLLM v0.32.0, SGLang v0.5.22, TensorRT-LLM 1.4.0rc1.



Part VI:
Other Approaches to Efficiency



Find the Dominant Cost First

Prefill: the pass over the prompt. **Time to first token**: queueing plus prefill. **Time per output token**: one decoding step.

Request time $\approx$ time to first token $+$
(output tokens -1) $\times$ time per output token.

as is

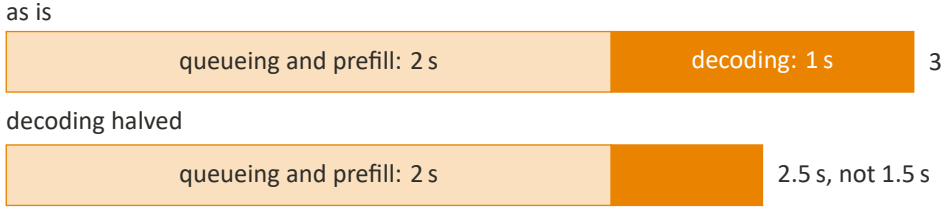




Find the Dominant Cost First

Prefill: the pass over the prompt. **Time to first token**: queueing plus prefill. **Time per output token**: one decoding step.

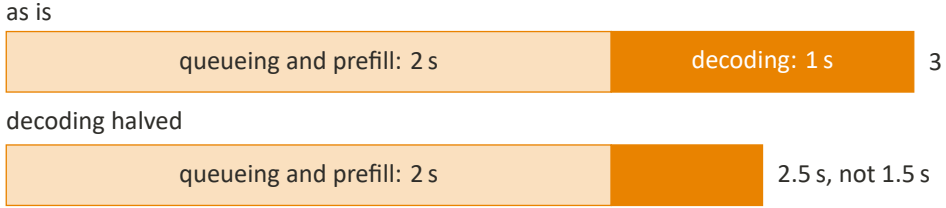
Request time $\approx$ time to first token $+$
(output tokens -1) $\times$ time per output token.



Find the Dominant Cost First

Prefill: the pass over the prompt. **Time to first token**: queueing plus prefill. **Time per output token**: one decoding step.

Request time $\approx$ time to first token $+$
(output tokens -1) $\times$ time per output token.



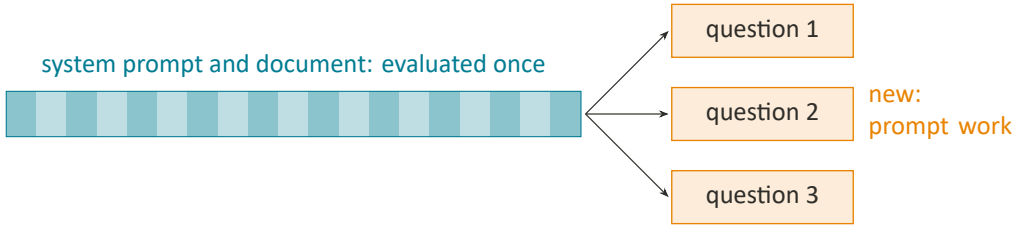
Speculation shortens only the decoding part.



Prefix Caching

SGLang: Zheng et al. (2024)

Three requests share a system prompt and a document, then ask different questions.

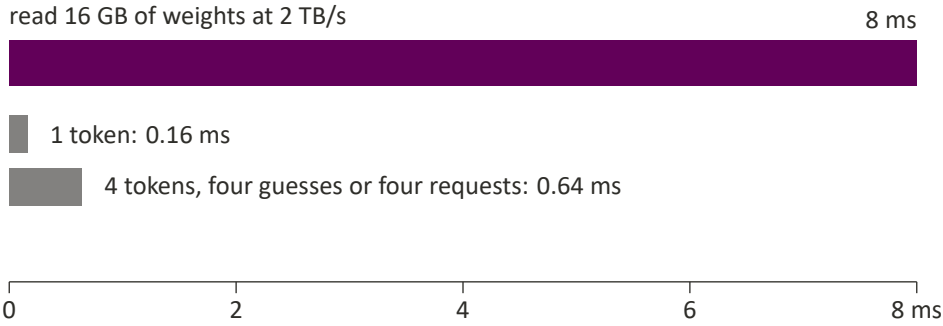


Prefix caching: the shared prefix is evaluated once; its keys and values are reused.



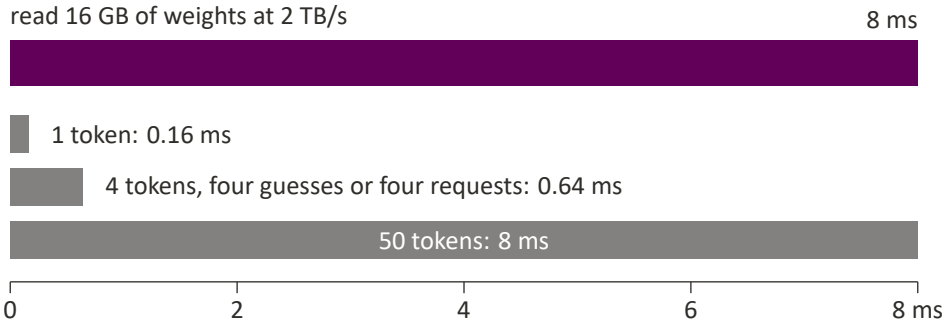
Batching Shares Each Weight Read

Orca: Yu et al. (2022)



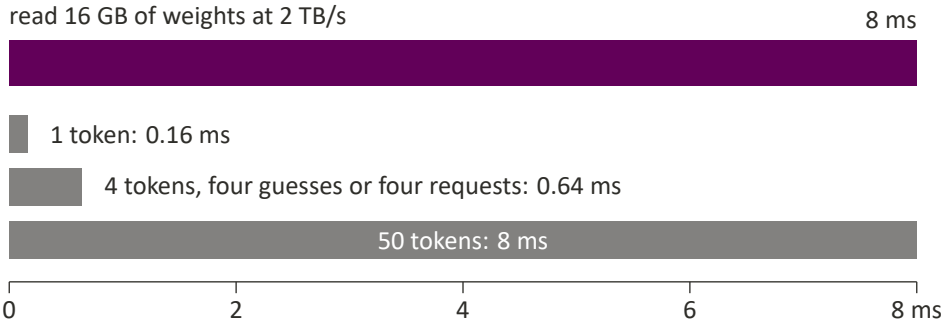
Batching Shares Each Weight Read

Orca: Yu et al. (2022)



Batching Shares Each Weight Read

Orca: Yu et al. (2022)

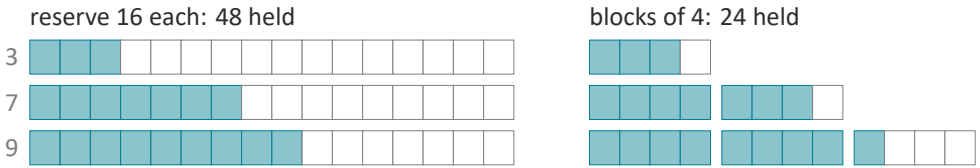


Batching and speculation compete for the same idle arithmetic.

The KV Cache Limits the Batch

Kwon et al. (2023)

Requests share the weights, but each holds its own KV cache.



PagedAttention: fixed-size cache blocks, allocated on demand.

Less wasted memory, more requests per weight read.

The Map in Four Principles

Principle	Examples
Move fewer bytes	quantization; fewer KV heads
Do more work per fetched byte	speculative verification; batching
Reuse previous computation	the KV cache; prefix caching
Use memory better	paging

A saving in one cost can expose another.



Speculative Decoding: Five Results

1. Checking several supplied tokens costs about one pass.
2. Accept, else draw from the residual: the output has law p.
3. Rounds keep the law of the whole sequence.
4. Constant α: a round commits $1 + \alpha + \cdots + \alpha^K$ tokens on average.
5. Better drafts and trees add progress per pass.

Cheap guesses change the schedule; exact verification keeps the law.



Research 1: Verifying Whole Drafts

One proposed token: the maximal coupling is the best exact verifier. With several drafts, a block or a tree, which verifier makes the most progress?

Several drafts for one token: the best verifier solves an optimal-transport problem, a linear program whose best-known solution time is exponential in the number of drafts. Practical rules fall short of it; for drafts drawn independently from one drafter it can now be computed to any accuracy.

Trees: judging whole paths, from leaf to root, stays exact and reports longer accepted runs, but is proved optimal only for a single chain.

Not settled by these results: the best verifier for a general tree, and how to choose the tree and the verifier together.

References: Sun et al. (NeurIPS 2023) · Hu et al. (ICLR 2025) · Weng et al. (NeurIPS 2025) · Thomas & Pal (ICLR 2026) · Sun et al. (ICLR 2025).



Research 2: Speculation Under Load

On a busy server, when does speculation still pay, and how should the draft length follow the load?

With many requests, batching already reuses the weights; speculation then competes for compute and memory with other useful work.

Measured: the best draft length falls as the batch grows, and speculation added without tuning can degrade serving. With long contexts the cache dominates, and speculation can speed up even large batches.

In a decision model of draft length, the best rule stops drafting once a rejection becomes likely. A rule that looks at the drafted tokens needs its own exactness argument.

References: Su et al. (2023) · Liu et al. (2024) · Sadhukhan et al. (ICLR 2025) · Huang et al. (COLM 2025).



Research 3: Training for Acceptance

A drafter is useful when the target accepts its tokens. How should it be trained for that, rather than to be a good model on its own?

One-token acceptance is one minus the total variation distance, so that distance is the natural loss. In practice it is not consistently the best, and training on text the drafter itself generates matters.

A drafter can keep learning from the queries a server receives, and can be trained on the contexts it will meet when drafting.

Which training loss maximizes accepted progress per unit of drafting cost?

References: Zhou et al. (ICLR 2024) · Liu et al. (ICML 2024) · Zhang et al. (ICLR 2025) · Li et al. (NeurIPS 2025).



Research 4: One State, Many Decisions

How small can a shared state representation be if it must support many future questions from a decision model (like Jev)?

Decision models return choice probabilities without generating text, but may still read a long state for each question. Prefix caching reuses prefill; repeated attention can remain costly.

How to compress the state before questions arrive, keeping the decision error below a desired error rate ε. What memory is necessary, and what changes under adaptive queries?

Compare memory, latency, and decision error against full prefix caching and learned compression.

References: Jev / TypeSafe AI · Gist (NeurIPS 2023) · COCOM (WSDM 2025).



Questions?

Next: post-training.

