

Speculative Decoding

CS6013 Seminar presentation

Indian Institute of Technology
Bombay

Ratan	Roll No.: [23B0059]
Keshaw	Roll No.: [24B0908]
Deevyanshu	Roll No.: [22B1029]
Ishaan	Roll No.: [24M2103]



vLLM

Architecture, Scheduling and CPU–GPU Execution

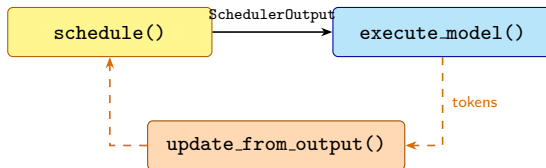
EngineCore → Scheduler → GPUModelRunner

Where does decoding time go, and what can a serving system hide?



EngineCore: the control loop

Every **engine step** runs the same three calls:



- ▶ `schedule()` (CPU): picks requests, how many tokens each one runs, and their KV blocks.
- ▶ `execute_model()` (GPU): one forward pass over the whole scheduled batch.
- ▶ `update_from_output()` (CPU): appends sampled tokens, finishes or frees requests.

One engine step = one target-model forward pass = **one new decode token**



Scheduler: token budget + KV gatekeeper

Core question: how many new tokens should request r run in this step?

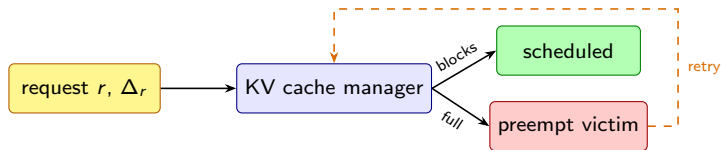
$$\Delta_r = \text{target}_r - \text{computed}_r, \quad \sum_r \Delta_r \leq \text{token budget}$$

Admission

- ▶ Max batched tokens per step
- ▶ Max running requests
- ▶ Long prompts split into chunks

KV memory

- ▶ Allocate KV blocks for Δ_r
- ▶ Reuse cached shared prefixes
- ▶ Out of blocks $\Rightarrow$ preempt

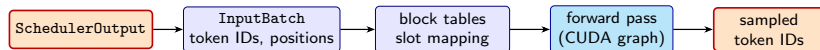


One batch can mix **chunked prefill** and **decode**. With speculative decoding, target_r also counts the draft tokens to verify.



GPUModelRunner: schedule $\rightarrow$ tensors $\rightarrow$ tokens

Turns a SchedulerOutput into one flat GPU batch.



On the GPU

embed $\rightarrow$ decoder layers (paged attention over KV blocks) $\rightarrow$ logits $\rightarrow$ sampler

Weights are read **once per step**, no matter how many requests share the batch.

Example: three decode requests

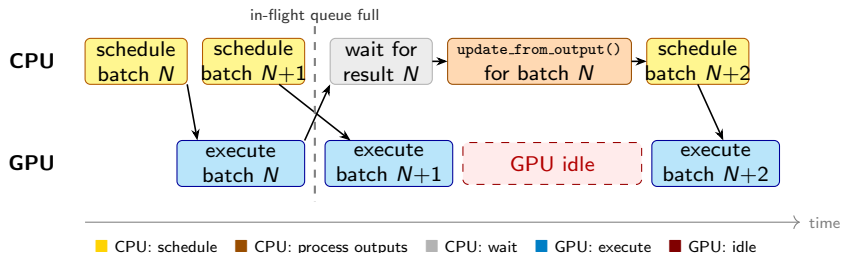
`num_scheduled_tokens = [1, 1, 1]`

Different lengths and block tables, still one flat batch of 3 tokens: each request gets its own positions and KV slots.



Async execution: overlapping CPU and GPU

CPU-side scheduling for batch $N+1$ runs while the GPU still executes batch N .



What overlaps

- ▶ GPU executes batch N
- ▶ EngineCore already schedules $N+1$
- ▶ Several batches can be in flight

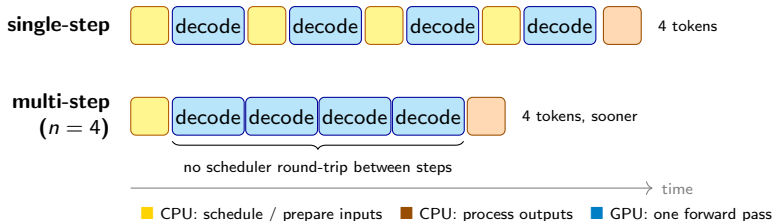
What still serialises

- ▶ Once the queue is full, the CPU waits
- ▶ It consumes the oldest result first
- ▶ Slow CPU work leaves GPU



Multi-step scheduling: amortising CPU overhead

vLLM V0 approach: schedule a decode batch once, then run n decode steps on the GPU before returning to the scheduler.



What is amortised

- ▶ Scheduler invocation
- ▶ CPU-side input preparation
- ▶ CPU–GPU synchronisation
- ▶ Output processing between steps

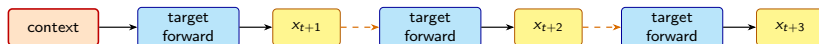
What does not change

- ▶ Decoding stays autoregressive
- ▶ Each step needs the previous token
- ▶ Still **one token per target forward pass**



The remaining bottleneck: one token per forward pass

Async and multi-step scheduling hide **CPU** overhead. The serial dependency remains:



What remains expensive

- ▶ x_{t+2} cannot start before x_{t+1} exists
- ▶ Every pass streams all weights + KV cache
- ▶ At small batch the GPU is memory-bound: compute sits idle

What scheduling cannot do

- ▶ Remove the token dependency
- ▶ Reduce target passes per token
- ▶ It only shortens the gaps between passes

Next: speculative decoding

Guess k tokens cheaply, verify them in **one** target pass $\Rightarrow$ up to $k+1$ tokens per pass.



Speculative Sampling & Speculative Decoding

Fast Inference from Transformers via Speculative Decoding

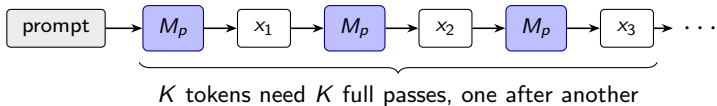
Leviathan, Kalman, Matias — ICML 2023

Accelerating Large Language Model Decoding with Speculative
Sampling

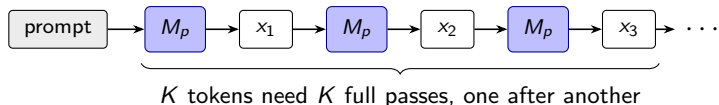
Chen, Borgeaud, Irving, Lepiau, Sifre, Jumper — DeepMind, 2023



Why is decoding slow?



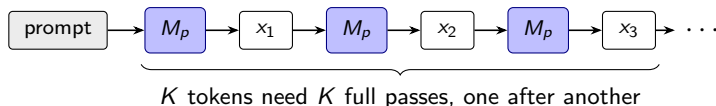
Why is decoding slow?



- ▶ Each pass moves **all the weights** from memory to the compute units.
- ▶ But it does only about 2 FLOPs per weight, so the GPU mostly **waits for memory**.



Why is decoding slow?



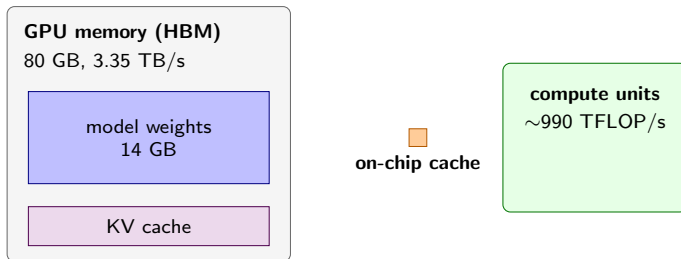
- ▶ Each pass moves **all the weights** from memory to the compute units.
- ▶ But it does only about 2 FLOPs per weight, so the GPU mostly **waits for memory**.

Solution?

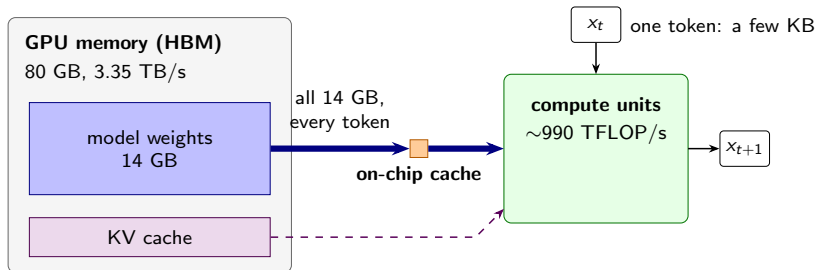
Generate several tokens **cheaply**, then verify **all of them at once** with a single pass of the large model.



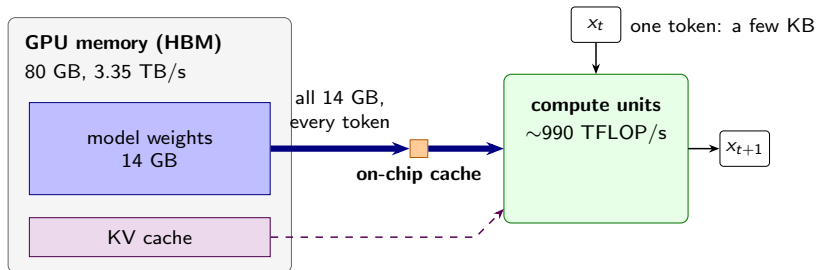
Where does the time go? Inside a GPU



Where does the time go? Inside a GPU



Where does the time go? Inside a GPU



Per token, for a 7B model in bf16 (rough numbers)

Moving the weights: $14 \text{ GB} / 3.35 \text{ TB/s} \approx 4 \text{ ms}$. Arithmetic: $14 \text{ GFLOP} / 990 \text{ TFLOP/s} \approx 0.014 \text{ ms}$.

The cache holds 0.4% of the model, so every token streams **all** the weights through it again. The compute units sit idle more than 99% of the time: checking k tokens in one pass costs about the same as checking one.



A quick experiment: finish the sentence

An apple a day keeps the _____ .



A quick experiment: finish the sentence

An apple a day keeps the **doctor away** .

Easy: a memorized idiom. Even a tiny language model gets this right.



A quick experiment: finish the sentence

An apple a day keeps the **doctor away** .

Easy: a memorized idiom. Even a tiny language model gets this right.

No amount of money ever bought a second of _____ .



A quick experiment: finish the sentence

An apple a day keeps the **doctor away** .

Easy: a memorized idiom. Even a tiny language model gets this right.

No amount of money ever bought a second of **time** .

Even if the idiom was never memorized, “a second of” points to *time*.



A quick experiment: finish the sentence

An apple a day keeps the **doctor away** .

Easy: a memorized idiom. Even a tiny language model gets this right.

No amount of money ever bought a second of **time** .

Even if the idiom was never memorized, “a second of” points to *time*.

The derivative of x^x is _____ .



A quick experiment: finish the sentence

An apple a day keeps the **doctor away** .

Easy: a memorized idiom. Even a tiny language model gets this right.

No amount of money ever bought a second of **time** .

Even if the idiom was never memorized, “a second of” points to *time*.

The derivative of x^x is **$x^x(1 + \ln x)$** .

Hard: this takes reasoning, and a small model will likely get it wrong.



A quick experiment: finish the sentence

An apple a day keeps the **doctor away** .

Easy: a memorized idiom. Even a tiny language model gets this right.

No amount of money ever bought a second of **time** .

Even if the idiom was never memorized, “a second of” points to *time*.

The derivative of x^x is **$x^x(1 + \ln x)$** .

Hard: this takes reasoning, and a small model will likely get it wrong.

Most tokens are easy, and only a few are hard. Let a **small** model guess the easy ones, and the **large** model check.



A quick experiment: finish the sentence

An apple a day keeps the **doctor away** .

Easy: a memorized idiom. Even a tiny language model gets this right.

No amount of money ever bought a second of **time** .

Even if the idiom was never memorized, “a second of” points to *time*.

The derivative of x^x is **$x^x(1 + \ln x)$** .

Hard: this takes reasoning, and a small model will likely get it wrong.

Who can do the guessing? Draft models, from cheapest to strongest

n-gram table
(a table lookup)

→
copy from
the context

→
tiny LM
(millions of parameters)

→
small LLM, same family
(billions of parameters)

Any of these works, as long as it uses the target's vocabulary (tokenizer).



The idea: guess, then check

A small, fast **draft** model writes a short continuation. The large **target** model checks, in one pass, whether it is good enough.



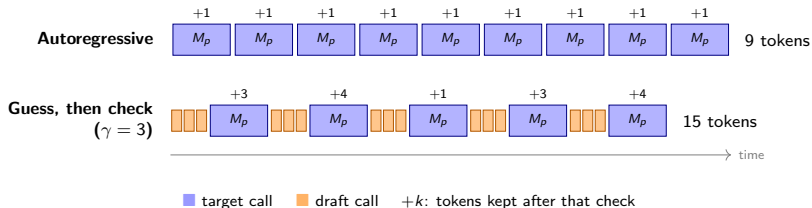
→ time

■ target call ■ draft call + k : tokens kept after that check



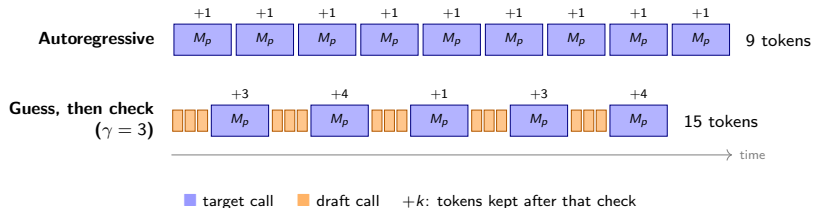
The idea: guess, then check

A small, fast **draft** model writes a short continuation. The large **target** model checks, in one pass, whether it is good enough.



The idea: guess, then check

A small, fast **draft** model writes a short continuation. The large **target** model checks, in one pass, whether it is good enough.



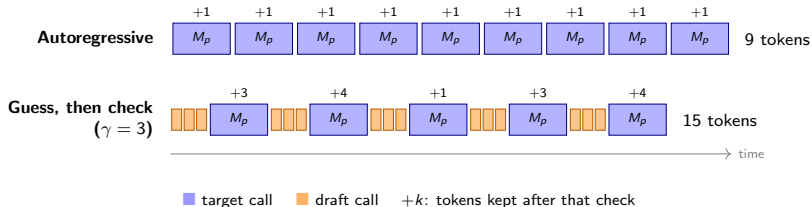
Question

What should "good enough" mean?



The idea: guess, then check

A small, fast **draft** model writes a short continuation. The large **target** model checks, in one pass, whether it is good enough.



Question

What should “good enough” mean?

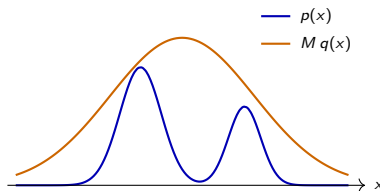
What we want

The output must be distributed **exactly** as if the target had generated it alone: the same quality, only faster. The classic tool for this is **rejection sampling**.



Review: rejection sampling

Goal: sample from p when we can only sample from q . Pick a constant M with $p(x) \leq M q(x)$ for all x .



Review: rejection sampling

Goal: sample from p when we can only sample from q . Pick a constant M with $p(x) \leq M q(x)$ for all x .

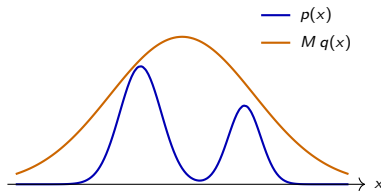
Rejection sampling

repeat

draw $x \sim q$ and $u \sim U(0, 1)$

if $u \leq \frac{p(x)}{M q(x)}$: **return** x

else: reject x and try again



Review: rejection sampling

Goal: sample from p when we can only sample from q . Pick a constant M with $p(x) \leq M q(x)$ for all x .

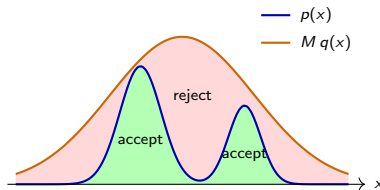
Rejection sampling

repeat

draw $x \sim q$ and $u \sim U(0, 1)$

if $u \leq \frac{p(x)}{M q(x)}$: **return** x

else: reject x and try again



The point $(x, u M q(x))$ is uniform under the orange curve. We keep x if the point lands under the blue curve.



Rejection sampling: correctness and cost

One trial proposes x and keeps it with probability $\frac{p(x)}{Mq(x)}$:

$$\mathbb{P}(\text{propose } x \text{ and keep it}) = q(x) \cdot \frac{p(x)}{M q(x)} = \frac{p(x)}{M} \propto p(x).$$

So a kept sample is distributed **exactly** as p .



Rejection sampling: correctness and cost

One trial proposes x and keeps it with probability $\frac{p(x)}{Mq(x)}$:

$$\mathbb{P}(\text{propose } x \text{ and keep it}) = q(x) \cdot \frac{p(x)}{Mq(x)} = \frac{p(x)}{M} \propto p(x).$$

So a kept sample is distributed **exactly** as p .

A trial succeeds with probability $\sum_x \frac{p(x)}{M} = \frac{1}{M}$, so we need **M trials on average**.



Rejection sampling: correctness and cost

One trial proposes x and keeps it with probability $\frac{p(x)}{Mq(x)}$:

$$\mathbb{P}(\text{propose } x \text{ and keep it}) = q(x) \cdot \frac{p(x)}{Mq(x)} = \frac{p(x)}{M} \propto p(x).$$

So a kept sample is distributed **exactly** as p .

A trial succeeds with probability $\sum_x \frac{p(x)}{M} = \frac{1}{M}$, so we need **M trials on average**.

A poor fit for decoding

- ▶ Every trial needs a **new** draft sample.
- ▶ $M = \max_x \frac{p(x)}{q(x)}$ can be huge, and the number of trials is unbounded.

Can we use **one** draft sample per position, and repair it when it is rejected?



Setup and notation

Target model M_p

Large and slow. Its next-token distribution:

$$p(x) \equiv p(x_t \mid x_{<t})$$



Setup and notation

Target model M_p

Large and slow. Its next-token distribution:

$$p(x) \equiv p(x_t \mid x_{<t})$$

Draft model M_q

Small and fast. Its next-token distribution:

$$q(x) \equiv q(x_t \mid x_{<t})$$



Setup and notation

Target model M_p

Large and slow. Its next-token distribution:

$$p(x) \equiv p(x_t \mid x_{<t})$$

Draft model M_q

Small and fast. Its next-token distribution:

$$q(x) \equiv q(x_t \mid x_{<t})$$

Draft length γ

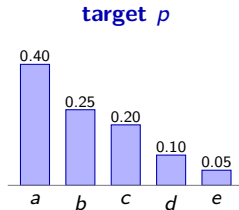
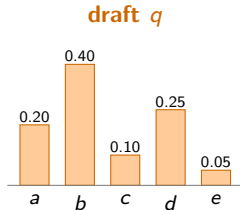
In each iteration, M_q proposes γ tokens and M_p scores all of them in one parallel pass.



Speculative sampling: one token

Speculative sampling

Draw a draft token $x \sim q$.



We hold one sample from q , but we want a sample from p .

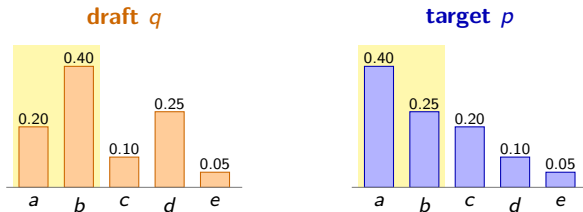


Speculative sampling: one token

Speculative sampling

Draw a draft token $x \sim q$.

Accept it with probability $\min\left(1, \frac{p(x)}{q(x)}\right)$.



a: $q(a) \leq p(a)$, so always accept. b: $q(b) > p(b)$, so accept with probability $\frac{0.25}{0.40} = 0.625$.



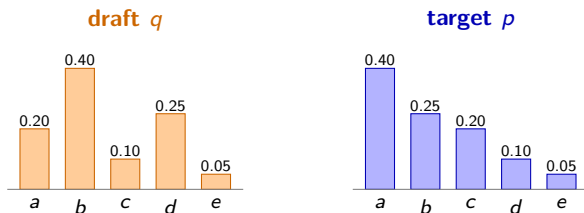
Speculative sampling: one token

Speculative sampling

Draw a draft token $x \sim q$.

Accept it with probability $\min(1, \frac{p(x)}{q(x)})$.

If rejected, sample instead from the **residual**
 $p'(x) = \text{norm}(\max(0, p(x) - q(x)))$.



After a rejection, draw **once** from p' . No constant M and no retry loop.



Algorithm 1: speculative decoding

SpeculativeDecodingStep

Inputs: M_p , M_q , $prefix$.

▷ Sample γ guesses $x_1, \dots, x_\gamma$ from M_q autoregressively.

for $i = 1$ **to** γ **do**

$q_i(x) \leftarrow M_q(prefix + [x_1, \dots, x_{i-1}])$

$x_i \sim q_i(x)$

end for

▷ Run M_p in parallel.

$p_1(x), \dots, p_{\gamma+1}(x) \leftarrow$
 $M_p(prefix), \dots, M_p(prefix$
 $[x_1, \dots, x_\gamma])$

▷ Determine the number of accepted guesses n .

$r_1 \sim U(0, 1), \dots, r_\gamma \sim U(0, 1)$

$n \leftarrow \min(\{i - 1 \mid 1 \leq i \leq \gamma, r_i > \frac{p_i(x_i)}{q_i(x_i)}\} \cup \{\gamma\})$

▷ Adjust the distribution from M_p if needed.

$p'(x) \leftarrow p_{n+1}(x)$

+ **if** $n < \gamma$ **then**

$p'(x) \leftarrow \text{norm}(\max(0, p_{n+1}(x) - q_{n+1}(x)))$
end if

▷ Return one token from M_p , and n tokens from M_q .

$t \sim p'(x)$

return $prefix + [x_1, \dots, x_n, t]$

Four parts: draft, score, count, correct.



Algorithm 1: speculative decoding

SpeculativeDecodingStep

Inputs: M_p , M_q , $prefix$.

▷ Sample γ guesses $x_1, \dots, x_\gamma$ from M_q autoregressively.

for $i = 1$ **to** γ **do**

$q_i(x) \leftarrow M_q(prefix + [x_1, \dots, x_{i-1}])$

$x_i \sim q_i(x)$

end for

▷ Run M_p in parallel.

$p_1(x), \dots, p_{\gamma+1}(x) \leftarrow$
 $M_p(prefix), \dots, M_p(prefix$
 $[x_1, \dots, x_\gamma])$

▷ Determine the number of accepted guesses n .

$r_1 \sim U(0, 1), \dots, r_\gamma \sim U(0, 1)$

$n \leftarrow \min(\{i - 1 \mid 1 \leq i \leq \gamma, r_i > \frac{p_i(x_i)}{q_i(x_i)}\} \cup \{\gamma\})$

▷ Adjust the distribution from M_p if needed.

$p'(x) \leftarrow p_{n+1}(x)$

+ **if** $n < \gamma$ **then**

$p'(x) \leftarrow \text{norm}(\max(0, p_{n+1}(x) - q_{n+1}(x)))$
end if

▷ Return one token from M_p , and n tokens from M_q .

$t \sim p'(x)$

return $prefix + [x_1, \dots, x_n, t]$

γ cheap calls of the draft, one after another.



Algorithm 1: speculative decoding

SpeculativeDecodingStep

Inputs: M_p , M_q , $prefix$.

▷ Sample γ guesses $x_1, \dots, x_\gamma$ from M_q autoregressively.

for $i = 1$ **to** γ **do**

$q_i(x) \leftarrow M_q(prefix + [x_1, \dots, x_{i-1}])$

$x_i \sim q_i(x)$

end for

▷ Run M_p in parallel.

$p_1(x), \dots, p_{\gamma+1}(x) \leftarrow$
 $M_p(prefix), \dots, M_p(prefix$
 $[x_1, \dots, x_\gamma])$

+

▷ Determine the number of accepted guesses n .

$r_1 \sim U(0, 1), \dots, r_\gamma \sim U(0, 1)$

$n \leftarrow \min(\{i - 1 \mid 1 \leq i \leq \gamma, r_i > \frac{p_i(x_i)}{q_i(x_i)}\} \cup \{\gamma\})$

▷ Adjust the distribution from M_p if needed.

$p'(x) \leftarrow p_{n+1}(x)$

if $n < \gamma$ **then**

$p'(x) \leftarrow \text{norm}(\max(0, p_{n+1}(x) - q_{n+1}(x)))$
end if

▷ Return one token from M_p , and n tokens from M_q .

$t \sim p'(x)$

return $prefix + [x_1, \dots, x_n, t]$

One call of the target scores all $\gamma + 1$ prefixes at once.



Algorithm 1: speculative decoding

SpeculativeDecodingStep

Inputs: M_p , M_q , $prefix$.

▷ Sample γ guesses $x_1, \dots, x_\gamma$ from M_q autoregressively.

for $i = 1$ **to** γ **do**

$q_i(x) \leftarrow M_q(prefix + [x_1, \dots, x_{i-1}])$

$x_i \sim q_i(x)$

end for

▷ Run M_p in parallel.

$p_1(x), \dots, p_{\gamma+1}(x) \leftarrow$
 $M_p(prefix), \dots, M_p(prefix$
 $[x_1, \dots, x_\gamma])$

▷ Determine the number of accepted guesses n .

$r_1 \sim U(0, 1), \dots, r_\gamma \sim U(0, 1)$

$n \leftarrow \min(\{i - 1 \mid 1 \leq i \leq \gamma, r_i > \frac{p_i(x_i)}{q_i(x_i)}\} \cup \{\gamma\})$

▷ Adjust the distribution from M_p if needed.

$p'(x) \leftarrow p_{n+1}(x)$

+ **if** $n < \gamma$ **then**

$p'(x) \leftarrow \text{norm}(\max(0, p_{n+1}(x) - q_{n+1}(x)))$
end if

▷ Return one token from M_p , and n tokens from M_q .

$t \sim p'(x)$

return $prefix + [x_1, \dots, x_n, t]$

n is the number of leading guesses that pass the one-token test.



Algorithm 1: speculative decoding

SpeculativeDecodingStep

Inputs: M_p , M_q , $prefix$.

▷ Sample γ guesses $x_1, \dots, x_\gamma$ from M_q autoregressively.

for $i = 1$ **to** γ **do**

$q_i(x) \leftarrow M_q(prefix + [x_1, \dots, x_{i-1}])$

$x_i \sim q_i(x)$

end for

▷ Run M_p in parallel.

$p_1(x), \dots, p_{\gamma+1}(x) \leftarrow$
 $M_p(prefix), \dots, M_p(prefix$
 $[x_1, \dots, x_\gamma])$

+

▷ Determine the number of accepted guesses n .

$r_1 \sim U(0, 1), \dots, r_\gamma \sim U(0, 1)$

$n \leftarrow \min(\{i - 1 \mid 1 \leq i \leq \gamma, r_i > \frac{p_i(x_i)}{q_i(x_i)}\} \cup \{\gamma\})$

▷ Adjust the distribution from M_p if needed.

$p'(x) \leftarrow p_{n+1}(x)$

if $n < \gamma$ **then**

$p'(x) \leftarrow \text{norm}(\max(0, p_{n+1}(x) - q_{n+1}(x)))$
end if

▷ Return one token from M_p , and n tokens from M_q .

$t \sim p'(x)$

return $prefix + [x_1, \dots, x_n, t]$

One more token comes from the target: the residual after a rejection, or $p_{\gamma+1}$ if every guess passed.



The main result

Theorem (speculative sampling is exact)

For **any** distributions p and q , the token x returned by speculative sampling satisfies

$$\mathbb{P}(x = x') = p(x') \quad \text{for every token } x'.$$



The main result

Theorem (speculative sampling is exact)

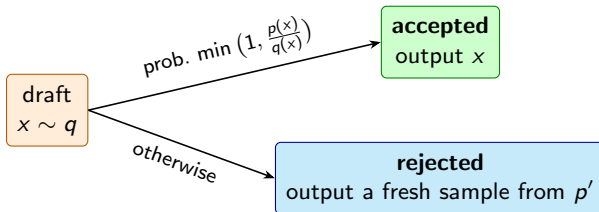
For **any** distributions p and q , the token x returned by speculative sampling satisfies

$$\mathbb{P}(x = x') = p(x') \quad \text{for every token } x'.$$

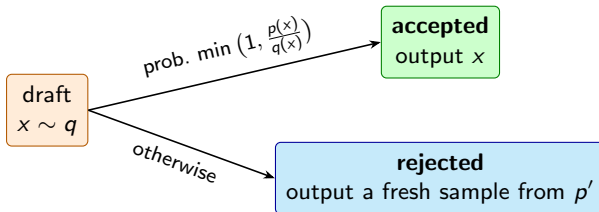
- ▶ The output has **exactly** the target's distribution: no loss in quality.
- ▶ The draft q only changes **how often** we accept, that is, the speed.



Proof (1/4): two ways to output x'



Proof (1/4): two ways to output x'



So, for any token x' :

$$\mathbb{P}(x = x') = \underbrace{\mathbb{P}(\text{accepted}, x = x')}_{\text{part 2}} + \underbrace{\mathbb{P}(\text{rejected}, x = x')}_{\text{part 3}}.$$



Proof (2/4): the accepted branch

Propose x' (probability $q(x')$), then keep it (probability $\min(1, \frac{p(x')}{q(x')})$):

$$\mathbb{P}(\text{accepted}, x = x') = q(x') \min\left(1, \frac{p(x')}{q(x')}\right).$$



Proof (2/4): the accepted branch

Propose x' (probability $q(x')$), then keep it (probability $\min(1, \frac{p(x')}{q(x')})$):

$$\mathbb{P}(\text{accepted}, x = x') = q(x') \min\left(1, \frac{p(x')}{q(x')}\right).$$

If $q(x') \leq p(x')$:

$$q(x') \cdot 1 = q(x')$$

If $q(x') > p(x')$:

$$q(x') \cdot \frac{p(x')}{q(x')} = p(x')$$



Proof (2/4): the accepted branch

Propose x' (probability $q(x')$), then keep it (probability $\min(1, \frac{p(x')}{q(x')})$):

$$\mathbb{P}(\text{accepted}, x = x') = q(x') \min\left(1, \frac{p(x')}{q(x')}\right).$$

If $q(x') \leq p(x')$:

$$q(x') \cdot 1 = q(x')$$

If $q(x') > p(x')$:

$$q(x') \cdot \frac{p(x')}{q(x')} = p(x')$$

$$\mathbb{P}(\text{accepted}, x = x') = \min(p(x'), q(x'))$$

$$\mathbb{P}(\text{accepted}) = \sum_x \min(p(x), q(x)) =: \beta$$



Proof (3/4): the rejected branch

A rejection happens with probability $\mathbb{P}(\text{rejected}) = 1 - \beta$.



Proof (3/4): the rejected branch

A rejection happens with probability $\mathbb{P}(\text{rejected}) = 1 - \beta$.

The residual's normalizer is the same number, because $\max(0, p - q) = p - \min(p, q)$:

$$\sum_x \max(0, p(x) - q(x)) = \sum_x (p(x) - \min(p(x), q(x))) = 1 - \beta,$$

so

$$p'(x') = \frac{p(x') - \min(p(x'), q(x'))}{1 - \beta}.$$



Proof (3/4): the rejected branch

A rejection happens with probability $\mathbb{P}(\text{rejected}) = 1 - \beta$.

The residual's normalizer is the same number, because $\max(0, p - q) = p - \min(p, q)$:

$$\sum_x \max(0, p(x) - q(x)) = \sum_x (p(x) - \min(p(x), q(x))) = 1 - \beta,$$

so

$$p'(x') = \frac{p(x') - \min(p(x'), q(x'))}{1 - \beta}.$$

The fresh sample does not depend on which draft was rejected:

$$\mathbb{P}(\text{rejected}, x = x') = (1 - \beta) p'(x') = p(x') - \min(p(x'), q(x')).$$



Proof (4/4): putting it together

$$\mathbb{P}(x = x') = \underbrace{\min(p(x'), q(x'))}_{\text{accepted}} + \underbrace{p(x') - \min(p(x'), q(x'))}_{\text{rejected}} = p(x'). \quad \blacksquare$$



Proof (4/4): putting it together

$$\mathbb{P}(x = x') = \underbrace{\min(p(x'), q(x'))}_{\text{accepted}} + \underbrace{p(x') - \min(p(x'), q(x'))}_{\text{rejected}} = p(x'). \quad \blacksquare$$

The accepted part and the resampled part add up to exactly the target p . Nothing about q was assumed.



Remark: lenience, trading exactness for speed

Accept with probability $\min\left(1, \frac{p(x)}{l q(x)}\right)$ for a **lenience** factor $l \in (0, 1]$.



Remark: lenience, trading exactness for speed

Accept with probability $\min\left(1, \frac{p(x)}{l q(x)}\right)$ for a **lenience** factor $l \in (0, 1]$.

- Acceptance rises: $\beta_l = \sum_x \min\left(\frac{p(x)}{l}, q(x)\right) \geq \beta.$



Remark: lenience, trading exactness for speed

Accept with probability $\min\left(1, \frac{p(x)}{l q(x)}\right)$ for a **lenience** factor $l \in (0, 1]$.

- ▶ Acceptance rises: $\beta_l = \sum_x \min\left(\frac{p(x)}{l}, q(x)\right) \geq \beta$.
- ▶ A weaker guarantee remains: no token is sampled with probability above $p(x)/l$, so rare tokens stay rare.



Remark: lenience, trading exactness for speed

Accept with probability $\min\left(1, \frac{p(x)}{l q(x)}\right)$ for a **lenience** factor $l \in (0, 1]$.

- ▶ Acceptance rises: $\beta_l = \sum_x \min\left(\frac{p(x)}{l}, q(x)\right) \geq \beta$.
- ▶ A weaker guarantee remains: no token is sampled with probability above $p(x)/l$, so rare tokens stay rare.
- ▶ But the output is **no longer exactly** p . Taking $l = 1$ gives back the exact method.



How good is the draft? The acceptance rate

Definition

$\beta_{x_{<t}}$: the probability that the draft token $x_t \sim q(x_t \mid x_{<t})$ is accepted.
 $\alpha = \mathbb{E}[\beta]$, averaged over prefixes: **one number** for how well M_q approximates M_p .



How good is the draft? The acceptance rate

Definition

$\beta_{x_{<t}}$: the probability that the draft token $x_t \sim q(x_t \mid x_{<t})$ is accepted.
 $\alpha = \mathbb{E}[\beta]$, averaged over prefixes: **one number** for how well M_q approximates M_p .

From the proof (the accepted branch, summed over x):

$$\beta = \sum_x q(x) \min \left(1, \frac{p(x)}{q(x)} \right) = \sum_x \min (p(x), q(x)).$$



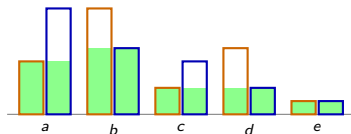
How good is the draft? The acceptance rate

Definition

$\beta_{x_{<t}}$: the probability that the draft token $x_t \sim q(x_t \mid x_{<t})$ is accepted.
 $\alpha = \mathbb{E}[\beta]$, averaged over prefixes: **one number** for how well M_q approximates M_p .

From the proof (the accepted branch, summed over x):

$$\beta = \sum_x q(x) \min \left(1, \frac{p(x)}{q(x)} \right) = \sum_x \min (p(x), q(x)).$$



β = the green part
of either set of bars
= 0.70 in the example



The acceptance rate is a distance between p and q

Definition

$$D_{LK}(p, q) = \sum_x |p(x) - M(x)| = \sum_x |q(x) - M(x)|, \quad \text{where } M = \frac{p+q}{2}.$$



The acceptance rate is a distance between p and q

Definition

$$D_{LK}(p, q) = \sum_x |p(x) - M(x)| = \sum_x |q(x) - M(x)|, \quad \text{where } M = \frac{p+q}{2}.$$

Lemma

$$D_{LK}(p, q) = 1 - \sum_x \min(p(x), q(x)).$$

Proof. Use $|p - M| = \frac{|p-q|}{2}$ and $\min(a, b) = \frac{a+b-|a-b|}{2}$:

$$\sum_x |p-M| = \sum_x \frac{|p-q|}{2} = 1 - \sum_x \frac{p+q-|p-q|}{2} = 1 - \sum_x \min(p, q). \quad \square$$



The acceptance rate is a distance between p and q

Definition

$$D_{LK}(p, q) = \sum_x |p(x) - M(x)| = \sum_x |q(x) - M(x)|, \quad \text{where } M = \frac{p+q}{2}.$$

Lemma

$$D_{LK}(p, q) = 1 - \sum_x \min(p(x), q(x)).$$

Proof. Use $|p - M| = \frac{|p-q|}{2}$ and $\min(a, b) = \frac{a+b-|a-b|}{2}$:

$$\sum_x |p-M| = \sum_x \frac{|p-q|}{2} = 1 - \sum_x \frac{p+q-|p-q|}{2} = 1 - \sum_x \min(p, q). \quad \square$$

Corollary

$$\beta = 1 - D_{LK}(p, q) \quad \text{and} \quad \alpha = 1 - \mathbb{E}[D_{LK}(p, q)].$$

D_{LK} is the total variation distance: symmetric, in $[0, 1]$, 0 iff $p = q$, and 1 iff p and q have disjoint supports.



Could an exact rule accept more often?

Question

An exact method takes a draft $X \sim q$ and outputs $Y \sim p$. How often can $Y = X$ be?

For any such pair:

$\mathbb{P}(X = Y = x) \leq \min(\mathbb{P}(X = x), \mathbb{P}(Y = x)) = \min(q(x), p(x))$. So

$$\mathbb{P}(X = Y) = \sum_x \mathbb{P}(X = Y = x) \leq \sum_x \min(p(x), q(x)) = \beta.$$

Speculative sampling reaches this bound. A rejected x has $q(x) > p(x)$, so $p'(x) = 0$ and the output differs from x . Hence $\mathbb{P}(X = Y) = \mathbb{P}(\text{accept}) = \beta$.

No exact rule that uses one draft keeps the draft token more often: the acceptance rule is **optimal**.



How many tokens does one target call produce?

Simplifying assumption

The β 's at successive positions are i.i.d., with mean α .

Let N be the number of tokens produced by one call of Algorithm 1 ($1 \leq N \leq \gamma + 1$).



How many tokens does one target call produce?

Simplifying assumption

The β 's at successive positions are i.i.d., with mean α .

Let N be the number of tokens produced by one call of Algorithm 1 ($1 \leq N \leq \gamma + 1$).

$N \geq k$ exactly when the first $k - 1$ guesses are accepted:

$$\mathbb{P}(N \geq k) = \alpha^{k-1}, \quad k = 1, \dots, \gamma + 1.$$

N is a **capped geometric** random variable.



How many tokens does one target call produce?

Simplifying assumption

The β 's at successive positions are i.i.d., with mean α .

Let N be the number of tokens produced by one call of Algorithm 1 ($1 \leq N \leq \gamma + 1$).

$N \geq k$ exactly when the first $k - 1$ guesses are accepted:

$$\mathbb{P}(N \geq k) = \alpha^{k-1}, \quad k = 1, \dots, \gamma + 1.$$

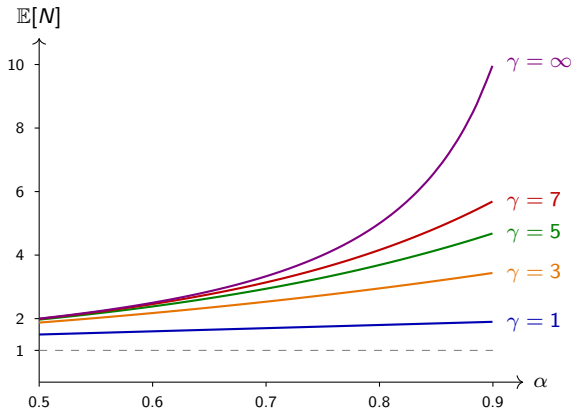
N is a **capped geometric** random variable.

$$\mathbb{E}[N] = \sum_{k=1}^{\gamma+1} \mathbb{P}(N \geq k) = \sum_{k=0}^{\gamma} \alpha^k$$

$$\mathbb{E}[N] = \frac{1 - \alpha^{\gamma+1}}{1 - \alpha} \xrightarrow{\gamma \rightarrow \infty} \frac{1}{1 - \alpha}$$



Expected tokens per target call



Example: $\alpha = 0.8$ and $\gamma = 5$ give $\mathbb{E}[N] = \frac{1-0.8^6}{0.2} \approx 3.69$ tokens per call.
Longer drafts only pay off when α is high.



Walltime speedup

Cost coefficient

$c = \frac{\text{time of one run of } M_q}{\text{time of one run of } M_p}.$ Unlike α , c depends on the hardware and the implementation.



Walltime speedup

Cost coefficient

$c = \frac{\text{time of one run of } M_q}{\text{time of one run of } M_p}$. Unlike α , c depends on the hardware and the implementation.

Assumption

There is enough idle compute that the $\gamma + 1$ parallel evaluations of M_p take as long as one.



Walltime speedup

Cost coefficient

$c = \frac{\text{time of one run of } M_q}{\text{time of one run of } M_p}$. Unlike α , c depends on the hardware and the implementation.

Assumption

There is enough idle compute that the $\gamma + 1$ parallel evaluations of M_p take as long as one.

Theorem (walltime improvement)

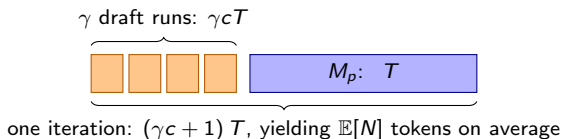
The expected improvement factor in total walltime is

$$S(\alpha, \gamma, c) = \frac{1 - \alpha^{\gamma+1}}{(1 - \alpha)(\gamma c + 1)}.$$



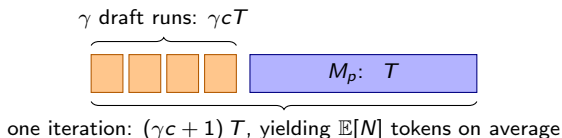
Proof of the speedup

Let T be the time of one run of M_p .



Proof of the speedup

Let T be the time of one run of M_p .



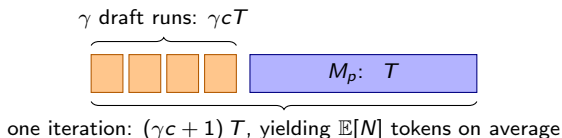
Expected time per token with speculative decoding:

$$\frac{(\gamma c + 1) T}{\mathbb{E}[N]} = \frac{(\gamma c + 1)(1 - \alpha)}{1 - \alpha^{\gamma+1}} T.$$



Proof of the speedup

Let T be the time of one run of M_p .



Expected time per token with speculative decoding:

$$\frac{(\gamma c + 1) T}{\mathbb{E}[N]} = \frac{(\gamma c + 1)(1 - \alpha)}{1 - \alpha^{\gamma+1}} T.$$

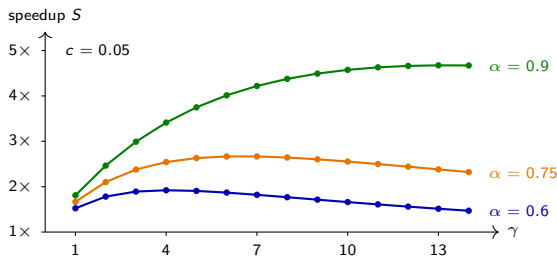
Standard decoding takes T per token. Dividing the two:

$$S = \frac{1 - \alpha^{\gamma+1}}{(1 - \alpha)(\gamma c + 1)}. \quad \square$$



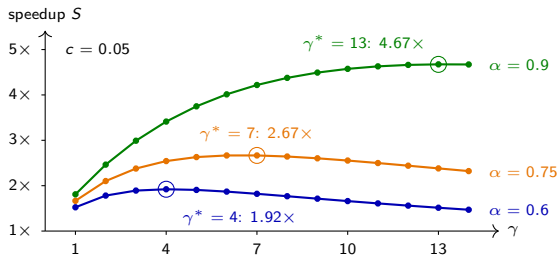
Choosing γ

$\gamma^* = \arg \max_{\gamma} S(\alpha, \gamma, c)$ is a one-dimensional integer search. A longer draft can yield more tokens per call, but it costs γc more, and its k -th guess survives only with probability α^k .



Choosing γ

$\gamma^* = \arg \max_{\gamma} S(\alpha, \gamma, c)$ is a one-dimensional integer search. A longer draft can yield more tokens per call, but it costs γc more, and its k -th guess survives only with probability α^k .

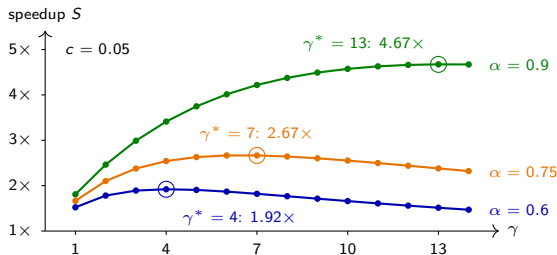


- γ^* grows with α and shrinks with c .



Choosing γ

$\gamma^* = \arg \max_{\gamma} S(\alpha, \gamma, c)$ is a one-dimensional integer search. A longer draft can yield more tokens per call, but it costs γc more, and its k -th guess survives only with probability α^k .



- ▶ γ^* grows with α and shrinks with c .
- ▶ A nearly free draft ($c \approx 0$, e.g. an n -gram table) gives $S = \mathbb{E}[N] \leq \frac{1}{1-\alpha}$.



The price: more arithmetic

Definition

$\hat{c}$: arithmetic operations per token of M_q , divided by those of M_p .

Theorem (arithmetic operations)

The expected factor of increase in the total number of operations is

$$\frac{(1 - \alpha)(\gamma \hat{c} + \gamma + 1)}{1 - \alpha^{\gamma+1}}.$$



The price: more arithmetic

Definition

$\hat{c}$: arithmetic operations per token of M_q , divided by those of M_p .

Theorem (arithmetic operations)

The expected factor of increase in the total number of operations is

$$\frac{(1 - \alpha)(\gamma \hat{c} + \gamma + 1)}{1 - \alpha^{\gamma+1}}.$$

Proof. Let $\hat{T}$ be the operations of standard decoding per token. One iteration costs $\hat{T} \hat{c} \gamma$ for the γ draft runs, plus $\hat{T}(\gamma + 1)$ for the $\gamma + 1$ target evaluations. Divide by the $\mathbb{E}[N]$ tokens produced, and by $\hat{T}$. $\square$



The price: more arithmetic

Definition

$\hat{c}$: arithmetic operations per token of M_q , divided by those of M_p .

Theorem (arithmetic operations)

The expected factor of increase in the total number of operations is

$$\frac{(1 - \alpha)(\gamma \hat{c} + \gamma + 1)}{1 - \alpha^{\gamma+1}}.$$

Proof. Let $\hat{T}$ be the operations of standard decoding per token. One iteration costs $\hat{T} \hat{c} \gamma$ for the γ draft runs, plus $\hat{T}(\gamma + 1)$ for the $\gamma + 1$ target evaluations. Divide by the $\mathbb{E}[N]$ tokens produced, and by $\hat{T}$. $\square$

Rejected guesses are wasted work, so the factor grows as α falls.



Speculative decoding (Leviathan et al.)

- ▶ Target: **T5-XXL**, 11B, encoder-decoder.
- ▶ Drafts: T5-small (77M), T5-base (250M), T5-large (800M). Existing checkpoints, no training.
- ▶ Tasks: English→German translation (WMT), CNN/DM summarization.
- ▶ One TPU-v4, batch size 1, compared with the T5X implementation.
- ▶ Greedy (temperature 0) and standard sampling (temperature 1).
- ▶ α also measured for a GPT-like 97M model and LaMDA 137B.



Experimental setup

Speculative decoding (Leviathan et al.)

- ▶ Target: **T5-XXL**, 11B, encoder-decoder.
- ▶ Drafts: T5-small (77M), T5-base (250M), T5-large (800M). Existing checkpoints, no training.
- ▶ Tasks: English→German translation (WMT), CNN/DM summarization.
- ▶ One TPU-v4, batch size 1, compared with the T5X implementation.
- ▶ Greedy (temperature 0) and standard sampling (temperature 1).
- ▶ α also measured for a GPT-like 97M model and LaMDA 137B.

Speculative sampling (Chen et al.)

- ▶ Target: **Chinchilla 70B** (80 layers, $d_{\text{model}} = 8192$) on 16 TPU v4, sharded across chips, batch size 1.
- ▶ Draft: 4B, 8 layers, $d_{\text{model}} = 6144$, same tokenizer and data. Wide and shallow, so little cross-chip communication: 1.8 vs 14.1 ms/token ($c \approx 0.13$).
- ▶ A 7B model would be slower here: it is fastest on 4 chips (5 ms/token) and slows down on 16.
- ▶ Tasks: XSum (ROUGE-2) and HumanEval (100-shot), with $\gamma = 4$.



Results: T5-XXL 11B

Walltime speedup over T5X at batch size 1. Temp.: sampling temperature (0 = greedy, 1 = standard sampling). Predicted: the speedup formula with the measured α and c .

Task	Draft M_q	Temp.	γ	α	c	Predicted	Measured
EnDe	T5-small (77M)	0	7	0.75	0.02	3.2×	3.4×
EnDe	T5-base (250M)	0	7	0.80	0.04	3.3×	2.8×
EnDe	T5-large (800M)	0	7	0.82	0.11	2.5×	1.7×
EnDe	T5-small	1	7	0.62	0.02	2.3×	2.6×
CNNDM	T5-small	0	5	0.65	0.02	2.4×	3.1×
CNNDM	T5-small	1	5	0.53	0.02	1.9×	2.3×



Results: T5-XXL 11B

Walltime speedup over T5X at batch size 1. Temp.: sampling temperature (0 = greedy, 1 = standard sampling). Predicted: the speedup formula with the measured α and c .

Task	Draft M_q	Temp.	γ	α	c	Predicted	Measured
EnDe	T5-small (77M)	0	7	0.75	0.02	3.2×	3.4×
EnDe	T5-base (250M)	0	7	0.80	0.04	3.3×	2.8×
EnDe	T5-large (800M)	0	7	0.82	0.11	2.5×	1.7×
EnDe	T5-small	1	7	0.62	0.02	2.3×	2.6×
CNNDM	T5-small	0	5	0.65	0.02	2.4×	3.1×
CNNDM	T5-small	1	5	0.53	0.02	1.9×	2.3×

- Larger drafts raise α , but raise c even faster: **T5-small** is the best trade-off.



Results: T5-XXL 11B

Walltime speedup over T5X at batch size 1. Temp.: sampling temperature (0 = greedy, 1 = standard sampling). Predicted: the speedup formula with the measured α and c .

Task	Draft M_q	Temp.	γ	α	c	Predicted	Measured
EnDe	T5-small (77M)	0	7	0.75	0.02	3.2×	3.4×
EnDe	T5-base (250M)	0	7	0.80	0.04	3.3×	2.8×
EnDe	T5-large (800M)	0	7	0.82	0.11	2.5×	1.7×
EnDe	T5-small	1	7	0.62	0.02	2.3×	2.6×
CNNDM	T5-small	0	5	0.65	0.02	2.4×	3.1×
CNNDM	T5-small	1	5	0.53	0.02	1.9×	2.3×

- ▶ Larger drafts raise α , but raise c even faster: **T5-small** is the best trade-off.
- ▶ Greedy decoding gives higher α and larger speedups.



Results: T5-XXL 11B

Walltime speedup over T5X at batch size 1. Temp.: sampling temperature (0 = greedy, 1 = standard sampling). Predicted: the speedup formula with the measured α and c .

Task	Draft M_q	Temp.	γ	α	c	Predicted	Measured
EnDe	T5-small (77M)	0	7	0.75	0.02	3.2×	3.4×
EnDe	T5-base (250M)	0	7	0.80	0.04	3.3×	2.8×
EnDe	T5-large (800M)	0	7	0.82	0.11	2.5×	1.7×
EnDe	T5-small	1	7	0.62	0.02	2.3×	2.6×
CNNDM	T5-small	0	5	0.65	0.02	2.4×	3.1×
CNNDM	T5-small	1	5	0.53	0.02	1.9×	2.3×

- ▶ Larger drafts raise α , but raise c even faster: **T5-small** is the best trade-off.
- ▶ Greedy decoding gives higher α and larger speedups.
- ▶ Predictions match measurements, up to implementation details and the i.i.d. assumption.



Results: acceptance rates α

Target M_p	Draft M_q	greedy	temperature 1
GPT-like (97M)	unigram	0.03	0.03
	bigram	0.05	0.05
	GPT-like (6M)	0.88	0.89
T5-XXL (EnDe)	unigram	0.08	0.07
	bigram	0.20	0.19
	T5-small	0.75	0.62
	T5-base	0.80	0.68
	T5-large	0.82	0.71
LaMDA (137B)	LaMDA (100M)	0.61	0.57
	LaMDA (2B)	0.71	0.71
	LaMDA (8B)	0.75	0.74



Results: acceptance rates α

Target M_p	Draft M_q	greedy	temperature 1
GPT-like (97M)	unigram	0.03	0.03
	bigram	0.05	0.05
	GPT-like (6M)	0.88	0.89
T5-XXL (EnDe)	unigram	0.08	0.07
	bigram	0.20	0.19
	T5-small	0.75	0.62
	T5-base	0.80	0.68
	T5-large	0.82	0.71
LaMDA (137B)	LaMDA (100M)	0.61	0.57
	LaMDA (2B)	0.71	0.71
	LaMDA (8B)	0.75	0.74

- Drafts about $100\times$ smaller than the target give α between 0.5 and 0.9.



Results: acceptance rates α

Target M_p	Draft M_q	greedy	temperature 1
GPT-like (97M)	unigram	0.03	0.03
	bigram	0.05	0.05
	GPT-like (6M)	0.88	0.89
T5-XXL (EnDe)	unigram	0.08	0.07
	bigram	0.20	0.19
	T5-small	0.75	0.62
	T5-base	0.80	0.68
	T5-large	0.82	0.71
LaMDA (137B)	LaMDA (100M)	0.61	0.57
	LaMDA (2B)	0.71	0.71
	LaMDA (8B)	0.75	0.74

- ▶ Drafts about $100\times$ smaller than the target give α between 0.5 and 0.9.
- ▶ Even a bigram table ($c \approx 0$) gives $\alpha \approx 0.2$: a $1.25\times$ speedup with $\gamma = 3$.



Results: lenience

T5-XXL target, T5-small draft, English→German, standard sampling, $c = 0.015$.

Speedups are expected values from the walltime formula.

Lenience /	1 (exact)	0.5	0.3	0.1
α	0.62	0.71	0.76	0.84
Expected speedup	$2.5\times$	$3.1\times$	$3.6\times$	$5\times$



Results: lenience

T5-XXL target, T5-small draft, English→German, standard sampling, $c = 0.015$.

Speedups are expected values from the walltime formula.

Lenience /	1 (exact)	0.5	0.3	0.1
α	0.62	0.71	0.76	0.84
Expected speedup	$2.5\times$	$3.1\times$	$3.6\times$	$5\times$

The extra speed is paid for with an output that is no longer distributed exactly as p .



Results: Chinchilla 70B

Batch size 1, $\gamma = 4$. ArS: ordinary autoregressive sampling. SpS: speculative sampling.

Benchmark	Sampling	Score: ArS / SpS	ms/token: ArS / SpS	Speedup
XSum (ROUGE-2)	nucleus 0.8	0.112 / 0.114	14.1 / 7.52	1.92×
XSum (ROUGE-2)	greedy	0.157 / 0.156	14.1 / 7.00	2.01×
HumanEval (100-shot)	nucleus 0.95, temperature 0.8	45.1% / 47.0%	14.1 / 5.73	2.46×



Results: Chinchilla 70B

Batch size 1, $\gamma = 4$. ArS: ordinary autoregressive sampling. SpS: speculative sampling.

Benchmark	Sampling	Score: ArS / SpS	ms/token: ArS / SpS	Speedup
XSum (ROUGE-2)	nucleus 0.8	0.112 / 0.114	14.1 / 7.52	1.92×
XSum (ROUGE-2)	greedy	0.157 / 0.156	14.1 / 7.00	2.01×
HumanEval (100-shot)	nucleus 0.95, temperature 0.8	45.1% / 47.0%	14.1 / 5.73	2.46×

- ▶ **1.9–2.5×** faster, with matching benchmark scores.



Results: Chinchilla 70B

Batch size 1, $\gamma = 4$. ArS: ordinary autoregressive sampling. SpS: speculative sampling.

Benchmark	Sampling	Score: ArS / SpS	ms/token: ArS / SpS	Speedup
XSum (ROUGE-2)	nucleus 0.8	0.112 / 0.114	14.1 / 7.52	1.92×
XSum (ROUGE-2)	greedy	0.157 / 0.156	14.1 / 7.00	2.01×
HumanEval (100-shot)	nucleus 0.95, temperature 0.8	45.1% / 47.0%	14.1 / 5.73	2.46×

- ▶ **1.9–2.5×** faster, with matching benchmark scores.
- ▶ Outputs are not bit-identical (different random streams and numerics), but they come from the same distribution.



Results: Chinchilla 70B

Batch size 1, $\gamma = 4$. ArS: ordinary autoregressive sampling. SpS: speculative sampling.

Benchmark	Sampling	Score: ArS / SpS	ms/token: ArS / SpS	Speedup
XSum (ROUGE-2)	nucleus 0.8	0.112 / 0.114	14.1 / 7.52	1.92×
XSum (ROUGE-2)	greedy	0.157 / 0.156	14.1 / 7.00	2.01×
HumanEval (100-shot)	nucleus 0.95, temperature 0.8	45.1% / 47.0%	14.1 / 5.73	2.46×

- ▶ **1.9–2.5×** faster, with matching benchmark scores.
- ▶ Outputs are not bit-identical (different random streams and numerics), but they come from the same distribution.
- ▶ Code has many predictable sub-sequences, so HumanEval gains the most.



Summary

Exactness

Accept $x \sim q$ with probability $\min(1, \frac{p(x)}{q(x)})$, else sample from $\text{norm}(\max(0, p - q))$. The output is distributed **exactly** as p .

Acceptance rate

$$\beta = \sum_x \min(p, q) = 1 - D_{LK}(p, q), \quad \alpha = \mathbb{E}[\beta]$$

Tokens per target call

$$\mathbb{E}[M] = \frac{1 - \alpha^{\gamma+1}}{1 - \alpha}$$

Speedup and cost

$$S = \frac{\mathbb{E}[M]}{\gamma c + 1}, \quad \text{operations} \\ \times \frac{(1 - \alpha)(\gamma \hat{c} + \gamma + 1)}{1 - \alpha^{\gamma+1}}$$

In practice: 2–3 $\times$ faster decoding, the same output distribution, and no change to the target model.



Can we do better?

- ▶ The draft is a separate model that never sees what the target computed.

Could the draft reuse the target's own hidden states?



Can we do better?

- ▶ The draft is a separate model that never sees what the target computed.

Could the draft reuse the target's own hidden states?

- ▶ One early rejection throws away every later guess.

Could we check several candidate continuations, a tree, in one target pass?



Can we do better?

- ▶ The draft is a separate model that never sees what the target computed.

Could the draft reuse the target's own hidden states?

- ▶ One early rejection throws away every later guess.

Could we check several candidate continuations, a tree, in one target pass?

- ▶ γ is fixed, but some continuations are much easier to guess than others.

Could the draft decide how far to guess?



Can we do better?

- ▶ The draft is a separate model that never sees what the target computed.

Could the draft reuse the target's own hidden states?

- ▶ One early rejection throws away every later guess.

Could we check several candidate continuations, a tree, in one target pass?

- ▶ γ is fixed, but some continuations are much easier to guess than others.

Could the draft decide how far to guess?

- ▶ Drafting still takes γ sequential calls.

Could the draft propose all γ tokens in a single pass?



Can we do better?

- ▶ The draft is a separate model that never sees what the target computed.

Could the draft reuse the target's own hidden states?

- ▶ One early rejection throws away every later guess.

Could we check several candidate continuations, a tree, in one target pass?

- ▶ γ is fixed, but some continuations are much easier to guess than others.

Could the draft decide how far to guess?

- ▶ Drafting still takes γ sequential calls.

Could the draft propose all γ tokens in a single pass?

- ▶ Everything relied on idle compute.

What happens at large batch sizes?



EAGLE

Speculative Sampling Requires Rethinking Feature Uncertainty

Li, Wei, Zhang, Zhang — ICML 2024



EAGLE: why not vanilla speculative decoding?

Problems with a separate draft LLM

- ▶ Needs a smaller model of the *same family and tokenizer*, which often doesn't exist.
 - ▶ *Same tokenizer*: verification compares draft and target probabilities of the same token ID, so both must share one vocabulary.
 - ▶ *Same family*: a draft trained on the same data guesses like the target, so more of its tokens get accepted.
 - ▶ LLaMA2 comes only in 7B/13B/70B: a 7B target has no smaller sibling, and training one from scratch is costly.
- ▶ Draft cost matters: a 7B draft for a 13B target gives *no* speedup.
- ▶ The draft never sees the target's internal states and guesses "from scratch".



EAGLE: why not vanilla speculative decoding?

Problems with a separate draft LLM

- ▶ Needs a smaller model of the *same family and tokenizer*, which often doesn't exist.
 - ▶ *Same tokenizer*: verification compares draft and target probabilities of the same token ID, so both must share one vocabulary.
 - ▶ *Same family*: a draft trained on the same data guesses like the target, so more of its tokens get accepted.
 - ▶ LLaMA2 comes only in 7B/13B/70B: a 7B target has no smaller sibling, and training one from scratch is costly.
- ▶ Draft cost matters: a 7B draft for a 13B target gives *no* speedup.
- ▶ The draft never sees the target's internal states and guesses “from scratch”.

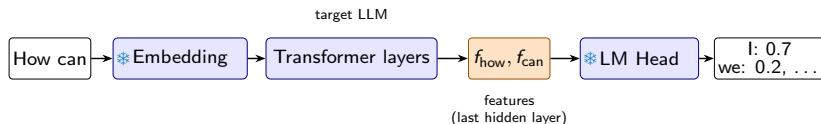
EAGLE

A tiny autoregressive draft head that **reuses the target's own features**: accuracy ≈ 0.8 , lossless, $2.7\text{--}3.5\times$ on LLaMA2-Chat-70B.



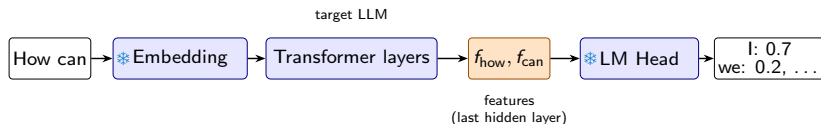
EAGLE key idea 1: guess features, not words

Inside an LLM, each word becomes a vector, which is refined by many layers. The last vector, the **feature** f , goes to the LM head, which gives next-word probabilities.



EAGLE key idea 1: guess features, not words

Inside an LLM, each word becomes a vector, which is refined by many layers. The last vector, the **feature** f , goes to the LM head, which gives next-word probabilities.



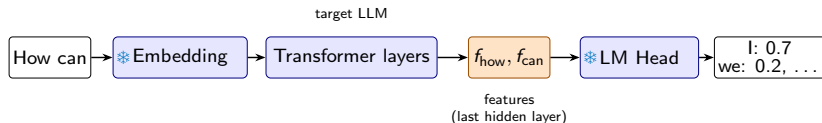
Vanilla draft: guess words

Guess the next *word* from the previous words. This is hard for a tiny model.



EAGLE key idea 1: guess features, not words

Inside an LLM, each word becomes a vector, which is refined by many layers. The last vector, the **feature** f , goes to the LM head, which gives next-word probabilities.



Vanilla draft: guess words

Guess the next *word* from the previous words. This is hard for a tiny model.

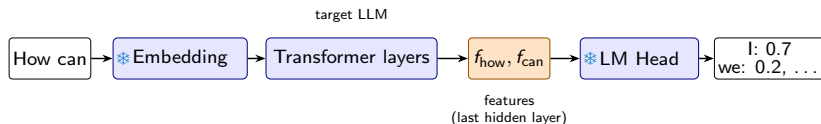
EAGLE: guess features

Guess the next *feature* $\hat{f}$ from the target's features, then reuse the target's LM head to get words. Features are smoother, so easier to guess.



EAGLE key idea 1: guess features, not words

Inside an LLM, each word becomes a vector, which is refined by many layers. The last vector, the **feature** f , goes to the LM head, which gives next-word probabilities.



Vanilla draft: guess words

Guess the next *word* from the previous words. This is hard for a tiny model.

EAGLE: guess features

Guess the next *feature* $\hat{f}$ from the target's features, then reuse the target's LM head to get words. Features are smoother, so easier to guess.

Paper's test: speedup $1.5\times \rightarrow 1.9\times$ just from this change.



EAGLE key idea 2: tell the draft which word was picked

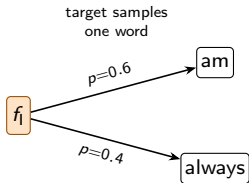
A feature f only gives the **probabilities** of the next word. The word actually output is **sampled** from them, so f alone does not say which word came next.

f_1



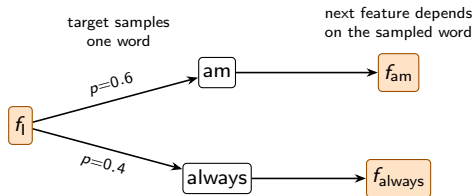
EAGLE key idea 2: tell the draft which word was picked

A feature f only gives the **probabilities** of the next word. The word actually output is **sampled** from them, so f alone does not say which word came next.



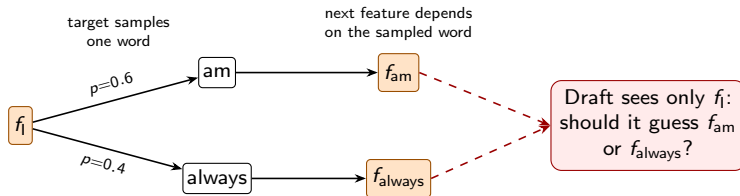
EAGLE key idea 2: tell the draft which word was picked

A feature f only gives the **probabilities** of the next word. The word actually output is **sampled** from them, so f alone does not say which word came next.



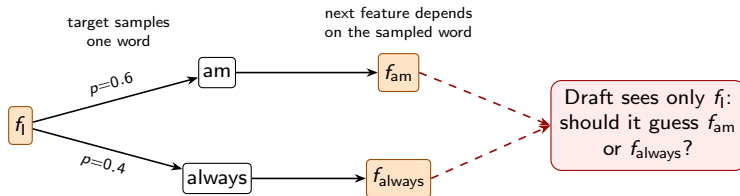
EAGLE key idea 2: tell the draft which word was picked

A feature f only gives the **probabilities** of the next word. The word actually output is **sampled** from them, so f alone does not say which word came next.

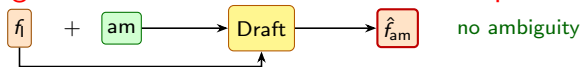


EAGLE key idea 2: tell the draft which word was picked

A feature f only gives the **probabilities** of the next word. The word actually output is **sampled** from them, so f alone does not say which word came next.



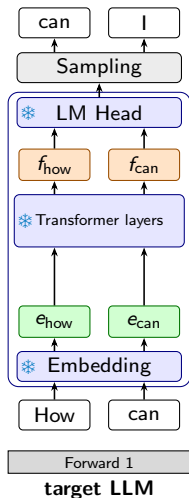
Fix: also give the draft the word that was sampled



Each feature is **paired with the word sampled from it**, e.g. $(f_{\text{how}}, \text{can})$, $(f_{\text{can}}, \text{I})$ on the next slide. Speedup: $1.9\times \rightarrow 2.8\times$, at almost no extra cost.



EAGLE inference (1/4): drafting, step by step

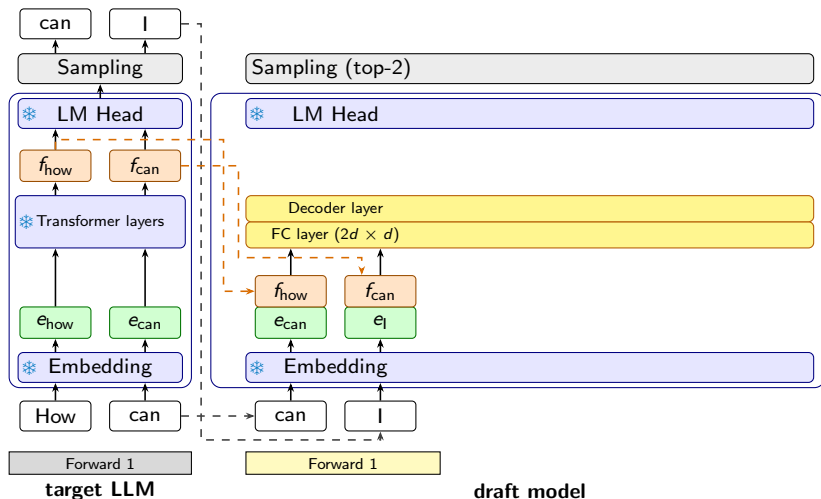


e word embedding f feature red border: guessed by the draft ❄️ frozen, copied from the target

The big **target LLM** reads “How can” and picks the next word “I”.
Along the way it computes a feature f for each word.



EAGLE inference (1/4): drafting, step by step



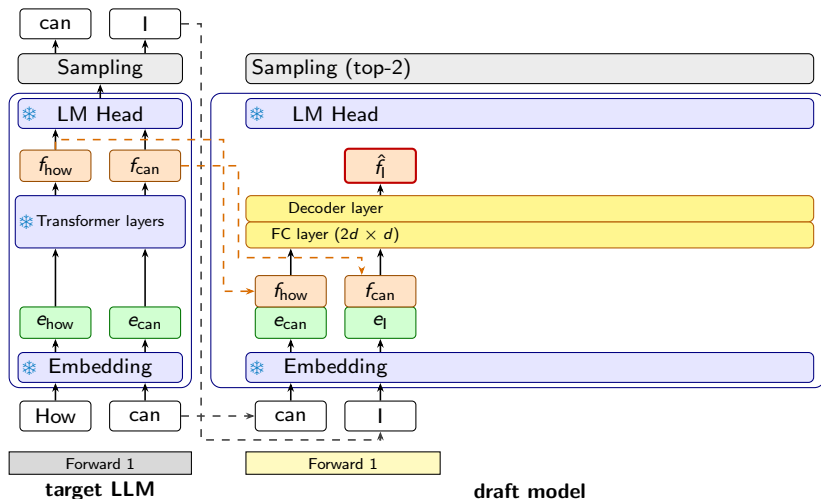
Legend:

- e word embedding
- f feature
- red border: guessed by the draft
- ❄ frozen, copied from the target

Forward 1, input: the target's features, each paired with the *next* word: $(f_{\text{how}}, \text{can})$, $(f_{\text{can}}, \text{I})$.



EAGLE inference (1/4): drafting, step by step

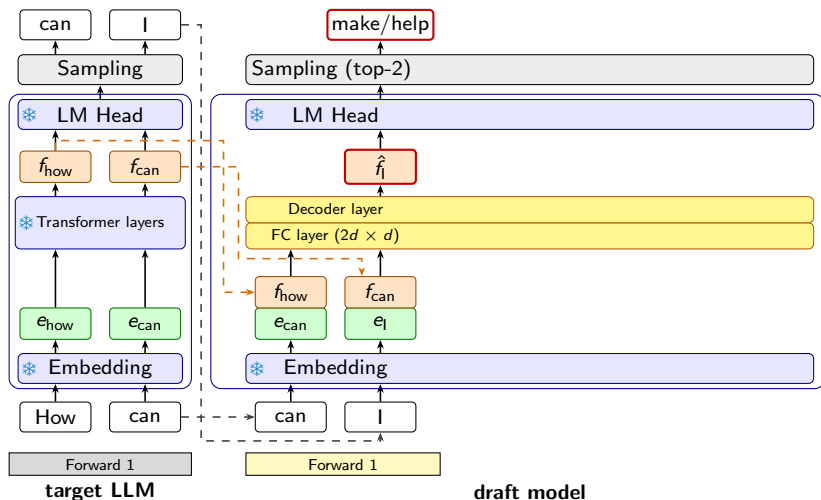


e word embedding **f** feature **red border:** guessed by the draft **❄** frozen, copied from the target

Forward 1: the small autoregression head (an FC layer, then one Transformer decoder layer) guesses the next feature $\hat{f}_{\text{I}}$.



EAGLE inference (1/4): drafting, step by step

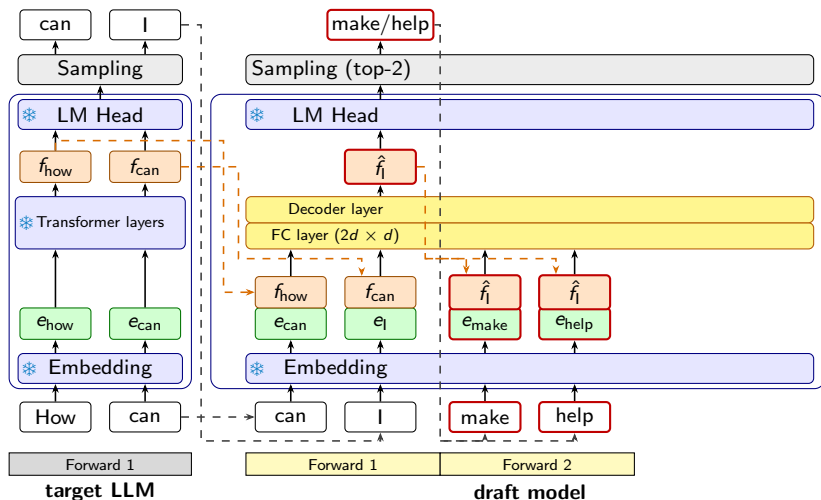


e word embedding **f** feature **red border:** guessed by the draft **❄** frozen, copied from the target

Forward 1: the LM head turns $\hat{f}_1$ into word probabilities. Keep the 2 likeliest: *make, help*.



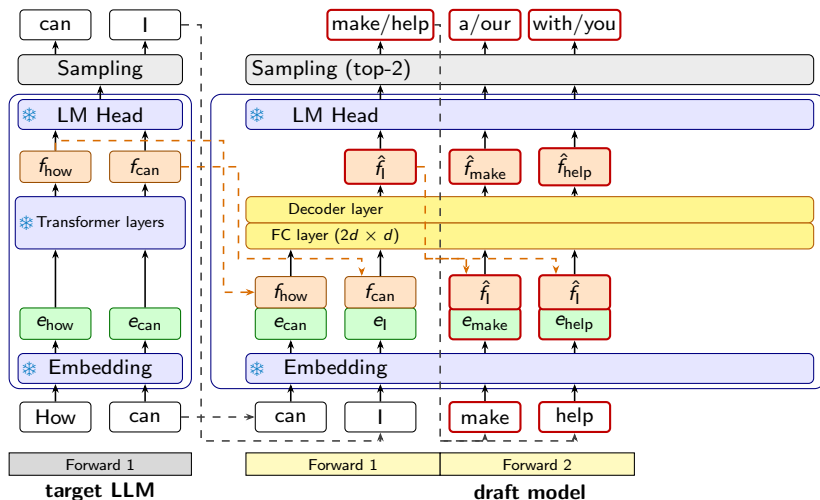
EAGLE inference (1/4): drafting, step by step



Forward 2, input: *make* and *help* together, each paired with their parent's guessed feature $\hat{f}_{\text{I}}$.



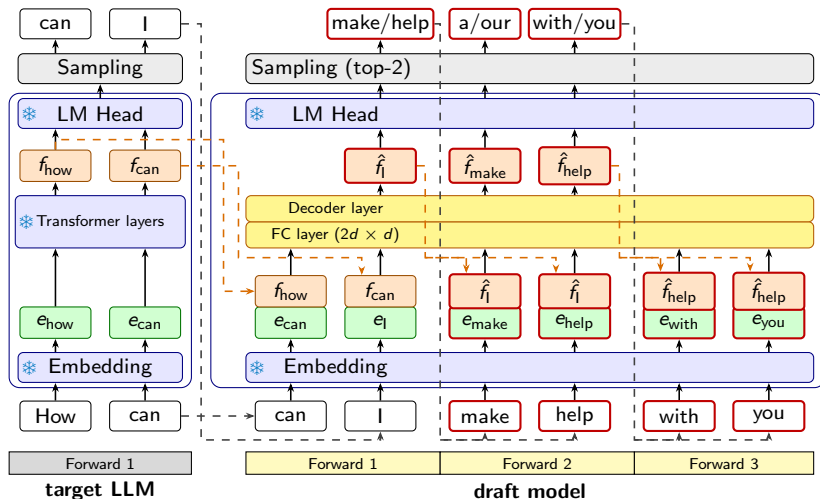
EAGLE inference (1/4): drafting, step by step



Forward 2: one pass gives children for both: *make* → *a/our*, *help* → *with/you*.



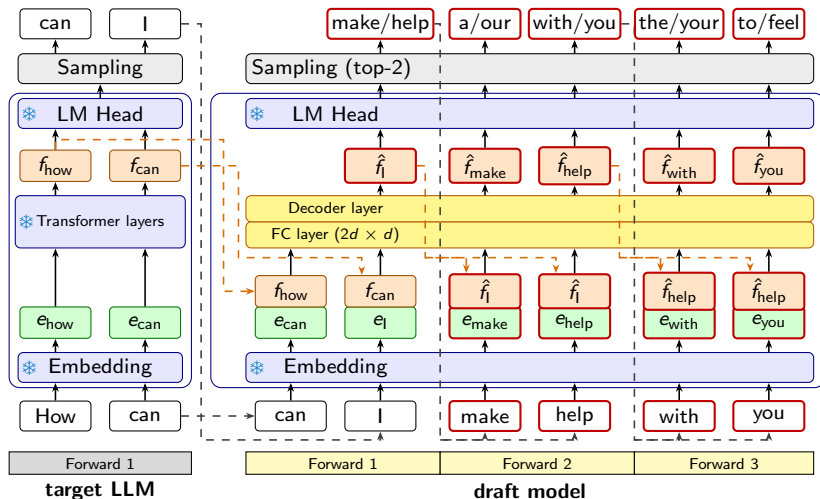
EAGLE inference (1/4): drafting, step by step



Forward 3, input: only the children of *help* are expanded here, each with $\hat{f}_{\text{help}}$.



EAGLE inference (1/4): drafting, step by step



Three cheap draft passes built a **tree of guesses**. Next, the target checks all of them at once.



EAGLE inference (2/4): the draft tree has a fixed shape

Instead of one chain of guesses, EAGLE guesses a **tree**: a few options at each step, so one of them is more likely to be right.

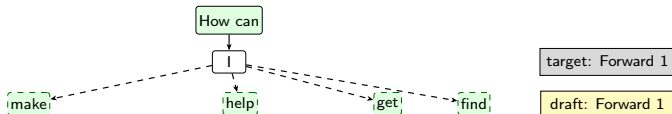


target: Forward 1



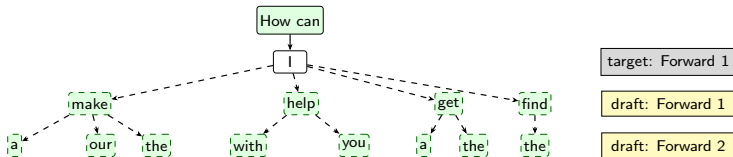
EAGLE inference (2/4): the draft tree has a fixed shape

Instead of one chain of guesses, EAGLE guesses a **tree**: a few options at each step, so one of them is more likely to be right.



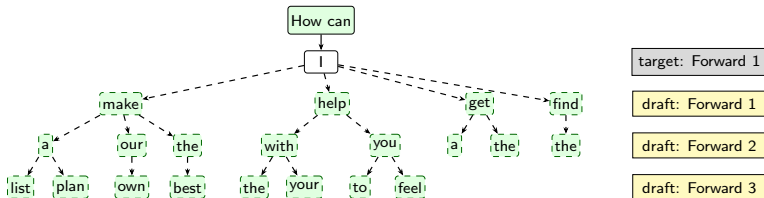
EAGLE inference (2/4): the draft tree has a fixed shape

Instead of one chain of guesses, EAGLE guesses a **tree**: a few options at each step, so one of them is more likely to be right.



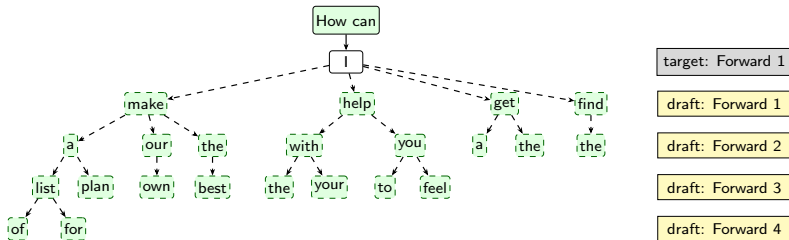
EAGLE inference (2/4): the draft tree has a fixed shape

Instead of one chain of guesses, EAGLE guesses a **tree**: a few options at each step, so one of them is more likely to be right.



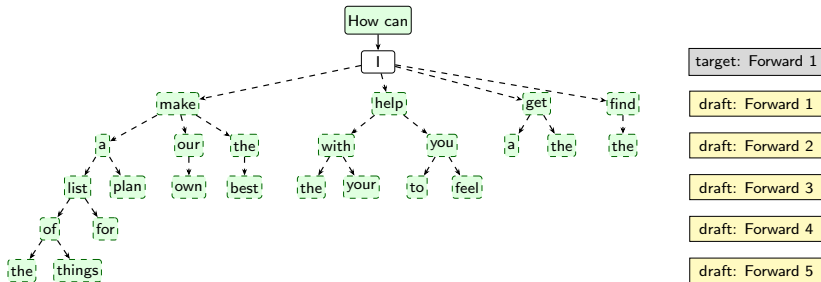
EAGLE inference (2/4): the draft tree has a fixed shape

Instead of one chain of guesses, EAGLE guesses a **tree**: a few options at each step, so one of them is more likely to be right.



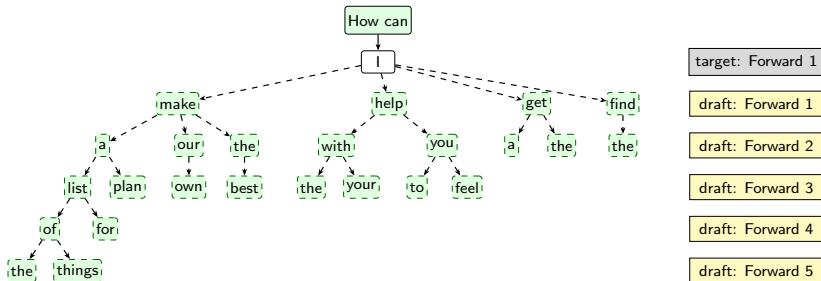
EAGLE inference (2/4): the draft tree has a fixed shape

Instead of one chain of guesses, EAGLE guesses a **tree**: a few options at each step, so one of them is more likely to be right.



EAGLE inference (2/4): the draft tree has a fixed shape

Instead of one chain of guesses, EAGLE guesses a **tree**: a few options at each step, so one of them is more likely to be right.

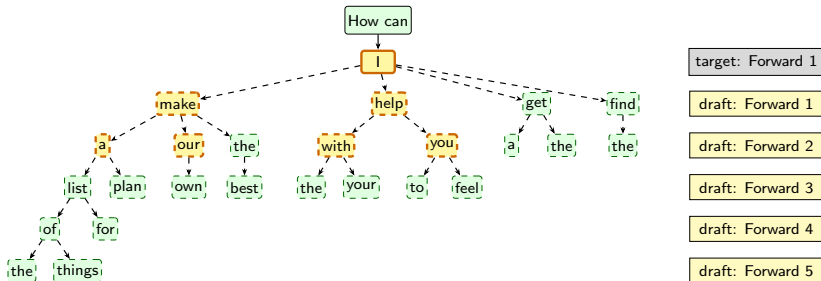


- This is the size EAGLE really uses: about 25 guesses, 5 levels. The **shape** is **chosen by hand in advance** and is the same for every input. The draft's top guess gets the most children, and the tree narrows with depth.



EAGLE inference (2/4): the draft tree has a fixed shape

Instead of one chain of guesses, EAGLE guesses a **tree**: a few options at each step, so one of them is more likely to be right.

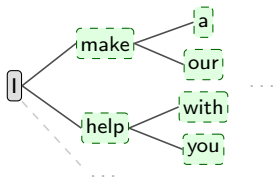


- ▶ This is the size EAGLE really uses: about 25 guesses, 5 levels. The **shape** is **chosen by hand in advance** and is the same for every input. The draft's top guess gets the most children, and the tree narrows with depth.
- ▶ The next slides zoom into the **highlighted part** to keep things small.



EAGLE inference (3/4): tree attention

Check **all** guesses with **one** target pass: put them in one sequence, and use a mask so each word sees **only its own path**.



(part of the full tree)

one sequence:

I make help a our with you

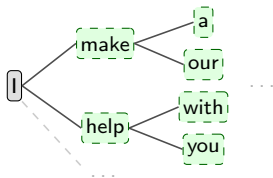
key (word it may look at)

	I	make	help	a	our	with	you
I							
make							
help							
a							
our							
with							
you							



EAGLE inference (3/4): tree attention

Check **all** guesses with **one** target pass: put them in one sequence, and use a mask so each word sees **only its own path**.



(part of the full tree)

one sequence:

I make help a our with you

key (word it may look at)

	I	make	help	a	our	with	you
I	Blue						
make		Blue					
help			Blue				
a				Blue			
our					Blue		
with						Blue	
you							Blue

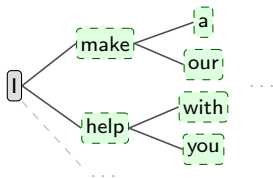
query (word being processed)

► Blue = may look at (every word also sees the prompt): itself



EAGLE inference (3/4): tree attention

Check **all** guesses with **one** target pass: put them in one sequence, and use a mask so each word sees **only its own path**.



(part of the full tree)

one sequence:

I make help a our with you

key (word it may look at)

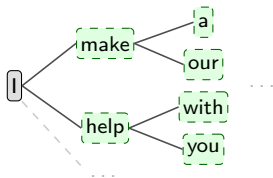
	I	make	help	a	our	with	you
I							
make							
help							
a							
our							
with							
you							

- ▶ Blue = may look at (every word also sees the prompt): itself, and its **ancestors**. *with* sees *I*, *help*, not *make*: as if the text were “How can I help with”.



EAGLE inference (3/4): tree attention

Check **all** guesses with **one** target pass: put them in one sequence, and use a mask so each word sees **only its own path**.



(part of the full tree)

one sequence:

I make help a our with you

key (word it may look at)

	I	make	help	a	our	with	you	position
I	Blue							3
make	Blue	Blue						4
help	Blue		Blue					4
a	Blue	Blue		Blue				5
our	Blue	Blue			Blue			5
with	Blue		Blue			Blue		5
you	Blue						Blue	5

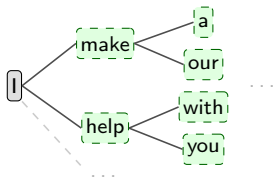
query (word being processed)

- ▶ Blue = may look at (every word also sees the prompt): itself, and its **ancestors**. *with* sees *I*, *help*, not *make*: as if the text were “How can I help with”.
- ▶ Positions continue the prompt (How = 1, can = 2); siblings share a position.



EAGLE inference (3/4): tree attention

Check **all** guesses with **one** target pass: put them in one sequence, and use a mask so each word sees **only its own path**.



(part of the full tree)

one sequence:

I make help a our with you

key (word it may look at)

	I	make	help	a	our	with	you	position
I	Blue							3
make	Blue	Blue						4
help	Blue		Blue					4
a	Blue	Blue		Blue				5
our	Blue	Blue			Blue			5
with	Blue		Blue			Blue		5
you	Blue						Blue	5

query (word being processed)

- ▶ Blue = may look at (every word also sees the prompt): itself, and its **ancestors**. *with* sees *I*, *help*, not *make*: as if the text were “How can I help with”.
- ▶ Positions continue the prompt (How = 1, can = 2); siblings share a position.
- ▶ **One** target pass gives the target’s probabilities at **every** node.

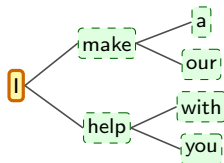


EAGLE inference (4/4): accept or reject

Walk down the tree with the vanilla rule (p : target, $\hat{p}$: draft).

Verify the tree

1. Start at the last accepted word.
2. Try its next child c : accept with probability $\min\left(1, \frac{p(c)}{\hat{p}(c)}\right)$.
3. **Accepted:** output c , move down to c , go to 2.
4. **Rejected:** $p \leftarrow \text{norm}(\max(0, p - \hat{p}))$, try the next sibling (go to 2).
5. **No child left**, or at a leaf: sample a *bonus* word from p and stop.



being tried accepted rejected bonus

One target pass (previous slide) gave p at every node. Start at I , the last word the target already accepted.

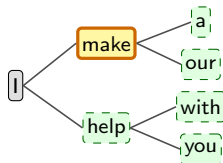


EAGLE inference (4/4): accept or reject

Walk down the tree with the vanilla rule (p : target, $\hat{p}$: draft).

Verify the tree

1. Start at the last accepted word.
2. Try its next child c : accept with probability $\min\left(1, \frac{p(c)}{\hat{p}(c)}\right)$.
3. **Accepted:** output c , move down to c , go to 2.
4. **Rejected:** $p \leftarrow \text{norm}(\max(0, p - \hat{p}))$, try the next sibling (go to 2).
5. **No child left**, or at a leaf: sample a *bonus* word from p and stop.



being tried accepted rejected bonus

Try *make*: $p = 0.2$, $\hat{p} = 0.5$, so accept with probability 0.4. The random draw says no. (example numbers)

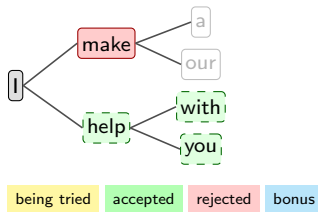


EAGLE inference (4/4): accept or reject

Walk down the tree with the vanilla rule (p : target, $\hat{p}$: draft).

Verify the tree

1. Start at the last accepted word.
2. Try its next child c : accept with probability $\min\left(1, \frac{p(c)}{\hat{p}(c)}\right)$.
3. **Accepted:** output c , move down to c , go to 2.
4. **Rejected:** $p \leftarrow \text{norm}\left(\max(0, p - \hat{p})\right)$, try the next sibling (go to 2).
5. **No child left**, or at a leaf: sample a *bonus* word from p and stop.



make is rejected, so its subtree (*a*, *our*) is skipped. Remove its share from p : the other children's p grows.

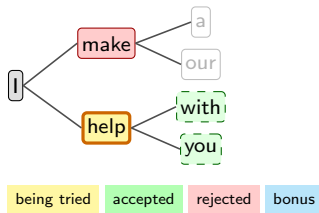


EAGLE inference (4/4): accept or reject

Walk down the tree with the vanilla rule (p : target, $\hat{p}$: draft).

Verify the tree

1. Start at the last accepted word.
2. Try its next child c : accept with probability $\min\left(1, \frac{p(c)}{\hat{p}(c)}\right)$.
3. **Accepted:** output c , move down to c , go to 2.
4. **Rejected:** $p \leftarrow \text{norm}(\max(0, p - \hat{p}))$, try the next sibling (go to 2).
5. **No child left**, or at a leaf: sample a *bonus* word from p and stop.



Try the sibling *help*: now $p = 0.5 \geq \hat{p} = 0.3$, so it is accepted with probability 1.



EAGLE inference (4/4): accept or reject

Walk down the tree with the vanilla rule (p : target, $\hat{p}$: draft).

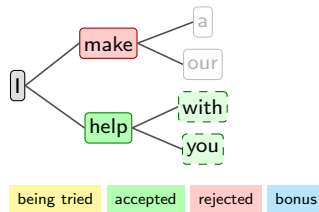
Verify the tree

1. Start at the last accepted word.
2. Try its next child c : accept with probability $\min\left(1, \frac{p(c)}{\hat{p}(c)}\right)$.

3. Accepted: output c , move down to c , go to 2.

4. Rejected: $p \leftarrow \text{norm}(\max(0, p - \hat{p}))$, try the next sibling (go to 2).

5. No child left, or at a leaf: sample a *bonus* word from p and stop.



help is accepted: output it and move down to *help*.

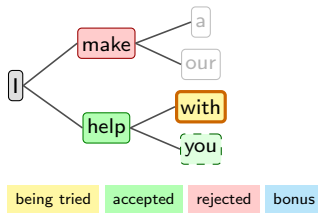


EAGLE inference (4/4): accept or reject

Walk down the tree with the vanilla rule (p : target, $\hat{p}$: draft).

Verify the tree

1. Start at the last accepted word.
2. Try its next child c : accept with probability $\min\left(1, \frac{p(c)}{\hat{p}(c)}\right)$.
3. **Accepted:** output c , move down to c , go to 2.
4. **Rejected:** $p \leftarrow \text{norm}(\max(0, p - \hat{p}))$, try the next sibling (go to 2).
5. **No child left**, or at a leaf: sample a *bonus* word from p and stop.



Try *with*: $p = 0.1$, $\hat{p} = 0.4$, so accept with probability 0.25. The random draw says no.

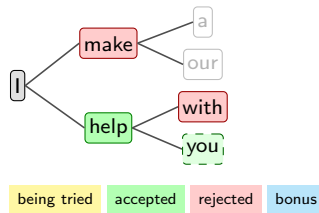


EAGLE inference (4/4): accept or reject

Walk down the tree with the vanilla rule (p : target, $\hat{p}$: draft).

Verify the tree

1. Start at the last accepted word.
2. Try its next child c : accept with probability $\min\left(1, \frac{p(c)}{\hat{p}(c)}\right)$.
3. **Accepted:** output c , move down to c , go to 2.
4. **Rejected:** $p \leftarrow \text{norm}\left(\max(0, p - \hat{p})\right)$, try the next sibling (go to 2).
5. **No child left**, or at a leaf: sample a *bonus* word from p and stop.



with is rejected: update p at this position and try the next sibling.

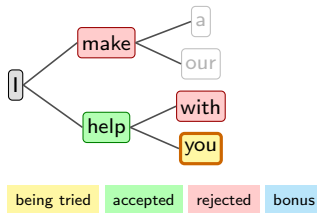


EAGLE inference (4/4): accept or reject

Walk down the tree with the vanilla rule (p : target, $\hat{p}$: draft).

Verify the tree

1. Start at the last accepted word.
2. Try its next child c : accept with probability $\min\left(1, \frac{p(c)}{\hat{p}(c)}\right)$.
3. **Accepted:** output c , move down to c , go to 2.
4. **Rejected:** $p \leftarrow \text{norm}(\max(0, p - \hat{p}))$, try the next sibling (go to 2).
5. **No child left**, or at a leaf: sample a *bonus* word from p and stop.



Try *you*: $p = 0.7 \geq \hat{p} = 0.35$, so it is accepted.

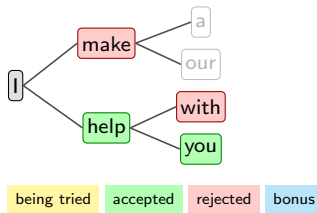


EAGLE inference (4/4): accept or reject

Walk down the tree with the vanilla rule (p : target, $\hat{p}$: draft).

Verify the tree

1. Start at the last accepted word.
2. Try its next child c : accept with probability $\min\left(1, \frac{p(c)}{\hat{p}(c)}\right)$.
3. **Accepted:** output c , move down to c , go to 2.
4. **Rejected:** $p \leftarrow \text{norm}(\max(0, p - \hat{p}))$, try the next sibling (go to 2).
5. **No child left**, or at a leaf: sample a *bonus* word from p and stop.



you is accepted: output it and move down. *you* is a leaf.

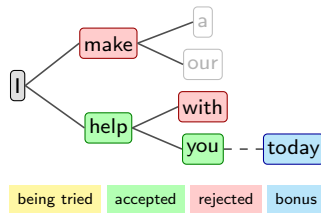


EAGLE inference (4/4): accept or reject

Walk down the tree with the vanilla rule (p : target, $\hat{p}$: draft).

Verify the tree

1. Start at the last accepted word.
2. Try its next child c : accept with probability $\min\left(1, \frac{p(c)}{\hat{p}(c)}\right)$.
3. **Accepted:** output c , move down to c , go to 2.
4. **Rejected:** $p \leftarrow \text{norm}(\max(0, p - \hat{p}))$, try the next sibling (go to 2).
5. **No child left**, or at a leaf: sample a *bonus* word from p and stop.



At a leaf: the target's p after *you* is already computed, so sample a **bonus** word, *today*. **3 words (help you today) from 1 target pass.**

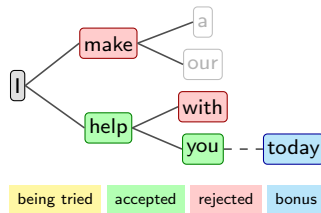


EAGLE inference (4/4): accept or reject

Walk down the tree with the vanilla rule (p : target, $\hat{p}$: draft).

Verify the tree

1. Start at the last accepted word.
2. Try its next child c : accept with probability $\min\left(1, \frac{p(c)}{\hat{p}(c)}\right)$.
3. **Accepted:** output c , move down to c , go to 2.
4. **Rejected:** $p \leftarrow \text{norm}(\max(0, p - \hat{p}))$, try the next sibling (go to 2).
5. **No child left**, or at a leaf: sample a *bonus* word from p and stop.

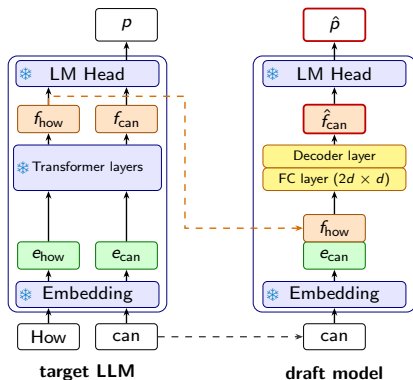


Lossless: ≥ 1 word per round, with exactly the target's distribution.
Next round drafts from the target's **true** features of *help*, *you*.



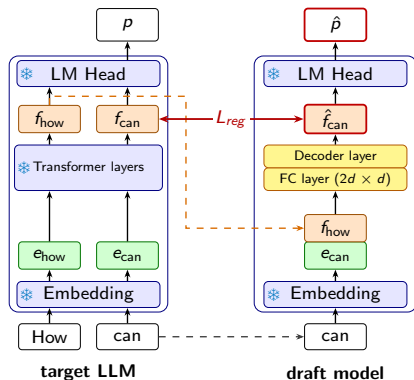
EAGLE training: two losses

Run the target on training text and record its features. The draft learns to **copy the target's next step**.



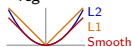
EAGLE training: two losses

Run the target on training text and record its features. The draft learns to **copy the target's next step**.



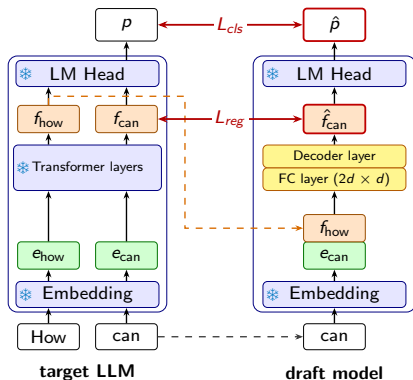
- **Feature loss:** guessed feature close to the real one.

$$L_{\text{reg}} = \text{SmoothL1}(f, \hat{f})$$



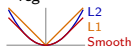
EAGLE training: two losses

Run the target on training text and record its features. The draft learns to **copy the target's next step**.



- **Feature loss:** guessed feature close to the real one.

$$L_{reg} = \text{SmoothL1}(f, \hat{f})$$



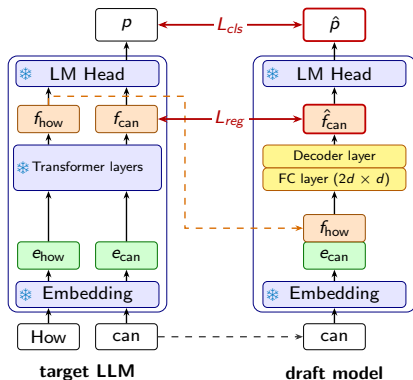
- **Word loss:** match the target's next-word probabilities, which decides acceptance.

$$L_{cls} = \text{CE}(p, \hat{p})$$



EAGLE training: two losses

Run the target on training text and record its features. The draft learns to **copy the target's next step**.



- **Feature loss:** guessed feature close to the real one.

$$L_{\text{reg}} = \text{SmoothL1}(f, \hat{f})$$



- **Word loss:** match the target's next-word probabilities, which decides acceptance.

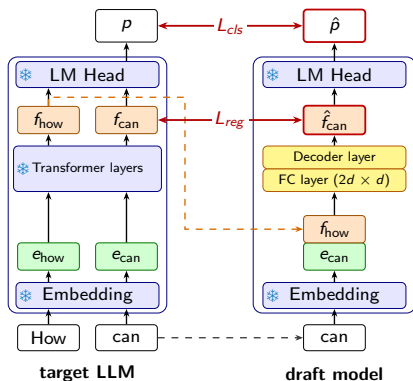
$$L_{\text{cls}} = \text{CE}(p, \hat{p})$$

- **Total:** $L = L_{\text{reg}} + 0.1 L_{\text{cls}}$. Only the **yellow** layers are trained.



EAGLE training: two losses

Run the target on training text and record its features. The draft learns to **copy the target's next step**.



- **Feature loss:** guessed feature close to the real one.

$$L_{\text{reg}} = \text{SmoothL1}(f, \hat{f})$$



- **Word loss:** match the target's next-word probabilities, which decides acceptance.

$$L_{\text{cls}} = \text{CE}(p, \hat{p})$$

- **Total:** $L = L_{\text{reg}} + 0.1 L_{\text{cls}}$. Only the **yellow** layers are trained.

- **Data:** 68k ShareGPT chats, with noise added to the input features so the draft copes with its own wrong guesses.

- **Cheap:** 1–2 days on 4 A100s (70B target).



EAGLE: main results

Speedup = how many times faster than normal one-word-at-a-time decoding. τ = average number of words accepted per target pass. V = Vicuna, LC2 = LLaMA2-Chat.

Model	$T = 0$ (greedy)				$T = 1$ (sampling)			
	MT τ	HumanEval	GSM8K	Alpaca	MT τ	HumanEval	GSM8K	Alpaca
V 7B	3.94	3.33×	3.01×	2.79×	3.17	2.39×	2.34×	2.21×
V 13B	3.98	3.58×	3.08×	3.03×	3.20	2.65×	2.57×	2.45×
V 33B	3.68	3.67×	3.25×	2.97×	3.22	2.76×	2.77×	2.52×
LC2 7B	3.62	3.17×	2.91×	2.78×	3.30	2.61×	2.40×	2.29×
LC2 13B	3.90	3.76×	3.20×	3.01×	3.45	2.89×	2.82×	2.66×
LC2 70B	3.81	3.52×	3.03×	2.97×	3.46	2.92×	2.74×	2.65×



EAGLE: main results

Speedup = how many times faster than normal one-word-at-a-time decoding. τ = average number of words accepted per target pass. V = Vicuna, LC2 = LLaMA2-Chat.

Model	$T = 0$ (greedy)				$T = 1$ (sampling)			
	MT τ	HumanEval	GSM8K	Alpaca	MT τ	HumanEval	GSM8K	Alpaca
V 7B	3.94	3.33×	3.01×	2.79×	3.17	2.39×	2.34×	2.21×
V 13B	3.98	3.58×	3.08×	3.03×	3.20	2.65×	2.57×	2.45×
V 33B	3.68	3.67×	3.25×	2.97×	3.22	2.76×	2.77×	2.52×
LC2 7B	3.62	3.17×	2.91×	2.78×	3.30	2.61×	2.40×	2.29×
LC2 13B	3.90	3.76×	3.20×	3.01×	3.45	2.89×	2.82×	2.66×
LC2 70B	3.81	3.52×	3.03×	2.97×	3.46	2.92×	2.74×	2.65×

- ▶ About 3× faster with greedy decoding, for every model size. Best on code (HumanEval), which is very predictable.



EAGLE: main results

Speedup = how many times faster than normal one-word-at-a-time decoding. τ = average number of words accepted per target pass. V = Vicuna, LC2 = LLaMA2-Chat.

Model	$T = 0$ (greedy)				$T = 1$ (sampling)			
	MT τ	HumanEval	GSM8K	Alpaca	MT τ	HumanEval	GSM8K	Alpaca
V 7B	3.94	3.33×	3.01×	2.79×	3.17	2.39×	2.34×	2.21×
V 13B	3.98	3.58×	3.08×	3.03×	3.20	2.65×	2.57×	2.45×
V 33B	3.68	3.67×	3.25×	2.97×	3.22	2.76×	2.77×	2.52×
LC2 7B	3.62	3.17×	2.91×	2.78×	3.30	2.61×	2.40×	2.29×
LC2 13B	3.90	3.76×	3.20×	3.01×	3.45	2.89×	2.82×	2.66×
LC2 70B	3.81	3.52×	3.03×	2.97×	3.46	2.92×	2.74×	2.65×

- ▶ About 3× faster with greedy decoding, for every model size. Best on code (HumanEval), which is very predictable.
- ▶ Still 2.2–2.9× when sampling ($T=1$), and still **lossless**.



EAGLE: serving many users at once (batching)

Batching = the target LLM handles several requests in one pass (MT-bench, $T=0$).

Batch size	V 7B	LC2 70B
1	2.90×	3.01×
2	2.87×	2.81×
3	2.65×	2.50×
4	2.76×	2.40×
Max throughput	1.97×	1.99×



EAGLE: serving many users at once (batching)

Batching = the target LLM handles several requests in one pass (MT-bench, $T=0$).

Batch size	V 7B	LC2 70B
1	2.90×	3.01×
2	2.87×	2.81×
3	2.65×	2.50×
4	2.76×	2.40×
Max throughput	1.97×	1.99×

- ▶ The speedup **drops** as the batch grows: the GPU is busier, so checking extra guessed words is no longer free.



EAGLE: serving many users at once (batching)

Batching = the target LLM handles several requests in one pass (MT-bench, $T=0$).

Batch size	V 7B	LC2 70B
1	2.90×	3.01×
2	2.87×	2.81×
3	2.65×	2.50×
4	2.76×	2.40×
Max throughput	1.97×	1.99×

- ▶ The speedup **drops** as the batch grows: the GPU is busier, so checking extra guessed words is no longer free.
- ▶ Total throughput still about **doubles**.



EAGLE-2

Faster Inference of Language Models with Dynamic Draft Trees

Li, Wei, Zhang, Zhang — EMNLP 2024



EAGLE-2: the problem with a fixed tree

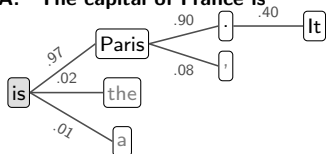
EAGLE uses the **same tree shape for every input**. But how easy the next words are to guess depends on the **context**.



EAGLE-2: the problem with a fixed tree

EAGLE uses the **same tree shape for every input**. But how easy the next words are to guess depends on the **context**.

A: “The capital of France is”



(illustration: the *same* tree shape; numbers = draft's confidence in each word)

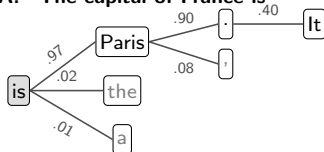
- ▶ **A** (easy): “Paris.” is almost certain. The extra siblings are wasted, and a *deeper* tree would get more words accepted.



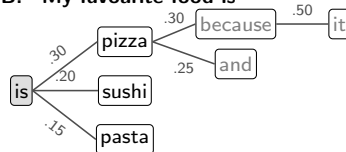
EAGLE-2: the problem with a fixed tree

EAGLE uses the **same tree shape for every input**. But how easy the next words are to guess depends on the **context**.

A: "The capital of France is"



B: "My favourite food is"



(illustration: the *same* tree shape; numbers = draft's confidence in each word)

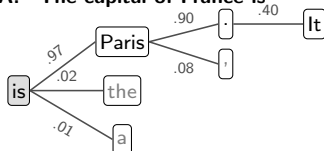
- ▶ **A** (easy): "Paris." is almost certain. The extra siblings are wasted, and a *deeper* tree would get more words accepted.
- ▶ **B** (hard): many answers are plausible. Deep guesses are rarely right, and a *wider* first level would help more.



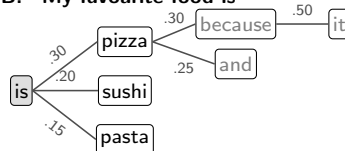
EAGLE-2: the problem with a fixed tree

EAGLE uses the **same tree shape for every input**. But how easy the next words are to guess depends on the **context**.

A: “The capital of France is”



B: “My favourite food is”

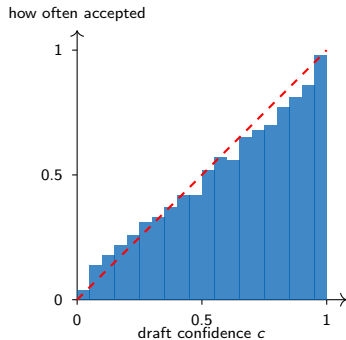


(illustration: the *same* tree shape; numbers = draft's confidence in each word)

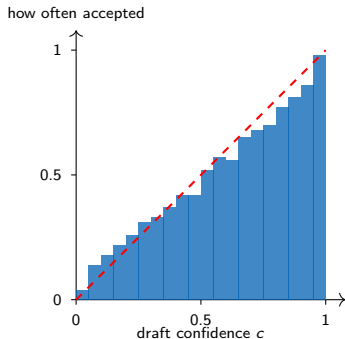
- ▶ **A** (easy): “Paris.” is almost certain. The extra siblings are wasted, and a *deeper* tree would get more words accepted.
- ▶ **B** (hard): many answers are plausible. Deep guesses are rarely right, and a *wider* first level would help more.
- ▶ The EAGLE-2 paper measured this: the acceptance rate at the *same* tree position varies a lot from input to input.



EAGLE-2 key observation: the draft knows when it is right



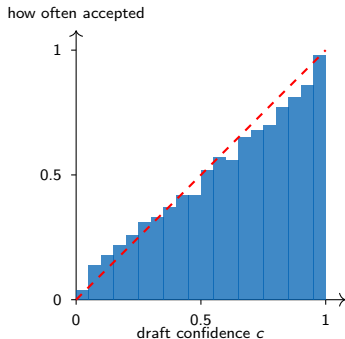
EAGLE-2 key observation: the draft knows when it is right



- ▶ When the draft is **sure**, its guess is almost always **right**.



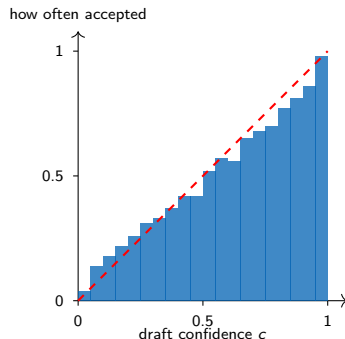
EAGLE-2 key observation: the draft knows when it is right



- ▶ When the draft is **sure**, its guess is almost always **right**.
- ▶ A guess's **value** V = its confidence $\times$ its parents' confidences
e.g. is (0.6) $\rightarrow$ a (0.7): $V = 0.42$



EAGLE-2 key observation: the draft knows when it is right



- ▶ When the draft is **sure**, its guess is almost always **right**.
- ▶ A guess's **value** V = its confidence $\times$ its parents' confidences
e.g. is (0.6) $\rightarrow$ a (0.7): $V = 0.42$

Idea

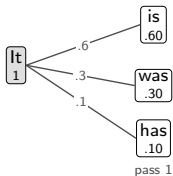
Grow the tree where V is high, **separately for each input**.



EAGLE-2 algorithm (1/2): expansion

Grow the tree one level per draft pass. At each level, expand only the ***k* most valuable new nodes**.

- Pass 1: the draft proposes *is*, *was*, *has*.

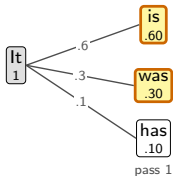


edge: confidence c ; node: value V . **expanded** ($k=2$ here)



EAGLE-2 algorithm (1/2): expansion

Grow the tree one level per draft pass. At each level, expand only the **k most valuable new nodes**.



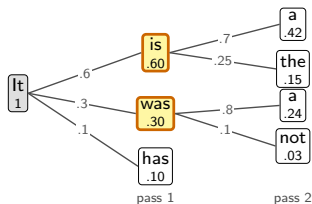
- Pass 1: the draft proposes *is*, *was*, *has*.
- Expand only the top-2 by V : *is* (.60), *was* (.30). *has* (.10) gets no children.

edge: confidence c ; node: value V . expanded ($k=2$ here)



EAGLE-2 algorithm (1/2): expansion

Grow the tree one level per draft pass. At each level, expand only the **most valuable new nodes**.



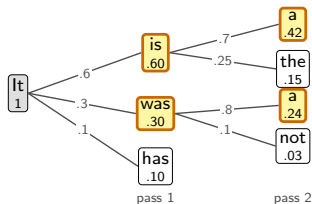
edge: confidence c ; node: value V . **expanded** ($k=2$ here)

- Pass 1: the draft proposes *is*, *was*, *has*.
- Expand only the top-2 by V : *is* (.60), *was* (.30). *has* (.10) gets no children.
- Pass 2: *is* and *was* go through the draft **together, in one pass**. Child value = parent value $\times$ confidence.



EAGLE-2 algorithm (1/2): expansion

Grow the tree one level per draft pass. At each level, expand only the **most valuable new nodes**.



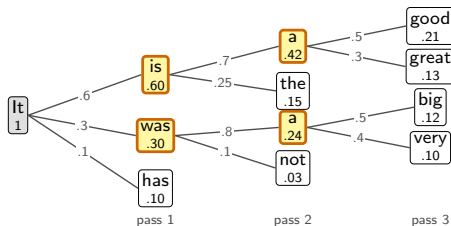
edge: confidence c ; node: value V . **expanded** ($k=2$ here)

- Pass 1: the draft proposes *is*, *was*, *has*.
- Expand only the top-2 by V : *is* (.60), *was* (.30). *has* (.10) gets no children.
- Pass 2: *is* and *was* go through the draft **together, in one pass**. Child value = parent value $\times$ confidence.
- Again keep the top-2 of the new level: *a* (.42) and *a* (.24).



EAGLE-2 algorithm (1/2): expansion

Grow the tree one level per draft pass. At each level, expand only the **most valuable new nodes**.



edge: confidence c ; node: value V . **expanded** ($k=2$ here)

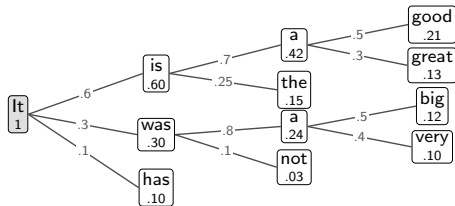
- Pass 1: the draft proposes *is*, *was*, *has*.
- Expand only the top-2 by V : *is* (.60), *was* (.30). *has* (.10) gets no children.
- Pass 2: *is* and *was* go through the draft **together, in one pass**. Child value = parent value $\times$ confidence.
- Again keep the top-2 of the new level: *a* (.42) and *a* (.24).
- Pass 3, and so on.

One draft pass per level, however wide the tree (paper: 6 levels, $k=10$). Easy inputs grow deep, and hard ones stay shallow.



EAGLE-2 algorithm (2/2): reranking

Expansion made many candidate words. Keep only the m **best by value** V , from *any* level, including nodes that were not expanded.

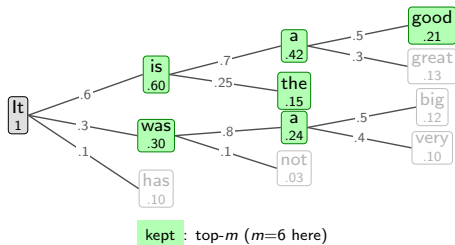


kept : top- m ($m=6$ here)



EAGLE-2 algorithm (2/2): reranking

Expansion made many candidate words. Keep only the m **best by value** V , from *any* level, including nodes that were not expanded.

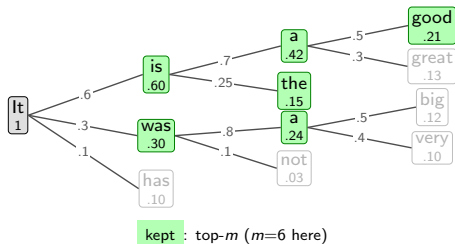


- Ranking: is .60, a .42, was .30, a .24, good .21, **the .15** — great .13, ...



EAGLE-2 algorithm (2/2): reranking

Expansion made many candidate words. Keep only the m **best by value** V , from *any* level, including nodes that were not expanded.

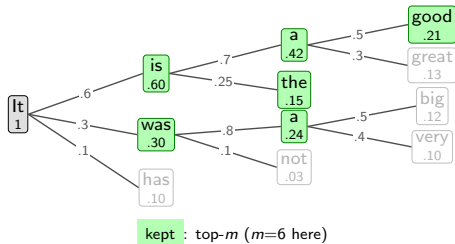


- ▶ Ranking: is .60, a .42, was .30, a .24, good .21, **the .15** — great .13, ...
- ▶ The unexpanded *the* (.15) beats the deeper *great* (.13).



EAGLE-2 algorithm (2/2): reranking

Expansion made many candidate words. Keep only the m **best by value** V , from *any* level, including nodes that were not expanded.

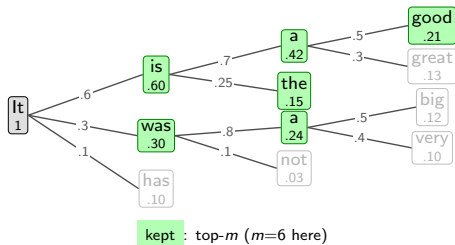


- ▶ Ranking: is .60, a .42, was .30, a .24, good .21, **the .15** — great .13, ...
- ▶ The unexpanded *the* (.15) beats the deeper *great* (.13).
- ▶ A child's V is never larger than its parent's, so the kept words always form a **connected tree**.



EAGLE-2 algorithm (2/2): reranking

Expansion made many candidate words. Keep only the m **best by value** V , from *any* level, including nodes that were not expanded.



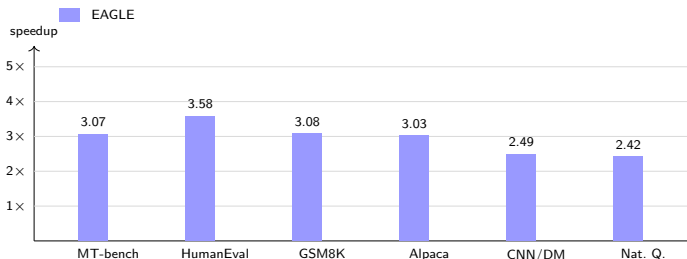
- ▶ Ranking: is .60, a .42, was .30, a .24, good .21, **the .15** — great .13, ...
- ▶ The unexpanded *the* (.15) beats the deeper *great* (.13).
- ▶ A child's V is never larger than its parent's, so the kept words always form a **connected tree**.

Then check the kept tree **exactly as in EAGLE** (tree attention + accept/reject). m = number of words checked per round.



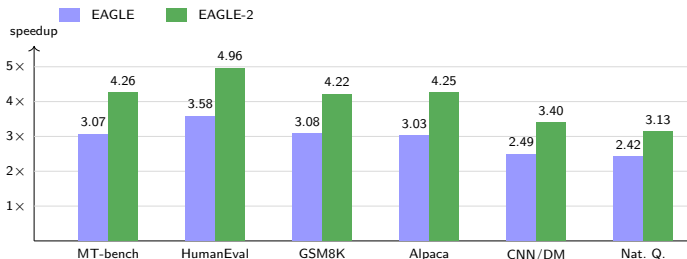
EAGLE-2: main results

Speedup with greedy decoding, Vicuna-13B (paper Table 1):



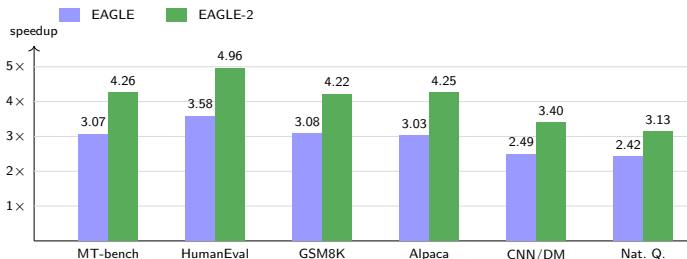
EAGLE-2: main results

Speedup with greedy decoding, Vicuna-13B (paper Table 1):



EAGLE-2: main results

Speedup with greedy decoding, Vicuna-13B (paper Table 1):

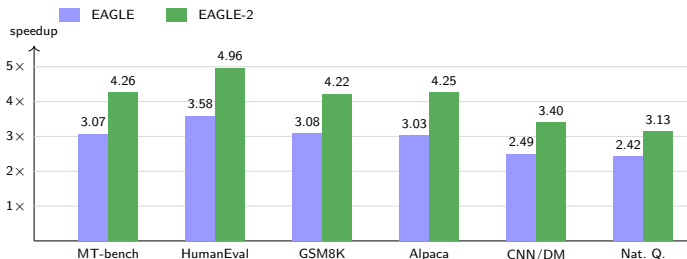


Mean over 6 datasets	Vicuna-13B		LLaMA2-Chat-13B	
	speedup	τ	speedup	τ
EAGLE	2.95×	3.82	3.09×	3.89
EAGLE-2	4.04×	4.65	4.10×	4.68



EAGLE-2: main results

Speedup with greedy decoding, Vicuna-13B (paper Table 1):



Mean over 6 datasets	Vicuna-13B		LLaMA2-Chat-13B	
	speedup	τ	speedup	τ
EAGLE	2.95×	3.82	3.09×	3.89
EAGLE-2	4.04×	4.65	4.10×	4.68

Overall 3.05–4.26 $\times$ faster than normal decoding and 20–40% faster than EAGLE, with the same draft model and still lossless.



EAGLE-3

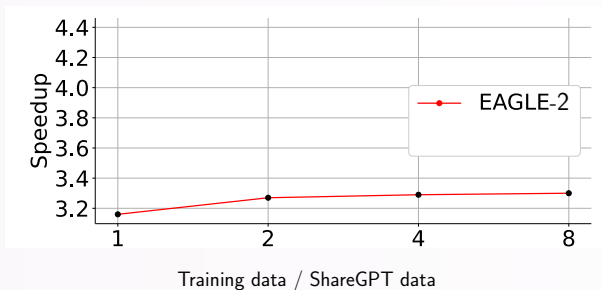
Scaling up Inference Acceleration of Large Language Models
via Training-Time Test

2025



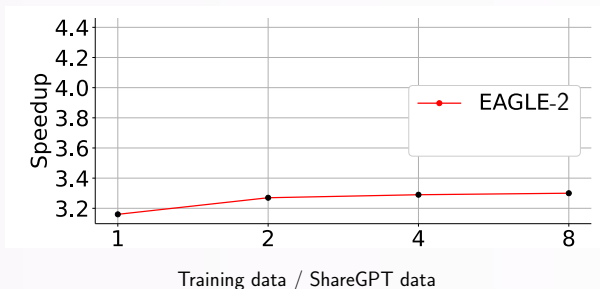
More training data, little extra speed

- ▶ EAGLE-2's draft architecture and inference cost stay fixed.
- ▶ Training data increases from $1\times$ to $8\times$ ShareGPT scale.



More training data, little extra speed

- ▶ EAGLE-2's draft architecture and inference cost stay fixed.
- ▶ Training data increases from $1\times$ to $8\times$ ShareGPT scale.



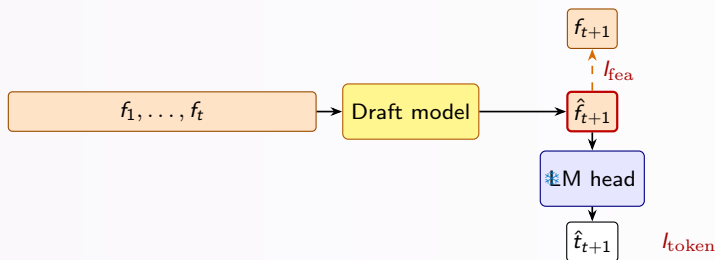
Observation

Only a small gain on LLaMA-3.1-8B and MT-Bench.



The extra constraint in the draft objective

- ▶ f_i is the target's top-layer state; $\hat{f}_{i+1}$ is its draft prediction.
- ▶ l_{fea} matches that state to the target. l_{token} supervises the token distribution after the LM head.

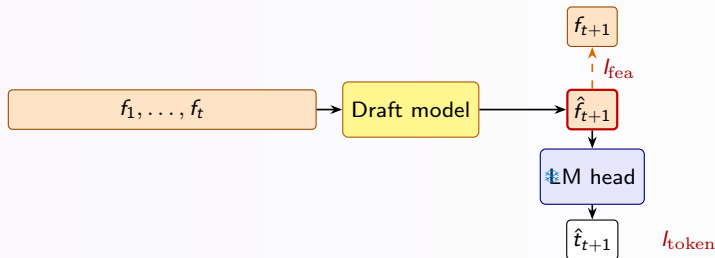


- ▶ Feature matching makes the output usable at the next draft step.
- ▶ It also constrains the representation used for token prediction.



The extra constraint in the draft objective

- ▶ f_i is the target's top-layer state; $\hat{f}_{i+1}$ is its draft prediction.
- ▶ l_{fea} matches that state to the target. l_{token} supervises the token distribution after the LM head.

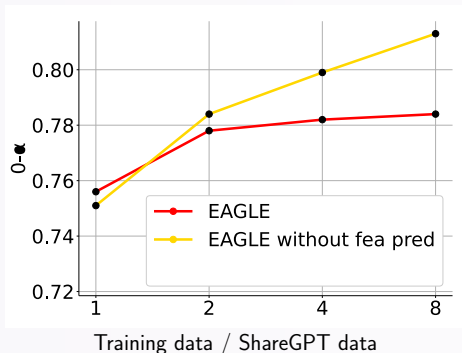


- ▶ Feature matching makes the output usable at the next draft step.
- ▶ It also constrains the representation used for token prediction.
- ▶ The next trial removes l_{fea} and keeps l_{token} .



Removing feature matching: the first token

- ▶ $\hat{a}_{t+1}$ is now an unconstrained draft state: it only needs to support the next-token prediction.
- ▶ $0-\alpha$ measures acceptance of the first draft token.



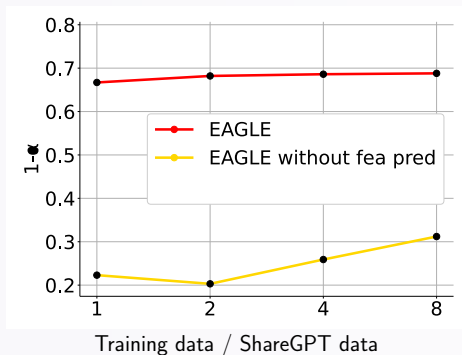
What changes

- ▶ l_{fea} removed.
- ▶ More training data.
- ▶ First-token acceptance improves substantially.



The same trial: the second token

- ▶ $1-\alpha$ measures second-token acceptance, conditional on the first token being accepted.



Observation

- ▶ Better first-token predictions.
- ▶ Very low second-token acceptance.
- ▶ More data alone does not repair the continuation.



The second step receives a different input

Training: first step

$$f_1, f_2, \dots, f_t$$

Target features only.

Inference: second step

$$f_1, f_2, \dots, f_t, \hat{a}_{t+1}$$

The draft's own output joins the context.

- ▶ After loss removal, $\hat{a}_{t+1}$ can be far from f_{t+1} .
- ▶ This continuation input was absent during training.



The second step receives a different input

Training: first step

$$f_1, f_2, \dots, f_t$$

Target features only.

Inference: second step

$$f_1, f_2, \dots, f_t, \hat{a}_{t+1}$$

The draft's own output joins the context.

- ▶ After loss removal, $\hat{a}_{t+1}$ can be far from f_{t+1} .
- ▶ This continuation input was absent during training.

The missing training case

Continuation from the draft's own unconstrained state.



EAGLE-3: three changes to the draft head

EAGLE-2

- ▶ Feature loss + token loss.
- ▶ First-step training.
- ▶ Top-layer target features.

EAGLE-3

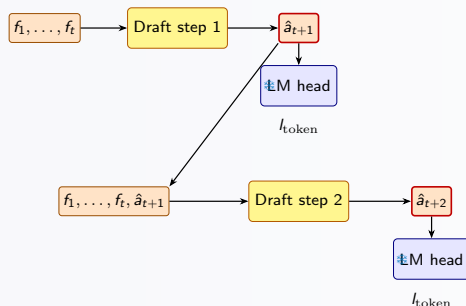
- ▶ Token loss only.
- ▶ Simulated continuation during training.
- ▶ Low-, middle-, and high-layer fusion.

- ▶ A single decoder layer still generates the draft states.
- ▶ EAGLE-2's dynamic tree and target verification carry over.



Training-time test: feeding the output back

- Token-only training learns the first prediction; continuation needs practice with the draft's own outputs.

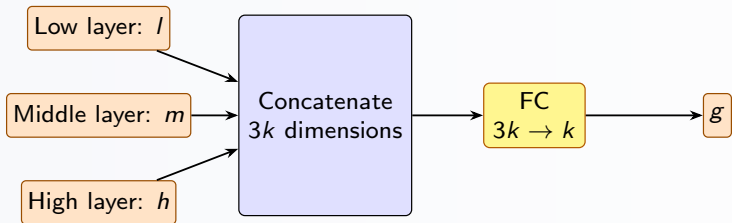


- The first output, $\hat{a}_{t+1}$, joins the context for the next training step.
- Both outputs pass through the shared LM head and receive token-loss supervision.
- Later steps now encounter the draft states used at inference.



Feature fusion replaces the top-layer input

- ▶ With a full-rank LM head, next-token logits uniquely fix the top-layer feature.
- ▶ Intermediate features offer other information for predicting further ahead.
- ▶ l, m, h are the low-, middle-, and high-layer target states, each of hidden width k .
- ▶ g is the fused target state, projected back to width k .



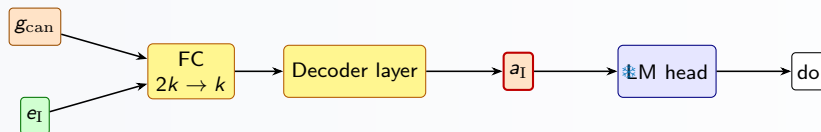
- ▶ Without l_{fea} , the draft input can use this fused representation.



The changed input: the first draft step

Prefix: “How can I”

- ▶ The target processes “How can” and samples “I”.
- ▶ g_{can} carries the fused context. e_I embeds the sampled “I”; a_I is the resulting draft state.

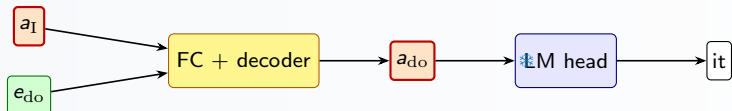


- ▶ (g_{can}, e_I) supplies context and the actual sampled token.
- ▶ The shared LM head turns a_I into the next-token distribution.



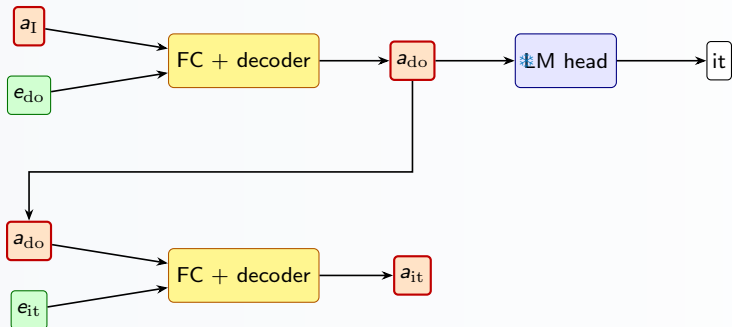
The changed input: continuation states

- ▶ The target has not processed “I”; g_I is unavailable.
- ▶ The draft substitutes its own state a_I and pairs it with e_{do} .



The changed input: continuation states

- ▶ The target has not processed “I”; g_I is unavailable.
- ▶ The draft substitutes its own state a_I and pairs it with e_{do} .



- ▶ The following input is (a_{do}, e_{it}) .
- ▶ Earlier target states remain in the cache.



Training-time test: the mixed context

- ▶ $g_1, \dots, g_t$ come from the target; $a_{t+1}, \dots, a_{t+j}$ come from j simulated draft steps.

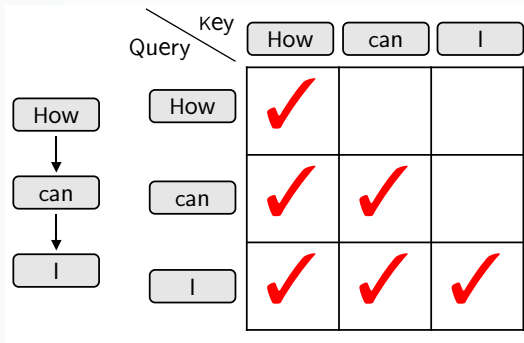
Context during a simulated continuation

$$\underbrace{g_1, g_2, \dots, g_t}_{\text{target states}}, \quad \underbrace{a_{t+1}, a_{t+2}, \dots, a_{t+j}}_{\text{draft states}}.$$

- ▶ Simulations start at every training position in parallel; each position grows its own continuation.
- ▶ A prediction may see its original prefix and earlier states on that branch.
- ▶ Self-attention is the only decoder operation that mixes context. Its mask changes to preserve these branches; the other decoder operations stay the same.



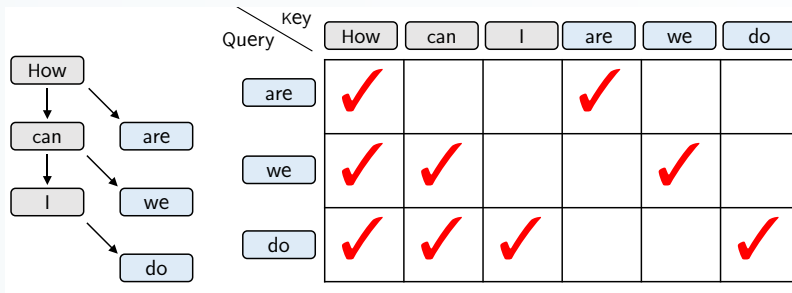
Training mask: the native step



- ▶ Gray tokens are the original training sequence; ticks mark visible keys for each query.
- ▶ The usual causal mask gives the prefixes "How", "How can", and "How can I".
- ▶ Three branches start from these prefixes: "are", "we", and "do".



Training mask: the first simulated step

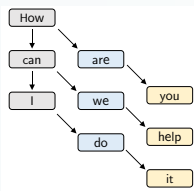


- ▶ Blue tokens are first-round predictions, fed back as new query states.
- ▶ “we” sees “How can” and its own state. “are” and “do” belong to different branches.
- ▶ Attention to the gray prefix stays causal; attention within the predicted round is diagonal.



Training mask: the second simulated step

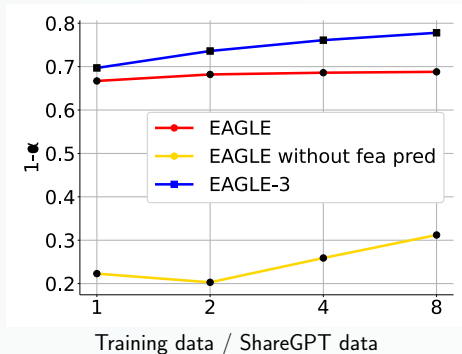
key Query	How	can	I	are	we	do	you	help	it
	you	✓		✓			✓		
help	✓	✓			✓			✓	
it	✓	✓	✓			✓			✓



- ▶ Yellow tokens are second-round predictions: “help” continues “How can we”.
- ▶ Its visible predicted states are “we” and “help”; the other branches stay invisible.
- ▶ Corresponding vector dot products give the predicted-round scores without a dense matrix multiplication.



Second-token acceptance after the changes



The completed design

- Token loss only.
- Training-time test.
- Multi-layer fusion.

Second-token acceptance now benefits from more data.



EAGLE-2 verification carries over

- ▶ Dynamic tree: depth $6 \rightarrow 8$; 10 nodes expanded per level; verification budgets 60/50/48 for 8/13/70B targets.
- ▶ Along one tree path, $\hat{T}_{j+1:j+k}$ contains k proposals after the prefix $T_{1:j}$. Here k is a draft length.
- ▶ At offset i , p_{j+i} and $\hat{p}_{j+i}$ are the target and draft distributions for the same accepted prefix.

The inherited sampling rule

Acceptance of $\hat{t}_{j+i}$:

$$\min\left(1, \frac{p_{j+i}(\hat{t}_{j+i})}{\hat{p}_{j+i}(\hat{t}_{j+i})}\right).$$

On rejection, the residual distribution is

$$\text{norm}(\max(0, p_{j+i} - \hat{p}_{j+i})).$$

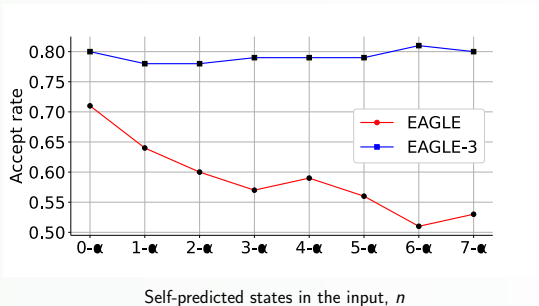
max clips negative entries; norm normalizes the mass.



Acceptance deeper into the continuation

- n - α measures next-token acceptance when the input contains n self-predicted states, conditional on all earlier draft tokens being accepted.

LLaMA-3.1-8B, MT-Bench; chain drafts for this measurement.



EAGLE: $f_1, \dots, f_i, \hat{f}_{i+1}, \dots, \hat{f}_{i+n}$,
EAGLE-3: $g_1, \dots, g_i, a_{i+1}, \dots, a_{i+n}$.



Separating the two design changes

τ : mean tokens generated per draft–verification cycle.
LLaMA-3.1-8B, MT-Bench.

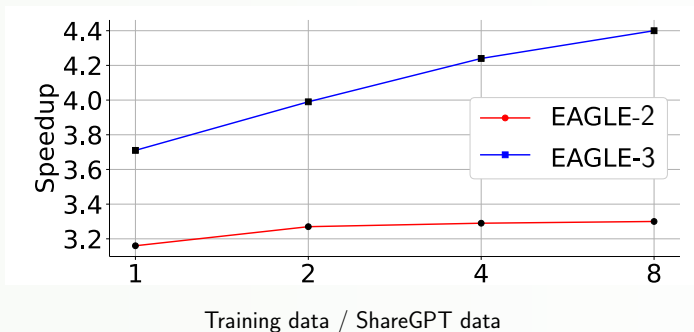
Draft design	Speedup	τ
EAGLE-2	3.16×	4.05
Token loss + training-time test	3.82×	5.37
+ multi-layer fusion	4.40×	6.13

- ▶ Middle row: loss removal *with* training-time test.
- ▶ Final row: multi-layer fusion added.



More data, after the draft-head changes

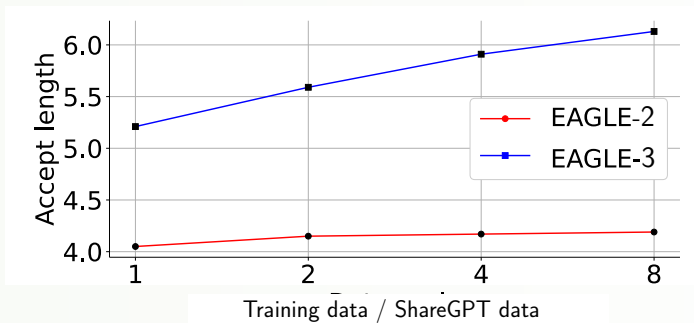
LLaMA-3.1-8B, MT-Bench; draft architecture fixed within each curve.



- EAGLE-3 gains substantially as the training set grows.



The same scaling experiment: accepted length

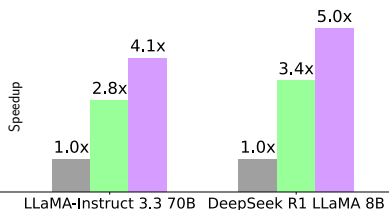
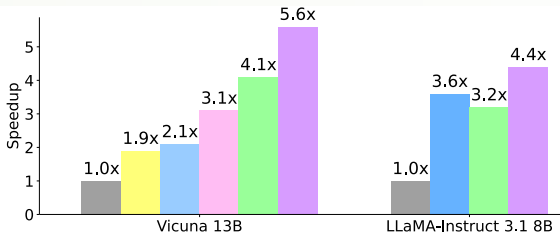


- More accepted tokens per verification cycle.
- The gain comes without a larger inference-time draft architecture.



Acceleration across chat and reasoning targets

Temperature 0, batch size 1; chat on MT-Bench, reasoning on GSM8K.



■ Vanilla ■ Speculative sampling ■ Medusa ■ HASS
■ EAGLE ■ EAGLE-2 ■ EAGLE-3



Five-task comparison: greedy decoding

Temperature 0; batch size 1. Means over MT-Bench, HumanEval, GSM8K, Alpaca, and CNN/Daily Mail.

Target	EAGLE-2	EAGLE-3	EAGLE-3 τ
Vicuna-13B	4.22×	5.51×	6.62
LLaMA-3.1-8B	3.23×	4.44×	6.23
LLaMA-3.3-70B	2.85×	4.12×	5.88
DeepSeek-R1-8B	3.26×	4.16×	5.84

Vicuna-13B on HumanEval: $6.47\times$ speedup, $\tau = 7.54$.



Five-task comparison: stochastic sampling

Temperature 1; batch size 1; the same five tasks.

Target	EAGLE-2	EAGLE-3	EAGLE-3 τ
Vicuna-13B	3.76×	4.65×	5.67
LLaMA-3.1-8B	2.80×	3.45×	4.92
LLaMA-3.3-70B	2.65×	3.95×	5.66
DeepSeek-R1-8B	2.77×	3.52×	4.89

- Acceleration also improves under sampling.



EAGLE-3 in SGLang

LLaMA-3.1-8B, MT-Bench; SGLang v0.4.4, one H100.

Chain length 3. Entries: throughput relative to target-only decoding.

Batch size	2	8	16	32	64
EAGLE	1.40×	1.23×	1.02×	0.94×	0.99×
EAGLE-3	1.81×	1.62×	1.48×	1.32×	1.38×

- ▶ Batch 1: target 158.34, EAGLE-2 244.10, EAGLE-3 373.25 tokens/s.
- ▶ Batch 64: 38% more throughput.



EAGLE-3 in vLLM

LLaMA-3.1-8B, MT-Bench; vLLM. Maximum chain length 2.
Entries: throughput relative to target-only decoding.

Batch size	2	8	16	32	56
EAGLE	1.30×	1.21×	1.10×	0.93×	0.71×
EAGLE-3	1.75×	1.58×	1.49×	1.36×	1.01×

- ▶ Batch 32: 36% more throughput.
- ▶ Gain narrows as the batch grows.



DFlash

Block Diffusion for Flash Speculative Decoding

2026



The remaining cost: sequential drafting

- ▶ EAGLE-3 still drafts sequentially; successive tree depths need earlier outputs.
- ▶ More draft layers add cost, while acceptance can saturate.
- ▶ γ is the number of proposals. T_{draft} and T_{verify} are the two parts of a cycle.
- ▶ $\tau \in [1, \gamma + 1]$ is its mean accepted length, including the target's correction or bonus token.
- ▶ L is latency per generated token; η compares it with target-only latency L_{target} .

$$L = \frac{T_{\text{draft}} + T_{\text{verify}}}{\tau}, \quad \eta = L_{\text{target}}/L.$$

- ▶ Higher acceptance helps; lower drafting time helps as well.



A chain of passes, or one block pass

- ▶ t_{step} is the time for one autoregressive draft pass.
- ▶ t_{parallel} is the time for one block-generation pass, with all masked positions decoded together.

Autoregressive drafting

$$T_{\text{draft}} = \gamma t_{\text{step}}.$$

One pass for each next token.

Parallel diffusion drafting

$$T_{\text{draft}} = t_{\text{parallel}}.$$

Unknown positions predicted together.

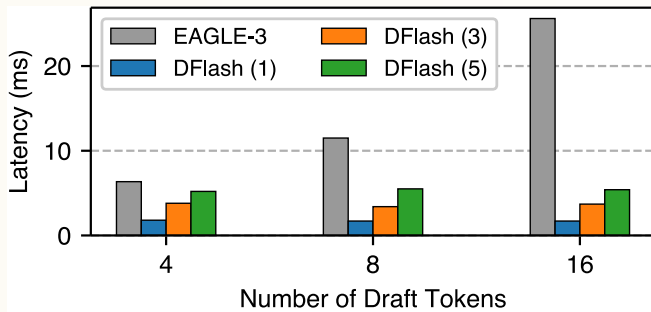
$$t_{\text{parallel}} \ll \gamma t_{\text{step}}.$$

Comparable model sizes; moderate blocks on a GPU.

- ▶ Lower draft latency leaves room for a deeper drafter.



Measured drafting cost



Bar colors: DFlash with 1, 3, or 5 layers; EAGLE-3 with one layer.

- Five-layer DFlash at 16 tokens is faster than one-layer EAGLE-3 at 8.



Parallel prediction still needs good conditioning

- ▶ Standalone diffusion generation often needs several denoising passes.
- ▶ A small drafter without target features predicts the continuation from scratch.

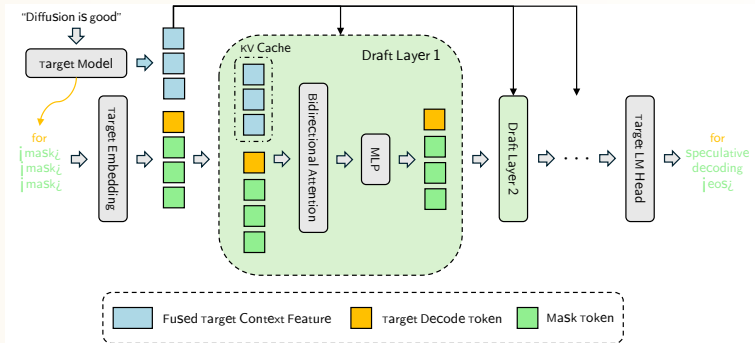
Five-layer diffusion drafter without target features

GSM8K, temperature 0: $2.83\times$ speedup, $\tau = 3.38$.

- ▶ The target's hidden states supply the contextual guidance for DFlash's block predictions.



DFlash: one denoising step for the whole block



- ▶ Target prefill supplies the clean anchor “for” and context features (blue).
- ▶ Bidirectional block attention sees the anchor and masks together; target features enter every draft layer as cached keys and values.
- ▶ **Single-step block denoising:** one forward pass predicts all three masked positions in parallel.



Fusing the target context

- ▶ $\mathbf{H}^{(l)}$ contains hidden states from target layer l . Five layers, $l_1, \dots, l_5$, span shallow to deep.
- ▶ Each state has width D . Concatenation gives width $5D$, reduced by the learned projection $W_c \in \mathbb{R}^{D \times 5D}$.
- ▶ RMSNorm produces the fused target context $\mathbf{H}_t$.

$$\mathbf{H}_t = \text{RMSNorm}\left(W_c[\mathbf{H}^{(l_1)}; \dots; \mathbf{H}^{(l_5)}]\right).$$

- ▶ The target embedding and LM head are shared and frozen.
- ▶ Context enters each layer as keys and values, rather than only through the input embedding.



KV injection at draft layer i

- ▶ $\mathbf{H}_d$ contains the current draft states; $\mathbf{H}_t$ is the fused target context.
- ▶ $[\cdot; \cdot]_{\text{seq}}$ joins token positions, so target context adds attention keys and values without adding draft queries.

$$\mathbf{Q}_i = W_i^Q \mathbf{H}_d,$$

$$\mathbf{K}_i = [W_i^K \mathbf{H}_t; W_i^K \mathbf{H}_d]_{\text{seq}},$$

$$\mathbf{V}_i = [W_i^V \mathbf{H}_t; W_i^V \mathbf{H}_d]_{\text{seq}}.$$

- ▶ Queries come from draft positions; target context supplies extra KV entries.
- ▶ Target states bypass draft attention updates and feed-forward layers.
- ▶ Projected target keys and values are cached across cycles.



Ablation: parallelism and conditioning

Qwen3-4B, GSM8K; H200, temperature 0.
Five-layer drafters; diffusion block size 8.

Drafting	Target conditioning	τ	Speedup
Autoregressive	Input fusion	4.2	2.1×
Autoregressive	KV injection	4.8	2.4×
Diffusion	Input fusion	3.5	2.9×
Diffusion	KV injection	4.2	3.3×

- ▶ KV injection improves acceptance in both draft families.
- ▶ Parallel drafting gains speed even at lower accepted length.



The next block begins with a clean token

`block_size` includes the clean anchor. The other `block_size - 1` positions are proposals.

Draft. The anchor stays clean. One pass predicts all masked positions.

Verify. The target checks the proposal and keeps an accepted prefix, then supplies a correction or bonus token.

Continue. That target-produced token anchors the next masked block.



The next block begins with a clean token

`block_size` includes the clean anchor. The other `block_size - 1` positions are proposals.

Draft. The anchor stays clean. One pass predicts all masked positions.

Verify. The target checks the proposal and keeps an accepted prefix, then supplies a correction or bonus token.

Continue. That target-produced token anchors the next masked block.



The next block begins with a clean token

block_size includes the clean anchor. The other block_size - 1 positions are proposals.

Draft. The anchor stays clean. One pass predicts all masked positions.

Verify. The target checks the proposal and keeps an accepted prefix, then supplies a correction or bonus token.

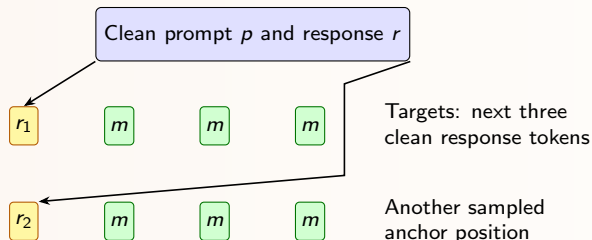
Continue. That target-produced token anchors the next masked block.

- ▶ Newly available target features extend the draft context.
- ▶ Earlier projected target KV entries are reused.



Training blocks match the inference inputs

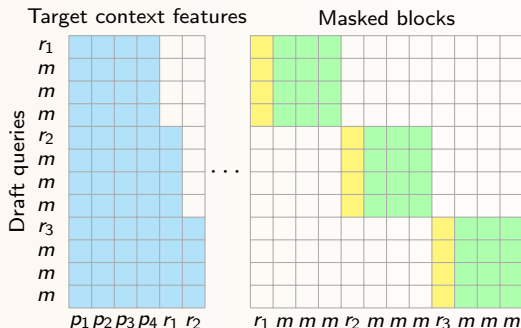
- r_1, r_2 illustrate two sampled response anchors; m is a mask token.



- The clean prompt and response pass through the frozen target to supply context features.
- Each sampled anchor starts a block: the anchor stays clean and the remaining positions are masked.
- Each mask is supervised by its clean response token. Random anchors expose varied continuations.



Training attention: context plus one block



■ Target context ■ Clean anchor ■ Masked position □ Invisible

- ▶ Target-context keys stop before each anchor. For r_2 , these are $p_1, \dots, p_4, r_1$, matching inference.
- ▶ Bidirectional attention within one block; no attention between blocks.
- ▶ FlexAttention trains the concatenated blocks jointly in one pass.



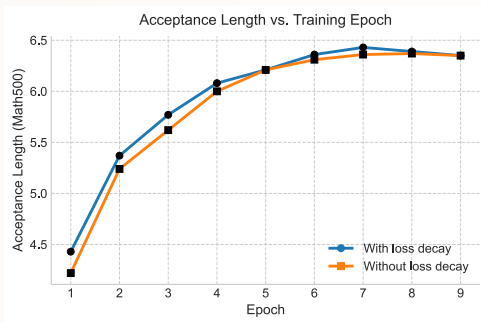
Training loss favors earlier positions

An early error ends the accepted prefix

w_k weights cross-entropy at block position k . Here γ controls the decay rate.

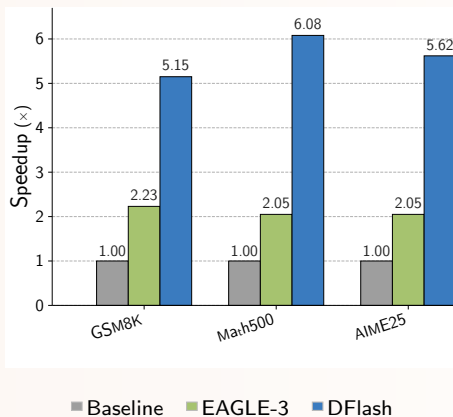
$$w_k = \exp\left(-\frac{k-1}{\gamma}\right).$$

- Earlier positions receive more weight; convergence is faster and acceptance is higher.



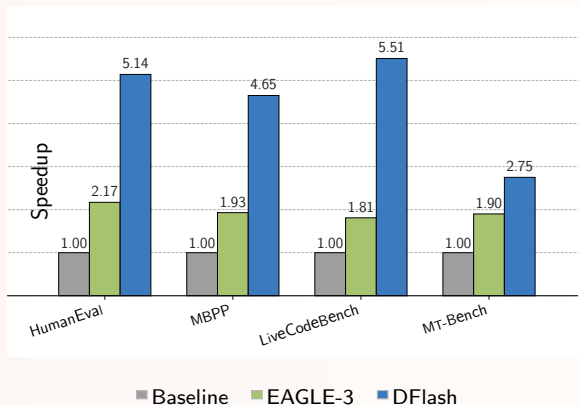
Math benchmarks: end-to-end acceleration

Qwen3-8B, thinking off; batch 1, temperature 0; Transformers, H200.
EAGLE-3: 60-token tree. DFlash: block size 16.



Code and chat: the same comparison

Qwen3-8B, thinking off; batch 1, temperature 0; Transformers, H200.
EAGLE-3: 60-token tree. DFlash: block size 16.



Draft budget and stochastic sampling

Qwen3-8B, thinking off; batch 1; Transformers, H200.

Generation limit: 2048 tokens.

Means: GSM8K, Math500, AIME25; HumanEval, MBPP, Live-CodeBench; MT-Bench.

Method	Temperature 0		Temperature 1	
	Speedup	τ	Speedup	τ
EAGLE-3 (tree 16)	1.76×	2.96	1.68×	2.83
EAGLE-3 (tree 60)	2.02×	3.40	1.88×	3.26
DFlash (block 16)	4.86×	6.49	4.03×	5.48

- Both acceptance length and speedup improve at both temperatures.



Draft depth: acceptance and speed

Math500, temperature 0, H200; same training data.
Five target-feature layers; block size 16.

Draft layers	τ	Speedup
3	5.64	4.69×
5	5.99	4.71×
8	6.33	4.64×

- More depth improves τ ; extra layer cost can offset that gain.
- Five layers give the highest speedup in this comparison.



DFlash in SGLang

Qwen3-8B, Math500; one B200; block size 16.
FlashAttention-4; Spec-v2 scheduling overlap.
Concurrency: simultaneous requests.

Concurrency	Baseline tokens/s	DFlash tokens/s	Speedup
1	230	1175	5.1×
8	1666	7485	4.5×
16	3133	12268	3.9×
32	5694	16076	2.8×

- Mean acceptance length: 8.01.
- Concurrency 32: 2.8× throughput.



Matched training data: DFlash and EAGLE-3

LLaMA-3.1-8B, GSM8K; same training data.
SGLang, one B200, Flashinfer; Spec-v1, no scheduling overlap.

Method	Speedup at concurrency				Mean τ
	1	8	16	32	
EAGLE-3 (tree 10)	1.6×	1.4×	1.2×	1.0×	3.49
EAGLE-3 (tree 60)	1.9×	1.3×	0.9×	0.6×	4.55
DFlash (block 10)	2.4×	2.1×	1.8×	1.6×	4.32

- ▶ EAGLE-3: released checkpoint, seven draft steps, 10-node expansion.
- ▶ DFlash is faster than both trees; its accepted prefix is slightly shorter than with tree size 60.



Qwen3.5-9B. Entries: tokens/s; relative speedup in parentheses.
Concurrency: simultaneous requests.

Concurrency	Math500	HumanEval	MT-Bench
1	849 (4.0 $\times$)	969 (4.6 $\times$)	627 (3.0 $\times$)
8	5096 (3.2 $\times$)	5434 (3.4 $\times$)	3536 (2.2 $\times$)
16	7669 (2.5 $\times$)	8131 (2.7 $\times$)	5297 (1.7 $\times$)
32	9836 (1.9 $\times$)	10258 (2.1 $\times$)	6787 (1.3 $\times$)

- Absolute throughput rises with concurrency; relative gains narrow.



Speculative Decoding: Performance or Illusion?

Xiaoxuan Liu, Jiaxiang Yu, Jongseok Park, Ion Stoica, Alvin Cheung
UC Berkeley — MLSys 2026

The previous sections report speedups at batch size 1.
Do they survive a production serving engine?



Why this paper?

How SD is usually evaluated

- ▶ Research prototypes, not serving engines
- ▶ Missing production optimisations (e.g. CUDA Graphs)
- ▶ **Batch size 1**
- ▶ Only averages: latency, dataset-level acceptance

What this paper does

- ▶ Benchmarks SD inside **vLLM**
- ▶ 5 SD variants, 4 targets, 6 workloads
- ▶ Time and memory breakdown per stage
- ▶ Per-position acceptance and an oracle upper bound

Central question

How far is SD in a real system from the speedup it promises, and from the best speedup it could achieve?



Experimental setup

System

- ▶ vLLM v0.10.1.1, NVIDIA H100 80 GB
- ▶ 8B: 1 GPU; 70B/106B: 4 GPUs (TP = 4)
- ▶ Continuous batching, chunked prefill, KV-cache management, CUDA Graphs
- ▶ Greedy decoding ($T = 0$)

SD variants ($k = 3$ draft tokens per step)

- ▶ n -gram (prompt lookup, training-free)
- ▶ EAGLE, EAGLE-3 (fine-tuned heads)
- ▶ Draft model (separate small LLM)
- ▶ MTP (heads co-trained with the target)

Targets: Llama-3.1-8B, Llama-3-70B, Qwen3-8B (thinking on/off), GLM-4.5-Ai

Workloads: CNN/DailyMail, ShareGPT, InstructCoder (code editing), GSM8K, AIME22–24, GPQA-Main (reasoning)

Batch sizes: 1 to 128 (time breakdown up to 512)



A subtle catch: greedy outputs are not identical

SD is lossless in theory, yet generation lengths change with batch size and SD method, *even without SD*.

Batch size	w/o SD	n -gram	EAGLE	EAGLE-3
1	737	750	731	736
8	722	676	709	704
128	714	739	709	709

Mean generated tokens per request, ShareGPT, Llama-3.1-8B (paper Tab. 1).

- ▶ Cause: GPU numerical nondeterminism (kernel parallelism, floating-point order), not bugs.
- ▶ So end-to-end latency is not comparable across runs.

Metric: **generated tokens / second**. Speedup = throughput with SD / without SD.



The model behind the measurements

The paper reads its results through the Leviathan et al. formula from the earlier section:

$$\mathbb{E}[\text{speedup}] = \frac{1 - \alpha^{k+1}}{(1 - \alpha)(kc + 1)}$$

α : acceptance rate k : proposal length c : draft cost / target cost per forward pass

Two levers

- ▶ Better drafter $\Rightarrow$ higher α , but usually higher c
- ▶ Only the *relative* draft cost c matters, not its absolute time

Hidden assumption

Verifying $k+1$ tokens costs about as much as decoding 1. True when **memory-bound** (small batch); breaks when **compute-bound** (large batch).

The next findings are this formula meeting a real serving system.



Finding 1: speedup shrinks with batch size

Every SD variant beats the no-SD baseline, but the gain falls as batches grow.

EAGLE on	small batch	large batch	change
Llama-3.1-8B, GSM8K	1.73 $\times$ (bs 1)	1.21 $\times$ (bs 128)	-30%
Llama-3.1-8B, ShareGPT	1.68 $\times$ (bs 1)	1.61 $\times$ (bs 32)	-4.3%
Llama-3-70B, ShareGPT	1.96 $\times$ (bs 1)	1.72 $\times$ (bs 32)	-14.0%

Why

- ▶ Small batch: memory-bound, idle compute absorbs drafting and rejected tokens
- ▶ Large batch: compute-bound, every rejected token costs real time

Bigger models suffer earlier

- ▶ 70B runs on 4 GPUs and is already compute-bound at medium batch
- ▶ Less spare compute for verifying tokens that get rejected



Finding 2: there is no universal drafter

n-gram

Usually the weakest, but on **InstructCoder** (code editing) it beats EAGLE and EAGLE-3 on Llama-3.1-8B and Qwen3-8B: outputs copy the prompt.

EAGLE-3 / draft model

EAGLE-3 or the draft model gives the highest speedup on GSM8K, CNN/DailyMail and ShareGPT.

Draft model: wins on 70B, loses on 8B. Acceptance is not the problem.

GSM8K	acceptance	c	Eq. (1), $k = 3$
Llama-3-70B + Llama-3.2-1B	75%	$\approx 12.5\%$	$\approx 1.99\times$
Qwen3-8B + Qwen3-0.6B	81%	$\approx 37.5\%$	$\approx 1.41\times$

Last column: our illustration, plugging the reported acceptance rate in as α .

The best drafter depends on the workload *and* on the draft/target cost ratio.



Do draft trees help at scale?

EAGLE/EAGLE-2 style trees verify many candidates per step. Tested in SGLang, depth 3, k = tokens verified per step.

Qwen3-8B, GSM8K	chain $k = 3$	tree $k = 6$	tree $k = 21$
Speedup at batch 1	1.65×	1.68×	1.85×
Accepted length	2.25	2.51	2.92
Acceptance rate	0.415	0.300	0.095

- ▶ Wider trees accept a little more, but verify **many more rejected tokens**.
- ▶ By batch 64, the $k = 21$ tree is **below** $1\times$ on every workload (Qwen3-8B and Llama-3-70B); the chain stays above $1\times$.

Trees buy a narrow low-batch gain. Chains are the robust choice for serving.



Finding 3: SD still works for long reasoning

Reasoning means short prompts, very long outputs (up to 32K tokens).
Medium batch sizes only, to avoid KV-cache preemption.

Qwen3-8B-Thinking

AIME22–24 and GPQA-Main:

- ▶ EAGLE-3: 1.64–1.80×
- ▶ n -gram: 1.50–1.58×

Long traces repeat symbols and phrases, which n -gram reuses.

GLM-4.5-Air (MTP)

GPQA-Main: 1.3–1.8×, below its potential. Only the first MTP head is released and it is reused for later tokens:

acceptance by position: 0.92 $\rightarrow$ 0.68 $\rightarrow$ 0.38

Speedups match those on ordinary tasks. Drafters that stay accurate over long contexts win.



Finding 4: verification is the real cost

Share of execution time per SD stage (CNN/DailyMail, batch 1–512):

Stage	Share of execution time
Verification (target model)	42%–95%, grows with model size and batch size
Drafting: n -gram	$< 2\%$
Drafting: EAGLE / EAGLE-3	12–20% at batch 1 $\rightarrow$ 3–7% at batch 512
Drafting: draft model	Qwen3-8B: $\approx 47\%$ at batch 1; Llama-3-70B: 21%
Rejection sampling	$< 1.7\%$
Other vLLM overhead	3–12%

Memory is not the issue: n -gram adds nothing; EAGLE heads add $\leq 5.3\%$ weights; a 0.6B draft model raises Qwen3-8B's KV cache per token from 144 to 256 KiB ($1.77\times$).

Every rejected draft token still pays for a slice of the target pass.



Finding 5: acceptance is highly variable

Tokens generated per step (accepted +1) vary at three levels:

Within a request

GPQA,
Qwen3-8B-Thinking:
per-step tokens rise as the
trace grows.
 n -gram: 1.6–3.7 (short) $\rightarrow$
2.7–5 ($> 13K$ tokens),
dipping at the final answer.

Across requests

InstructCoder,
Llama-3-70B, per-request
mean:
EAGLE: 2.7–7.4
 n -gram: 1.1–15.0
draft model: 5.6–18.3

Across datasets

n -gram: bursty,
heavy-tailed; std. $2\text{--}5\times$
EAGLE's, spans > 15
tokens on code.
EAGLE: compact, typically
2–4.

Implication

A fixed proposal length k is almost never the right length.



Case study: why n -gram wins on code editing

Typical InstructCoder request

Prompt: class Car ... “replace the string colour with an enum”

Output: adds class Color(Enum), then **copies the rest of the prompt** with one local edit.

n -gram proposes the next tokens straight from the prompt, so long copied spans are accepted almost for free.

Overlap predicts the winner

Group requests by BLEU-4 between prompt and output:

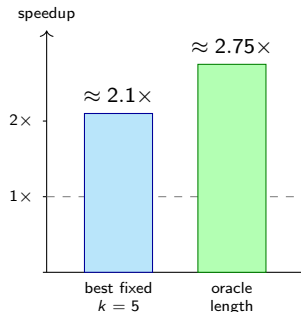
- ▶ Low overlap: n -gram loses (bad guesses)
- ▶ BLEU-4 > 0.6 (Llama-3.1-8B): n -gram beats EAGLE/EAGLE-3 at **every** batch size
- ▶ Up to 53% higher speedup with $k = 3$; up to 100% with $k = 5$

Cheap drafters win exactly where the output repeats its context.



Finding 6: large oracle headroom

Oracle: know each step's accepted length in advance and propose exactly that many tokens. No verification is wasted.



n -gram, InstructCoder, Llama-3.1-8B, batch 1

- ▶ The gap **widens with batch size**: fixed- k curves fall fast, the oracle degrades gently.
- ▶ EAGLE and n -gram win at *different positions*.
- ▶ **Oracle_Combine**: pick the better drafter at every step, propose its accepted length.
- ▶ Up to $2.2\times$ more speedup than the best fixed strategy, and up to $4.9\times$ over no SD (largest on InstructCoder, little on GSM8K).

Predicting *how far* and *with what* to speculate is worth more than a better drafter alone.



What the paper leaves open

These are upper bounds: the oracle assumes perfect prediction, and Oracle_Combine ignores the cost of keeping both drafters' state.

Today's SD in serving

- ▶ One drafter, chosen offline
- ▶ One fixed proposal length k
- ▶ Same policy at every batch size

What the headroom suggests

- ▶ Lightweight predictor of acceptance
- ▶ Adapt k per request, position and load
- ▶ Switch drafters, e.g. n -gram $\leftrightarrow$ EAGLE

less wasted verification $\Rightarrow$ speedup that survives large batches



Back to DFlash: better drafting, same verification

DFlash (previous section) attacks the **drafting** side.

What DFlash changes

- ▶ Block-diffusion drafter
- ▶ Whole block drafted in one pass
- ▶ High acceptance, low drafting cost c

What it does not change

- ▶ The target still verifies every drafted token
- ▶ Verification stays the dominant cost
- ▶ Rejected tokens still use target compute

Same trend with batch size, in vLLM

Qwen3.5-9B, Math500, concurrency $1 \rightarrow 32$:

$4.0\times \rightarrow 3.2\times \rightarrow 2.5\times \rightarrow 1.9\times$

Better drafting raises the ceiling; verification sets the floor.

