Throughput Optimization for Multi-Level Speculative Decoding

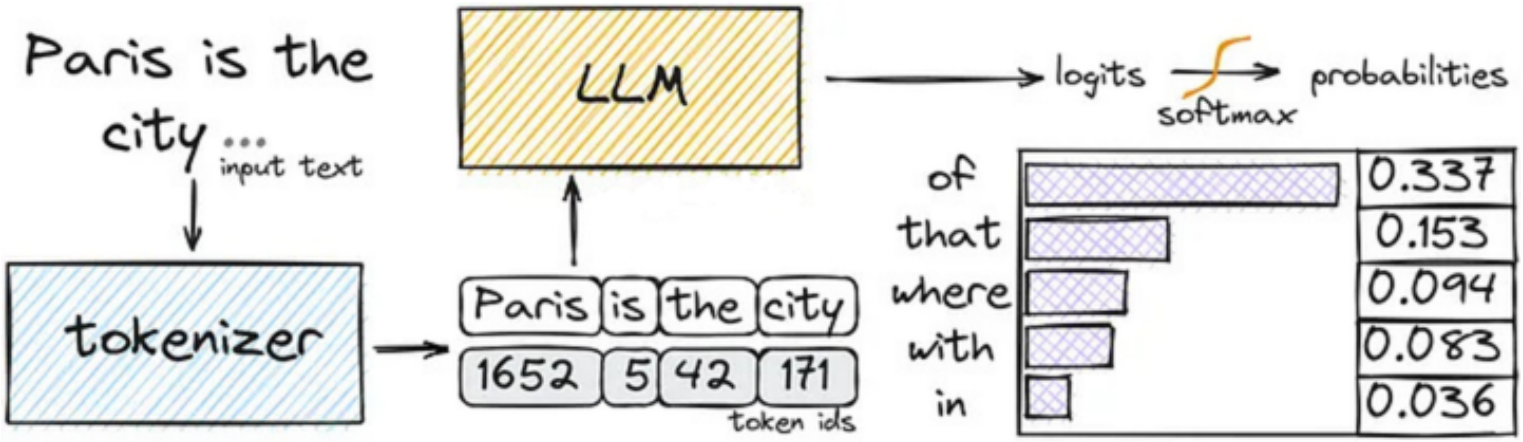
Fréjus 2026

Anne Benoit Loris Marchal Adrien Obrecht

March 16, 2026

Multi-Level Speculative Decoding (Inference)

Input: prefix, output: **one** token (word)



For this talk, we assume *deterministic* generation

Multi-Level Speculative Decoding (Inference)

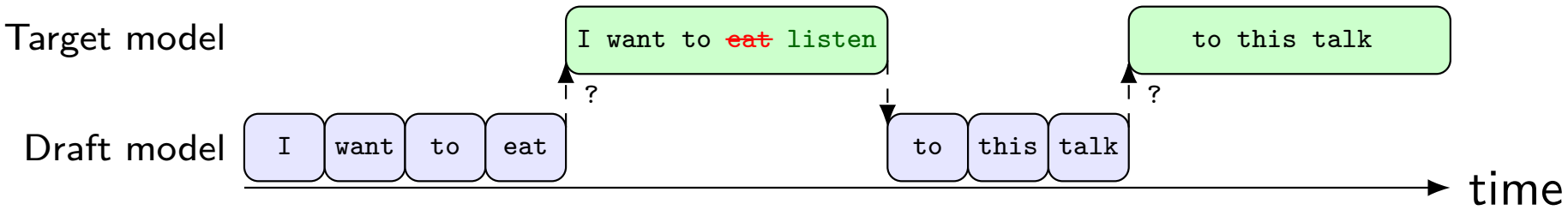
Input: prefix, output: **one** token (word)

- In 2025, data centers account for $\sim 1.5\%$ of the world's electricity demand, $+30\%$ each year.
- Speeding inference is a very heavy topic
- Big models, intrinsically *sequential* generation

Multi-Level Speculative Decoding

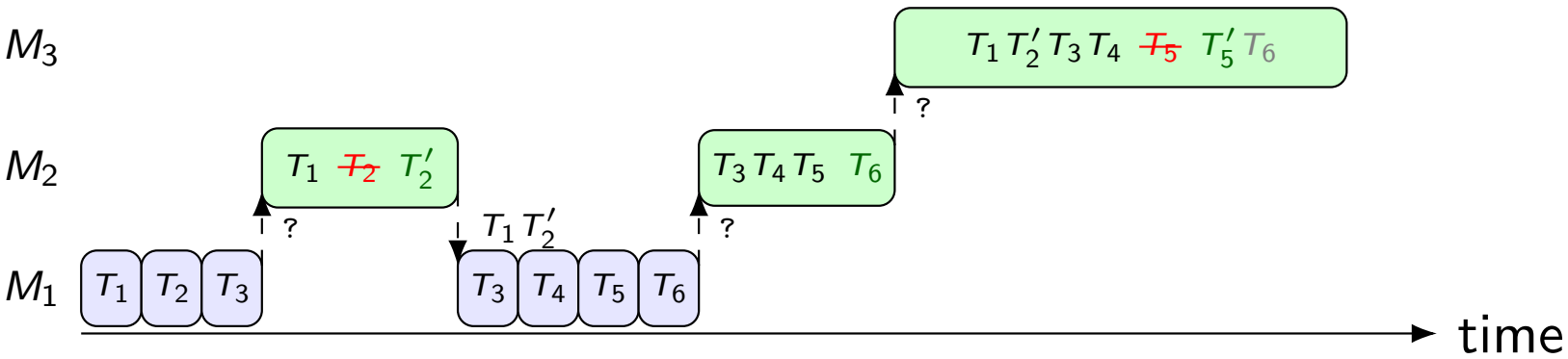
Drafting tokens: sequential, one run of the model = one token

Multi-Level Speculative Decoding



Drafting tokens: sequential, one run of the model = one token
Verification of tokens: parallel, one run of the model = many tokens

Multi-Level Speculative Decoding



Models $M_1, ..., M_t$ with respective costs C_i.
How to optimize for the throughput ?

Fixed length generation

Let k_n be the number of tokens generated by M_{n-1} for M_n.

Algorithm 1: Fixed length token generation, initially called with *model_id* = *target* and *prefix* = []

`FixedLengthGen` (*model_id*, *prefix*);

if *model_id* = 1 **then**

  return M_1(*prefix*);

→ Base case, normal drafting

Fixed length generation

Let k_n be the number of tokens generated by M_{n-1} for M_n.

Algorithm 2: Fixed length token generation, initially called with *model_id* = *target* and *prefix* = []

```
FixedLengthGen (model_id, prefix);
if model_id = 1 then
    return M_1(prefix);
genS ← []; /* genS is the generated sequence */
while length(genS) < k_{model_id} do ⟶ Accumulate k_n tokens
    genS ←
      genS ⊕ FixedLengthGen(model_id − 1, prefix ⊕ genS);
Verify(model_id, prefix, genS);
```

Analysis for 2 Models

Assumption: tokens are independent.
Each token generated by M_d is accepted by M_t with probability α.

- k: number of tokens proposed by M_d
- X: number of tokens accepted by M_t per verification step

Analysis for 2 Models

Assumption: tokens are independent.
Each token generated by M_d is accepted by M_t with probability α.

- k: number of tokens proposed by M_d
- X: number of tokens accepted by M_t per verification step

Distribution of accepted tokens

$$\mathbb{P}(X = i) = \begin{cases} \alpha^{i-1}(1-\alpha) & i \leq k \\ \alpha^k & i = k+1 \end{cases}$$

Analysis for 2 Models

Assumption: tokens are independent.
Each token generated by M_d is accepted by M_t with probability α.

- k: number of tokens proposed by M_d
- X: number of tokens accepted by M_t per verification step

Expected draft calls

Let $R(r)$ be the expected number of calls to M_d required to generate r accepted tokens.

$$R(r) = 1 + \sum_{i=1}^{\min(r,k+1)} \mathbb{P}(X = i)\, R(r - i)$$

Analysis for 2 Models

Assumption: tokens are independent.
Each token generated by M_d is accepted by M_t with probability α.

- k: number of tokens proposed by M_d
- X: number of tokens accepted by M_t per verification step

Cost and throughput

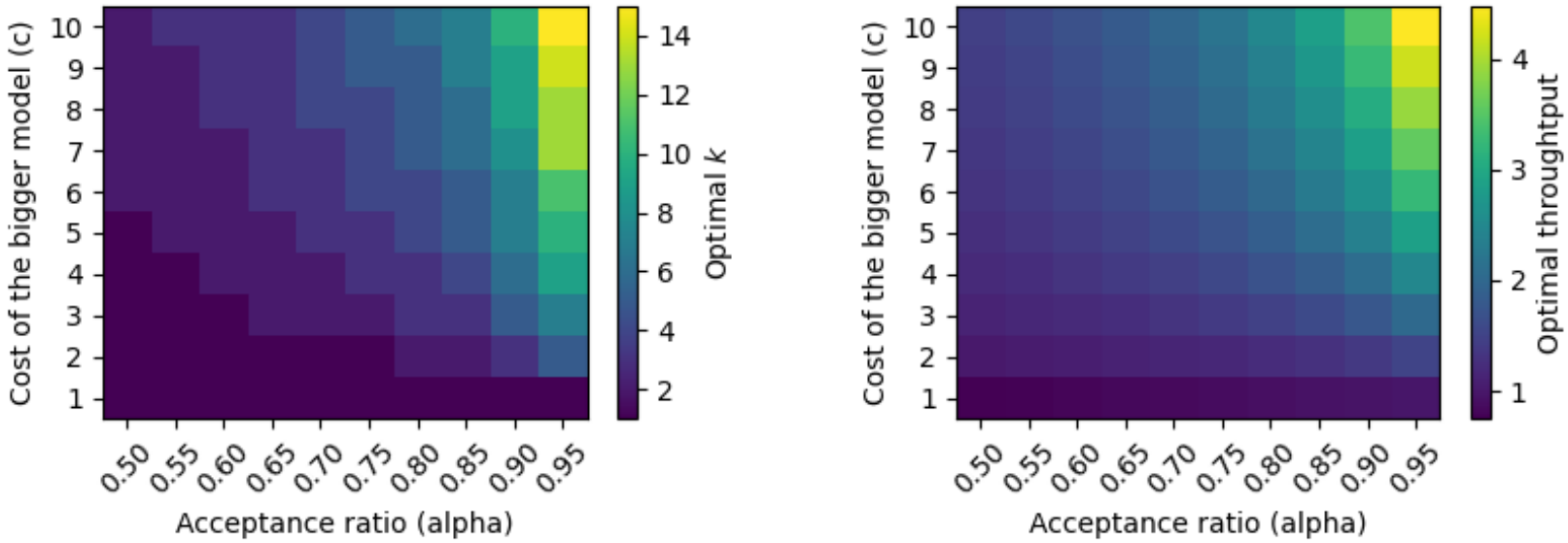
True cost:

$$TC = C_t + R(k)\, C_d$$

Throughput:

$$\frac{TC}{\mathbb{E}[X]}$$

Finding optimal parameters



(a) Optimal value for k (b) Optimal throughput

Figure: Comparison of optimal k and throughput for two models under varying acceptance rates and relative costs.

High acceptance rate + big model size differential $\rightarrow$ better throughput

Let's try in practice !

Experiments using the Grid5000 infrastructure

Draft/Target Model	Llama 3.1 8B	Llama 3.2 3B	Llama 3.2 1B
Llama 3.2 3B	85.3%		
Llama 3.2 1B	74.9%	87.9%	
SmolLM 360M	44%	51%	48%

Acceptance rate of different model pairs

Let's try in practice !

Experiments using the Grid5000 infrastructure

Draft/Target Model	Llama 3.1 8B	Llama 3.2 3B	Llama 3.2 1B
Llama 3.2 3B	85.3%		
Llama 3.2 1B	74.9%	87.9%	
SmolLM 360M	44%	51%	48%

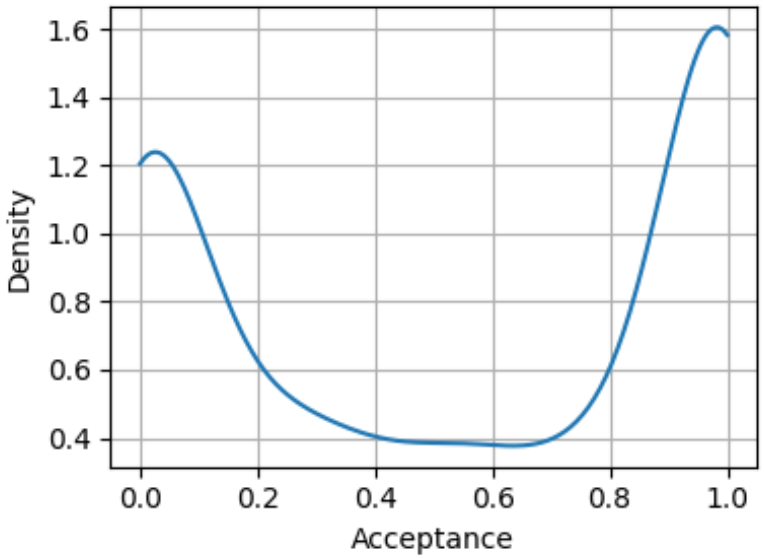
Acceptance rate of different model pairs

Model hierarchy	Throughput Impr.	Optimal k_i's
SmolLM → Llama 1B → Llama 3B → Llama 8B	×0.90	1-4-3
SmolLM → Llama 3.2 1B → Llama 3.1 8B	×1.68	1-5
Llama 3.2 3B → Llama 3.1 8B	×1.21	2
Llama 3.2 1B → Llama 3.1 8B	×1.48	3
SmolLM → Llama 3.1 8B	×1.30	2

Throughput improvements

Acceptance rate is **not** constant

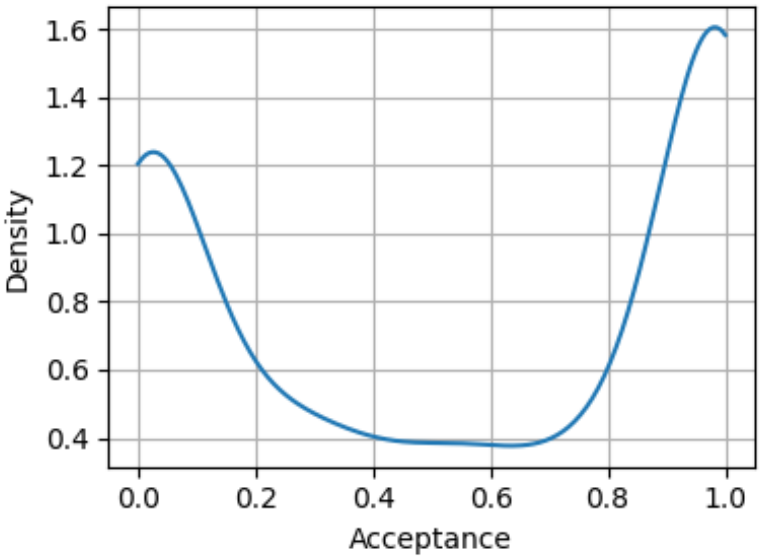
Draft: Llama 3.2 1B, Target: Llama 3.2 3B



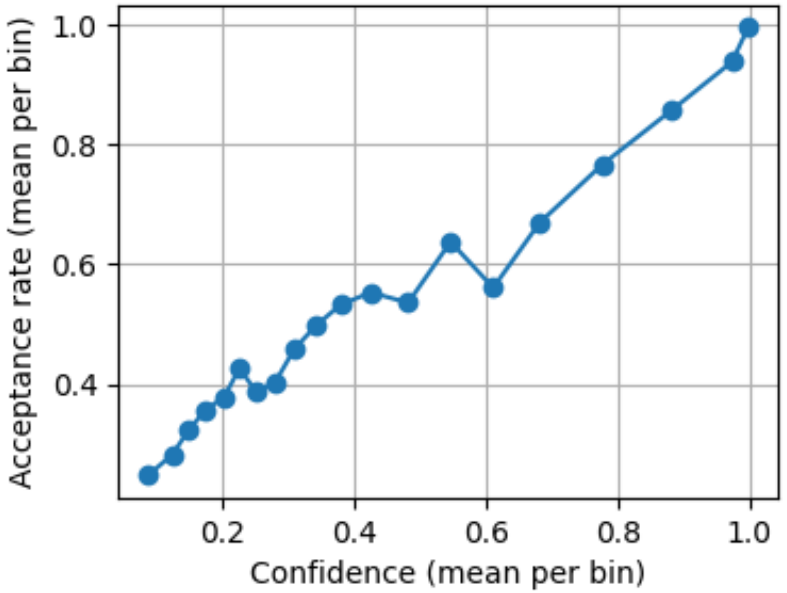
(a) Acceptance rate distribution.

Acceptance rate is **not** constant

Draft: Llama 3.2 1B, Target: Llama 3.2 3B



(a) Acceptance rate distribution.



(b) Acceptance rate vs confidence of token (from logits).

Use confidence to estimate acceptance rate !

Threshold based algorithm

Let p_n be the confidence threshold for tokens generated by M_{n-1} for M_n.

Algorithm 2: Dynamic token generation with threshold on token confidence

`ThresholdGen` (*model_id*, *prefix*);

if $model\_id = 1$ **then**
  return $M_1(prefix)$

Base case, normal drafting

Threshold based algorithm

Let p_n be the confidence threshold for tokens generated by M_{n-1} for M_n.

Algorithm 3: Dynamic token generation with threshold on token confidence

`ThresholdGen` (*model_id*, *prefix*);

if $model\_id = 1$ **then**
  return $M_1(prefix)$

$gen\_seq \leftarrow []$;

$cumul\_p \leftarrow 1$;

while $cumul\_p \geqslant p_{model\_id}$ **do**

Accumulate tokens until confidence below p_n

  $new\_seq \leftarrow$ `ThresholdGen`$(model\_id - 1, prefix \oplus gen\_seq)$;
  $gen\_seq \leftarrow gen\_seq \oplus new\_seq$;
  $cumul\_p \leftarrow cumul\_p \times new\_seq.confidence$;

return `Verify`$(model\_id, prefix, gen\_seq)$

Scary math slide

Instead of each token generated by M_d having a fixed probability of acceptance by M_t of α, it is now drawn from a distribution of pdf f.
$V(p)$: average accepted tokens per verification steps (with threshold p).

$$V(p) = \int_p^1 f(s)s\left(1 + V\left(\frac{p}{s}\right)\right) ds.$$

- Discretize $[0, 1]$
- Solve backwards
- Converges when discretization goes to 0 (*Volterra equations*)

In the end, given a threshold we can compute the throughput in polynomial time.

Experiments

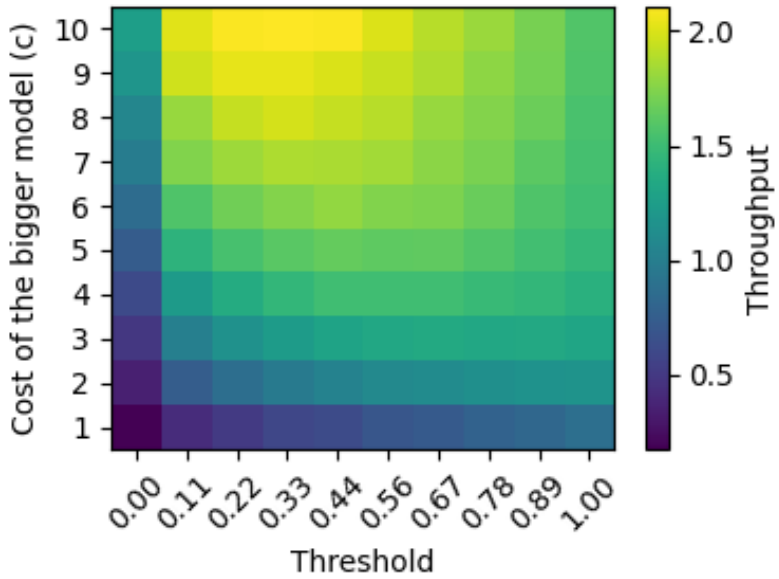
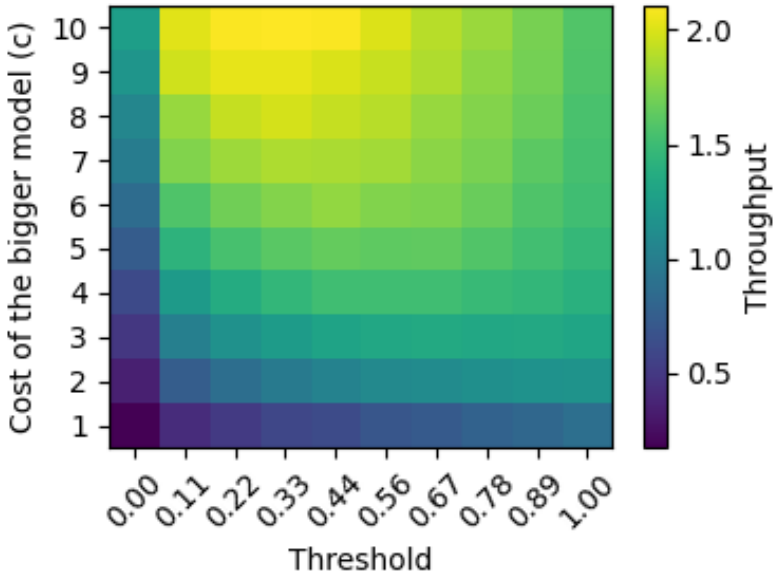


Figure: Optimal throughput for 2 models, under uniform confidence distribution.

Big threshold: no draft tokens, no speedup
Small threshold: too much drafting, slowdown

→ Our algo finds middle ground

Experiments



Better than fixed generation!

Models	Throughput	Optimal threshold
SmolLM → Llama 3.2 1B → Llama 3.1 8B	×2.15	0.94 – 0.43
Llama 3.2 3B → Llama 3.1 8B	×1.51	0.55
Llama 3.2 1B → Llama 3.1 8B	×2.07	0.4
SmolLM → Llama 3.1 8B	×1.81	0.35

Conclusion

- Considered a multi-level speculative decoding framework
- Found optimal parameters for fixed length and adaptive generation
- Experiments found that adaptive generation works better than fixed length, with up to 2× speedups.

This work was submitted to Europar2026, code is available online at `https://gitlab.inria.fr/aobrecht/europar2026`
Future works:

- Extending results to tree-based generation
- Real speedup tests using vLLM