中国电信云计算研究院
China Telecom Cloud Computing Research Institute

Ygg: Tree-Based Colaborative Speculative Decoding with Token-Only Transmission

Sijia Li[12], Yumeng Liang[2], Ruiqi Li[3], Jianjiang Li[1], and Jie Wu[24]
[1]University of Science and Technology Beijing
[2]China Telecom Cloud Computing Research Institute
[3]Peking University
[4]Temple University

Name after Yggdrasills, the world tree in Norse mythology that connects realms of different worlds.

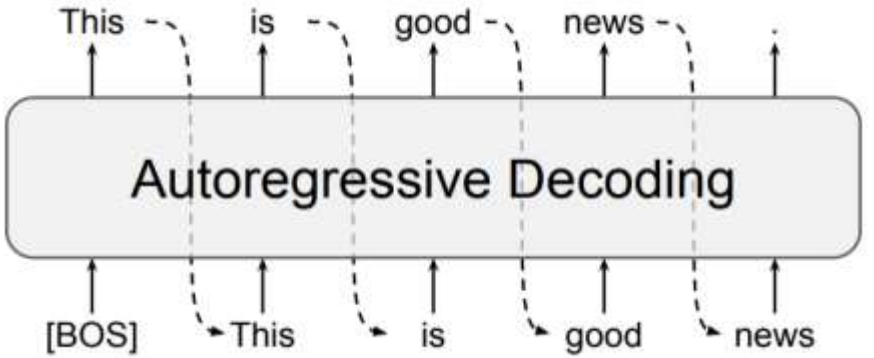
Background

LLM Inference

One token per step:

LLMs generate text via autoregressive decoding.

Each token depends on all previously generated tokens, ensuring coherent sequence generation.



Source: Ge et al., "Lossless acceleration for seq2seq generation", 2022

Bottlenecks

- Sequential generation
- Limited parallelism
- High latency
- Low throughput

Improving LLM inference QoS

Why

- High demand for LLM inference
- Strict latency requirements

How

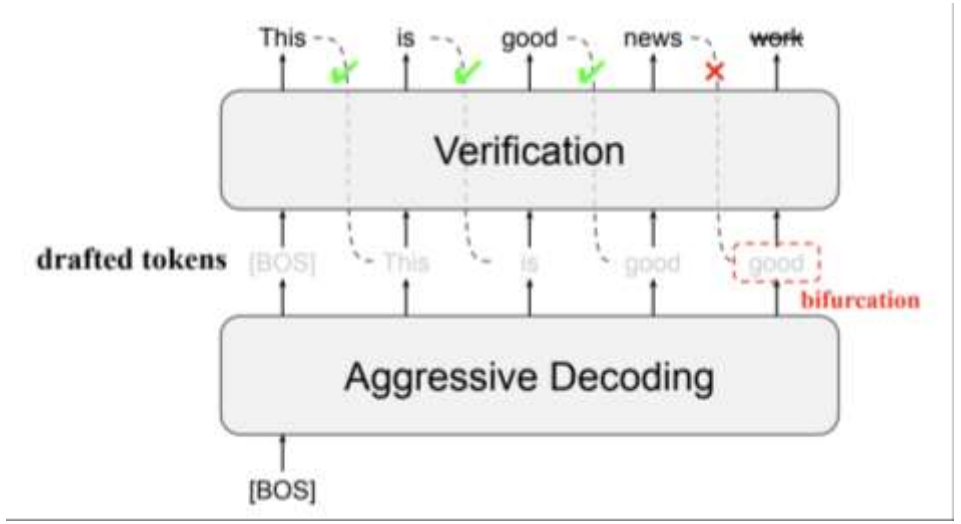
- Reducing inference latency
- Increasing system-level throughput

Background

Speculative Decoding

Accelerating LLM inference with two models

- Draft model (Aggressive Decoding): generates candidate tokens sequentially
- Target model (Verification): evaluates multiple drafted tokens in parallel



Source: Ge et al., "Lossless acceleration for seq2seq generation", 2022

An Example

[START] japan ' s benchmark ~~bond~~ n

[START] japan ' s benchmark nikkei 22 ~~7~~5

[START] japan ' s benchmark nikkei 225 index rose 22 ~~7~~6

[START] japan ' s benchmark nikkei 225 index rose 226 . 69 ~~7~~ points

[START] japan ' s benchmark nikkei 225 index rose 226 . 69 points , or 0 1

Source: Leviathan et al., "Fast inference from transformers via speculative decoding", 2023

Benefits

- Reducing the number of decoding steps
- Better utilizing cloud GPU parallelism
- Lower latency and higher throughput

Motivation

Limitation of Cloud-Only Speculative Decoding

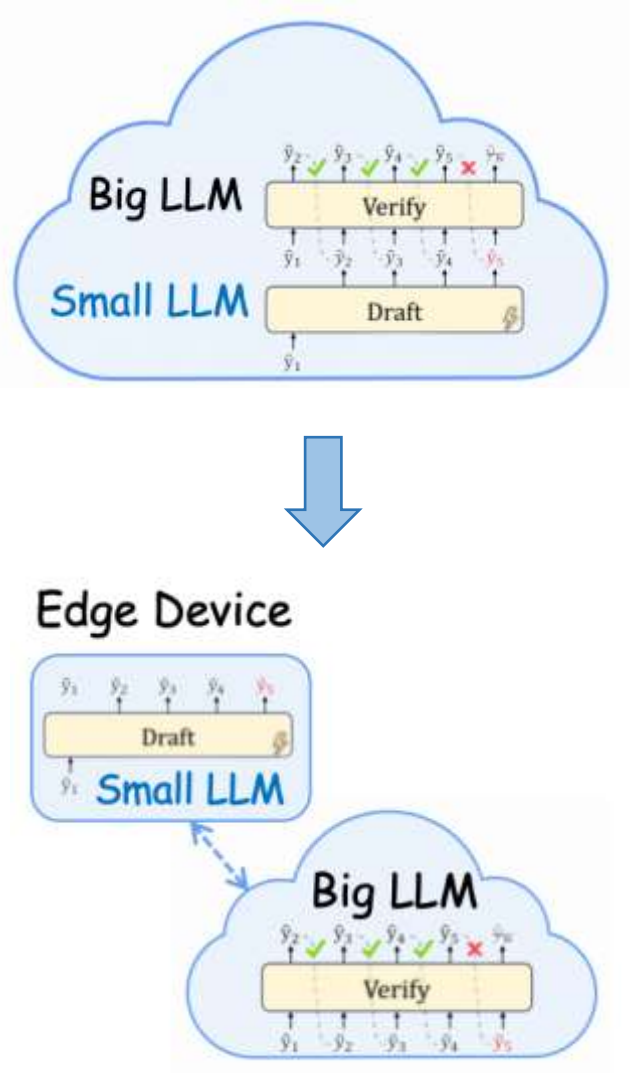
Deploy both the draft model and the target LLM in the cloud.

- The draft model consumes valuable cloud GPU memory.
- Limited cloud memory restricts batch size and serving throughput.

Edge-cloud Collaboration

The draft model is lightweight and can run on low-cost edge devices.

Offload the draft model to the edge, freeing cloud GPU memory for processing more samples in parallel and improving system throughput.



Motivation

Communication Bottleneck in Edge-Cloud SD

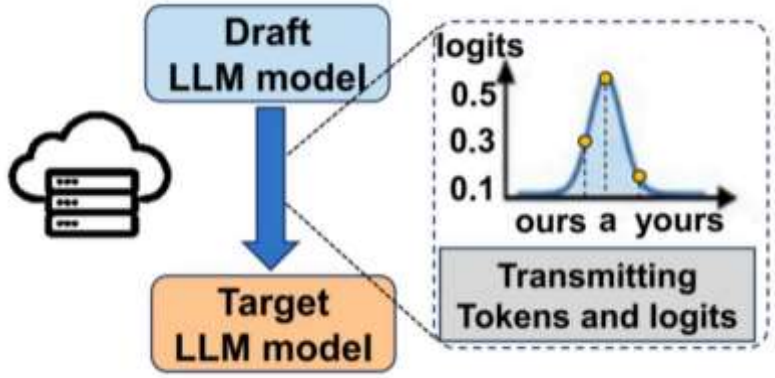
Standard speculative decoding verifies draft tokens using both:

- Candidate tokens
- Their probability distributions (logits)

Draft logits are needed for accept/reject decisions and rejection correction, so that speculative decoding preserves the same output distribution as the target model.

Logits Transmission Cause Heavy Communication

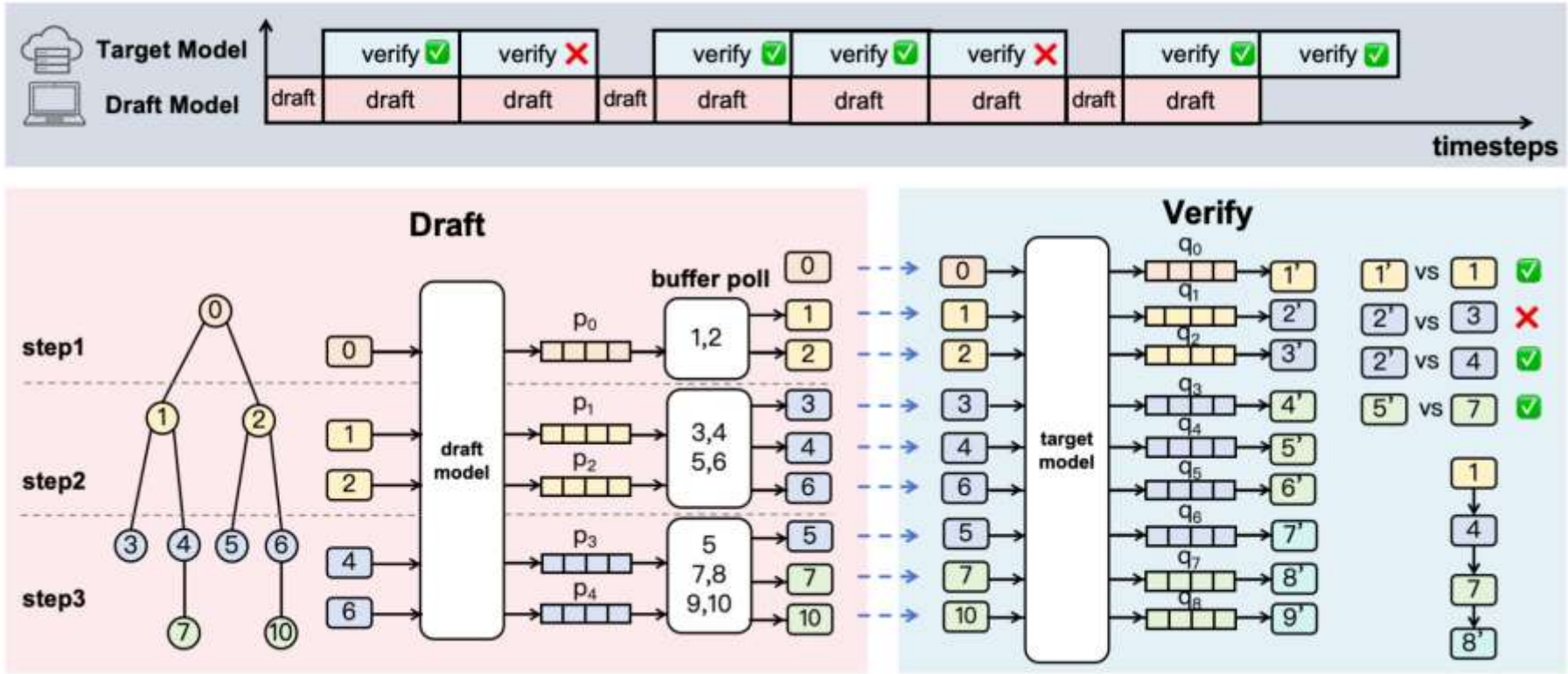
- Large vocabulary size 32K–128K
- About 1 MB payload per step
- Edge-cloud transfer introduces high inference latency



Framwork

Key Designs of Ygg

1. **Token-only Transmission:** Reduces communication latency
2. **Adaptive Tree Drafting:** Improves acceptance rate with fewer transmitted tokens.
3. **Asynchronous Pipelining:** Reduces idle bubbles with overlapped drafting and verification.



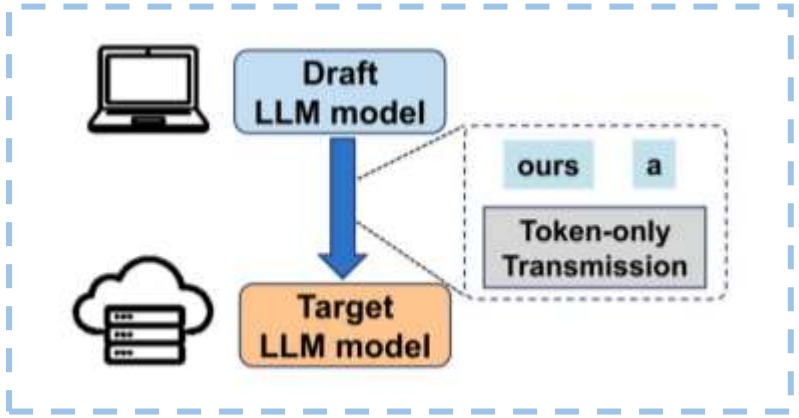
Token-only transmission

Token-only Transmission

Ygg avoids transmitting high-dimensional draft logits.

The edge only sends:

token IDs + parent indices + position IDs



Verification without Draft Logits

The cloud verify using target-model sampling.

- A token is accepted only if it matches the target-sampled token
- Otherwise, a token generatied by the target model is used as correction.

The final output still follows the target LLM distribution, while edge-cloud communication is significantly reduced.

Tree-based Drafting

Why Tree-based Drafting?

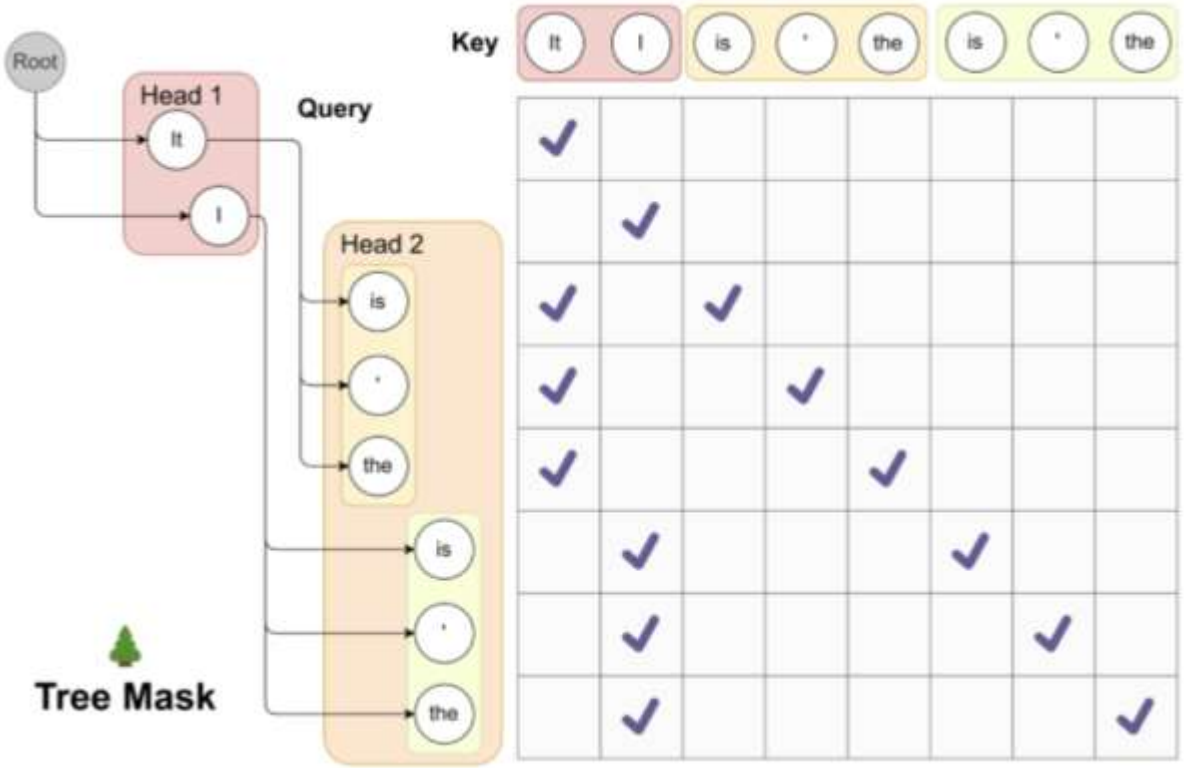
- A linear draft sequence is fragile:

  If one token is rejected, all subsequent tokens are wasted.

- Building a token tree can cover multiple likely generation paths:

  Higher acceptance rate with fewer edge-cloud interaction rounds.

- With tree attention, the target LLM can **verify the whole tree in one forward pass**.



Source: Miao et al., "SpecInfer: Accelerating Large Language Model Serving with Tree-Based Speculative Inference and Verification", 2024.

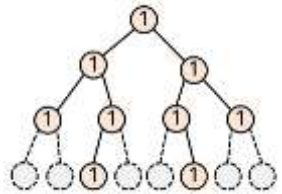
Budget-aware Adaptive Tree Construction

Better Trade-off Between Acceptance Rate and Redundant Computation

More Redundancy, Higher Acceptance Rate.

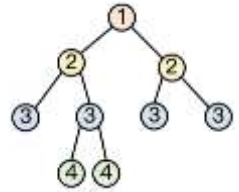
Opttree .....

Maximize redundancy to improve acceptance, then prune to the Top-K nodes.



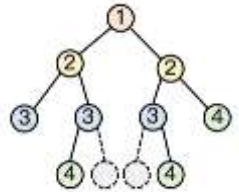
Specexce .....

Minimize redundancy, greedily expanding only the highest-confidence leaf.



Ygg .....

Maintain controlled redundancy with a global buffer, selecting high-confidence nodes across depths.



Algorithm 1: Budget-Aware Global Adaptive Tree Construction

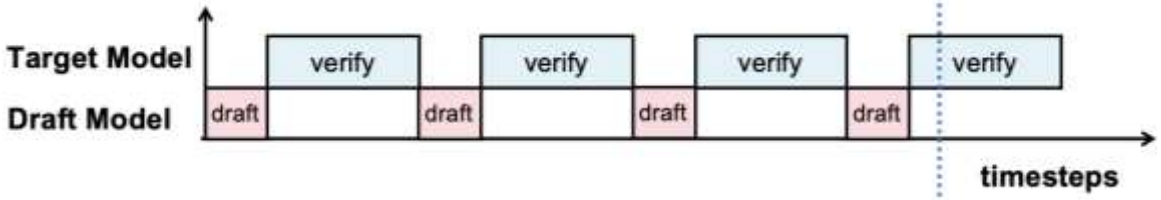
Input: Prefix $x_{1:t}$, Budget K, Max Depth D, Batch Size B, Branch Width W, Admit Rate N_{limit}
Output: Token Tree $\mathcal{T}$

Initialize Tree $\mathcal{T} \leftarrow \{x_{1:t}\}$, $Q \leftarrow \{(0, \text{root})\}$, Buffer $\mathcal{B} \leftarrow \emptyset$;
while *depth* $< D$ ***and*** $|\mathcal{T}| < K$ **do**
  // 1. Batch Expansion;
  Pop batch of parents $\mathcal{P}_{batch}$ from Q (size $\leq B$);
  Compute logits for $\mathcal{P}_{batch}$ via Draft Model;
  for *parent* $u \in \mathcal{P}_{batch}$ **do**
    Get top-W children $\{v_1, \ldots, v_W\}$ via LogSoftmax;
    Calculate scores: $S(v_i) \leftarrow S(u) + \log P(v_i|u)$;
    Add children to Buffer: $\mathcal{B} \leftarrow \mathcal{B} \cup \{v_1, \ldots, v_W\}$;
  // 2. Global Selection (Lateral Comparison);
  Sort $\mathcal{B}$ descending by score $S(\cdot)$;
  $N_{admit} \leftarrow \min(|\mathcal{B}|, N_{limit}, K - |\mathcal{T}|)$;
  $\mathcal{C}_{best} \leftarrow$ Top N_{admit} nodes from $\mathcal{B}$;
  Remove $\mathcal{C}_{best}$ from $\mathcal{B}$;
  // 3. Tree Update;
  for *node* $v \in \mathcal{C}_{best}$ **do**
    Add v to $\mathcal{T}$;
    Push v to Q for future expansion;
  if $N_{admit} = 0$ **then**
    break;
return $\mathcal{T}$

Asynchronous Pipelining

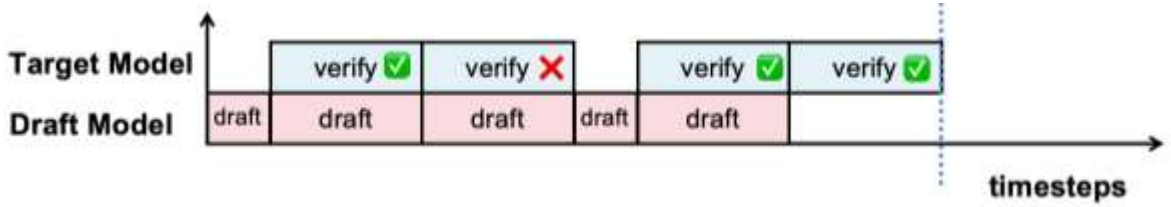
Standard Speculative Decoding

- Draft and verify execute sequentially.
- The draft model stays idle during verification.
- Verification latency creates pipeline bubbles.



Ygg

- Pre-draft future trees during cloud verification.
- Directly reuse matched pre-drafts after verification.
- Adapt pre-draft depth to verification latency.



Algorithm 2: Batched Pre-Drafting with Asynchronous Verification

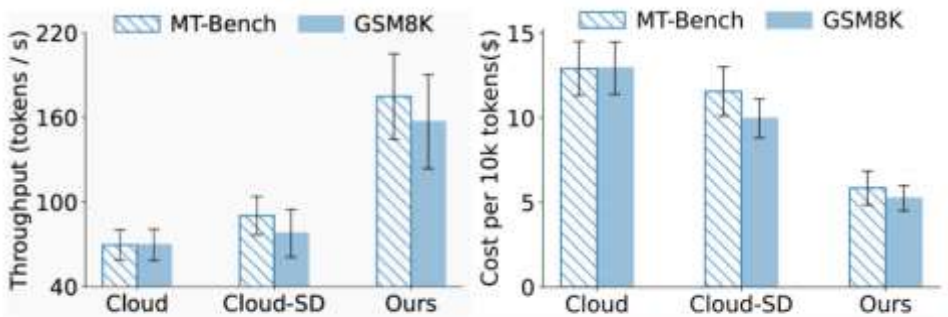
Input: Current tree $\mathcal{T}_{curr}$, draft model $\mathcal{M}_d$, cloud model $\mathcal{M}_c$, top-k k

send $\mathcal{T}_{curr}$ to Cloud;
async (Cloud): $(\mathcal{R}, l_{acc}) \leftarrow \text{Verify}(\mathcal{T}_{curr}, \mathcal{M}_c)$;
async (Edge): $\mathcal{T}' \leftarrow \text{Expand}(\mathcal{T}_{curr}, \mathcal{M}_d, 1)$;
    $\mathcal{L} \leftarrow \text{TopKLeaves}(\mathcal{T}', k)$;
    $\mathbb{S} \leftarrow \{\text{InitTree}(l) : l \in \mathcal{L}\}$;
    repeat: $\mathcal{B} \leftarrow \text{Frontier}(\mathbb{S})$;
        $\mathbb{S} \leftarrow \text{BatchExpand}(\mathcal{B}, \mathcal{M}_d)$;
    until $\mathcal{R}$ received;
$l_{acc} \leftarrow \text{AcceptedLeaf}(\mathcal{R})$;
$\mathcal{T}_{next} \leftarrow \begin{cases} \text{Retrieve}(\mathbb{S}, l_{acc}), & l_{acc} \in \mathcal{L} \\ \text{StandardDraft}(l_{acc}, \mathcal{M}_d), & \text{otherwise} \end{cases}$;
send $\mathcal{T}_{next}$ to Cloud if $l_{acc} \in \mathcal{L}$;

Results

Vs. Cloud-only methods

- Same cloud memory as autoregressive decoding.
- 1.94× higher throughput than Cloud-SD.
- Up to 47.3% lower serving cost.

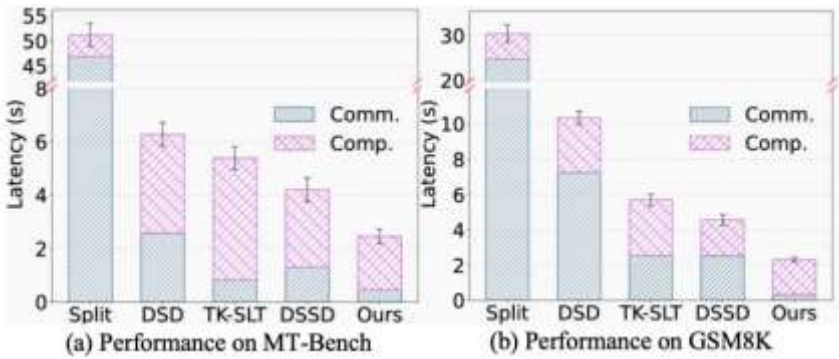


(a) Throughput Comparison (b) Cost Comparison

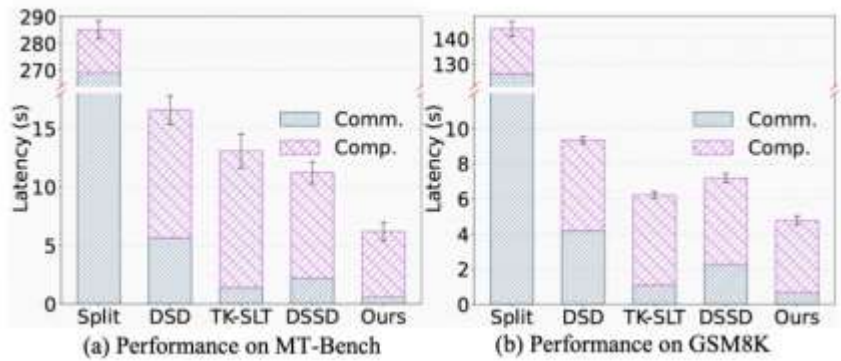
Method	Cloud Model Memory	Latency	Throughput
Cloud	121 GB	16.4 s	69.5 tokens/s
Cloud-SD	146 GB	**5.36 s**	90.3 tokens/s
Ours	**121 GB**	6.20 s	**175 tokens/s**

Vs. Other cloud-edge collaborative methods

- Lowest communication latency
- Best end-to-end latency across all baselines.



33B-7B model



7B-68M model

Results

Ablation

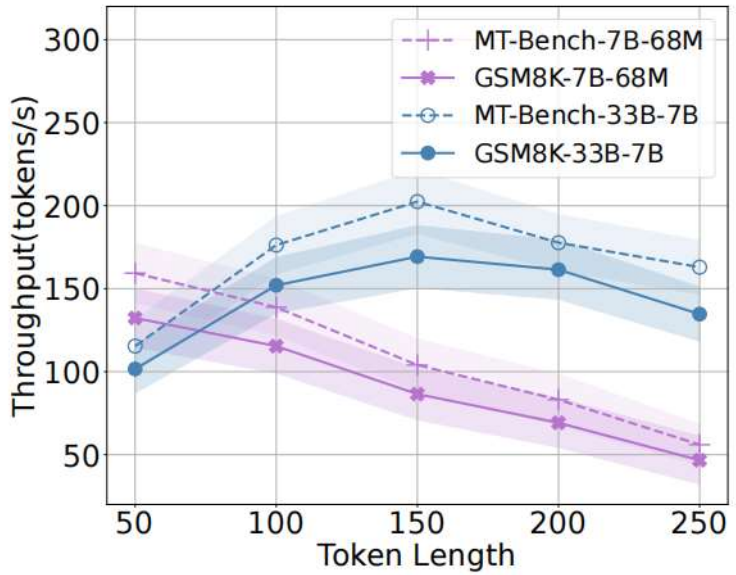
Method	Throughput (token/s)	Cost_{100k} ($)
w/o Tree	143.70	6.51
w/o Parallel	148.56	6.30
Full	**174.94**	**5.35**

- Removing either the tree-based drafting strategy or asynchronous pipelining significantly reduces throughput.
- The complete Ygg framework achieves the highest throughput and the lowest serving cost.

Performance with Different Sequence Length

- For the 33B-7B model pair, throughput remains stable above 150 tokens/s across different sequence lengths.
- For the 7B-68M model pair, throughput gradually decreases as sequence length increases.

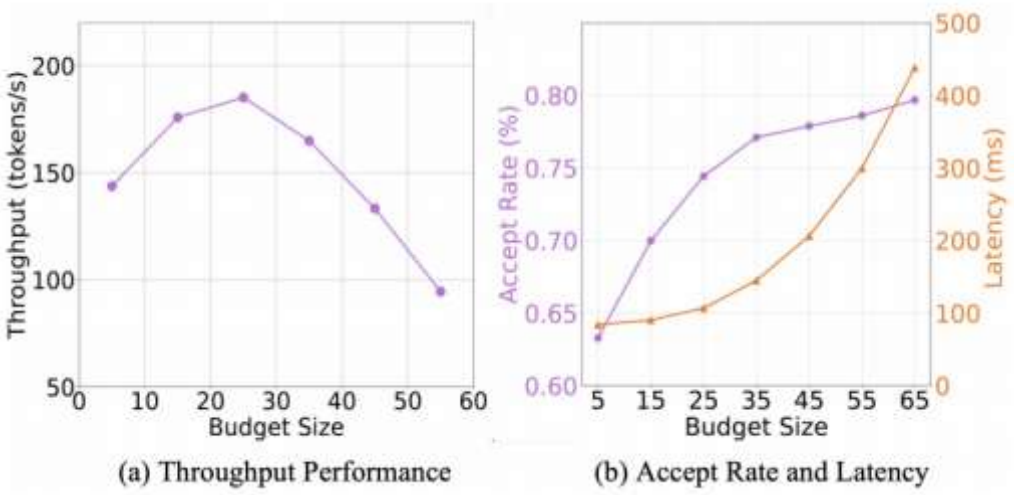
Ygg provides more stable acceleration for larger LLMs.



Results

Impact of Tree Budget

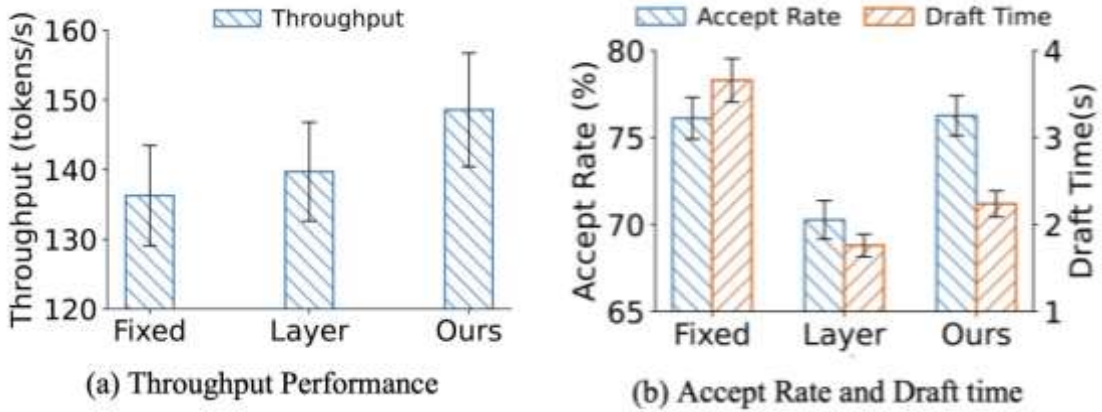
- More candidates improve acceptance rates.
- Excessive budgets introduce communication overhead.
- **A balanced budget maximizes throughput.**



(a) Throughput Performance (b) Accept Rate and Latency

Impact of tree construction algorithm

- **Controlled redundancy achieves the best throughput**



(a) Throughput Performance (b) Accept Rate and Draft time

中国电信云计算研究院
China Telecom Cloud Computing Research Institute

Thanks for Watching !

For any questions, please contact:

lisijia@xs.ustb.edu.cn