FAST-DLLM V2: Efficient Block-Diffusion LLM

Chengyue Wu[1,2] Hao Zhang[2] Shuchen Xue[2] Shizhe Diao[2]
Yonggan Fu[2] Zhijian Liu[2] Pavlo Molchanov[2] Ping Luo[1]
Song Han[2,3] Enze Xie[2]

[1] The University of Hong Kong [2] NVIDIA [3] MIT

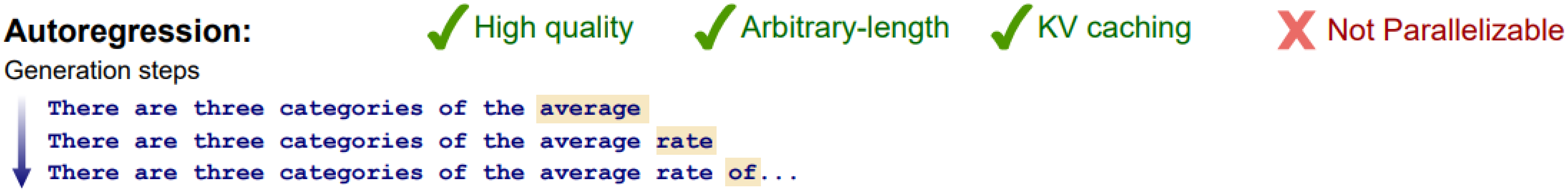
arXiv:2509.26328

Presented by Xiliang Xian

Outline

- ☐ **Background**
- ☐ Design
- ☐ Evaluations
- ☐ Discussion

Auto-Regressive(AR) LLM



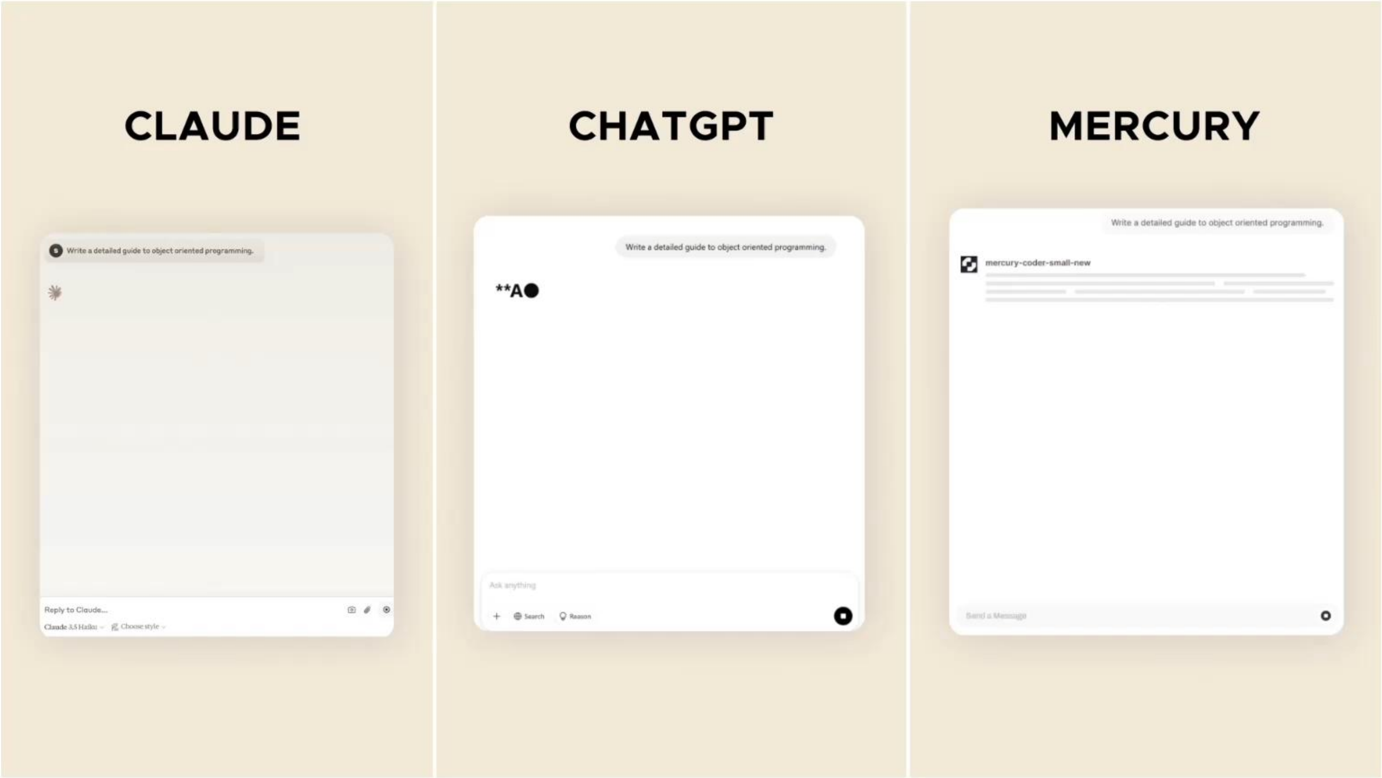
- ❑ produces one token at a time in strict left-to-right order

What is DLLM?

- Diffusion large language model

Why DLLM?

- DLLMs show **superior speed** vs Auto-Regressive(AR) LLM with **comparable performance**



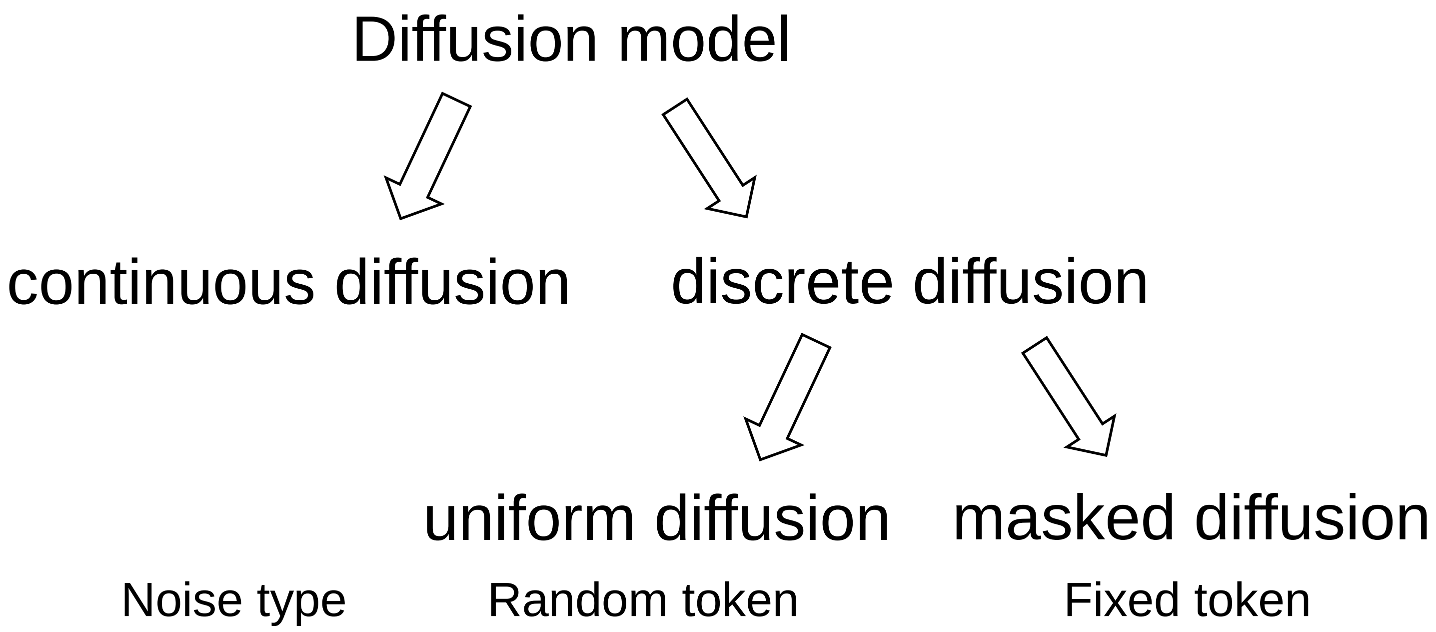
A Glimpse into Recent Development

- ❑ **Closed-source model**
- ❖ Gemini Diffusion
- ❖ Mercury (claimed to achieve 1109 tokens/s on H100s)
- ❖ Seed Diffusion (claimed to achieve 2146 tokens/s on H20s）
- ❑ **Open-source model**
- ❖ LLaDA (train from scratch)
- ❖ Dream (fine-tune from Qwen-2.5 7B)

Introduction

- ❑ Model Apply **Block Diffusion**
- ❑ **Data Efficient** Fine-tune Training Model
- ❑ Introduce Approximate **KV Cache** for **Full Attention**
- ❑ Introduce **Tokens** Parallel Decoding

Classification



Masked Diffusion Model (MDM)

Mainstream Formulation in DLLM

❑ Training details：

most Encoder-only (full attention)

❖ Forward：

LLM	is	large	language	model
[MASK]	is	large	[MASK]	model

Mask Ratio t ~ U(0,1)

❖ Reverse denoising：

[MASK]	is	large	[MASK]	model
LLM	is	large	language	model

Masked Diffusion Model (MDM)

❑ Inference details：

Prompt	[MASK]	[MASK]	[MASK]	[MASK]	[MASK]
Prompt	[MASK]	is	[MASK]	language	[MASK]
		……			
Prompt	LLM	is	large	language	model

Multi-steps：predict and remask

Predict all masked tokens simutaneously

Random remask, low confidence remask, etc

- ❖ Way for variable-length sequence:
  - ➢ Discard tokens after first <EOS>(end of sentence)

- ❑ Apply **Block Diffusion**
- ❑ **Data Efficient** Fine-tune Training
- ❑ Introduce Approximate **KV Cache** for **Full Attention**
- ❑ Introduce **Tokens** Parallel Decoding

Block Diffusion (ICLR 25)

- ❑ Apply **Block Diffusion**
- ❑ **Data Efficient** Fine-tune Training
- ❑ Introduce Approximate **KV Cache** for **Full Attention**
- ❑ Introduce **Tokens** Parallel Decoding

❑ Semi-Autoregressive

Autoregression: ✓ High quality ✓ Arbitrary-length ✓ KV caching ✗ Not Parallelizable

Generation steps

```
There are three categories of the average
There are three categories of the average rate
There are three categories of the average rate of...
```

Diffusion: ✗ Lower quality ✗ Fixed-length ✗ No KV caching ✓ Parallelizable

```
      the reusability                    will continue to       the
Repeal the reusability cuts and      the law will continue to reduce the
Repeal the reusability cuts and prove the law will continue to reduce the deficit.
```

Block Diffusion (Ours): ✓ High quality ✓ Arbitrary-length ✓ KV caching ✓ Parallelizable

```
On September 17,       we      be
On September 17, 2016, we will be giving the      release of
On September 17, 2016, we will be giving the beta-release of the      to our server     testing ...
```

Limitations in MDMs

- ❑ Apply **Block Diffusion**
- ❑ **Data Efficient** Fine-tune Training
- ❑ Introduce Approximate **KV Cache** for **Full Attention**
- ❑ Introduce **Tokens** Parallel Decoding

❑ Inductive bias conflict in training

- ❖ natural language is overwhelmingly processed in a sequential order

❑ Inference inefficiency

- ❖ take LLaDA as an example:
- ➢ Full attention is slower than causal attention
- ➢ **Cannot** leverage **KV cache**
- ➢ Reduce the number of inference steps ⇨ Decode more tokens simutaneously ⇨ **Bad accuracy**

Limitations in MDMs

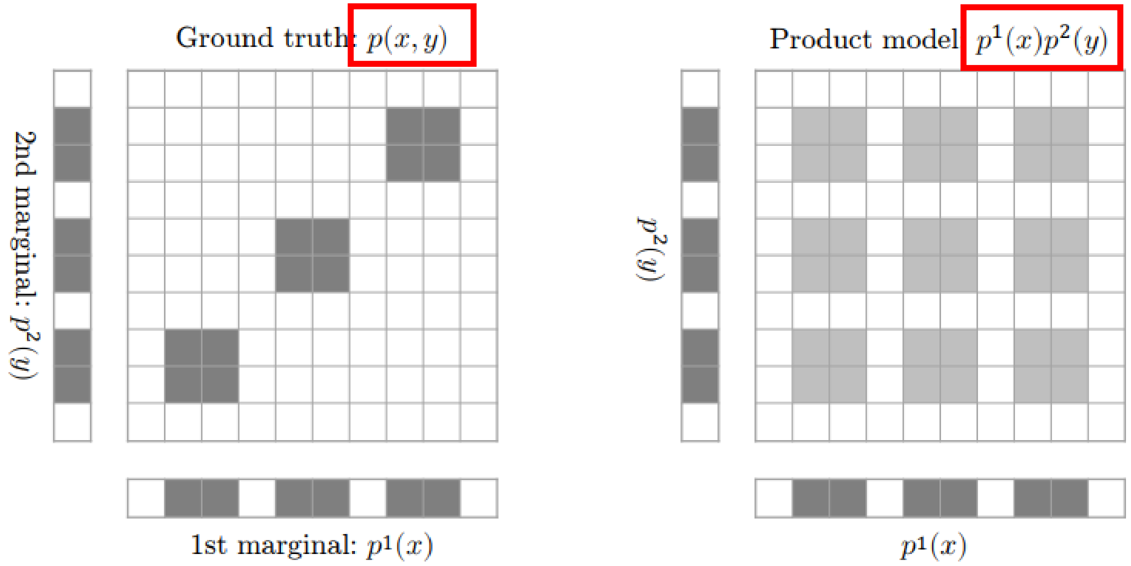
- ❑ Apply **Block Diffusion**
- ❑ **Data Efficient** Fine-tune Training
- ❑ Introduce Approximate **KV Cache** for **Full Attention**
- ❑ Introduce **Tokens** Parallel Decoding

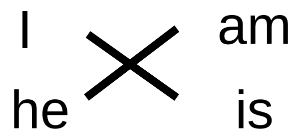
❑ Inference inefficiency

……

➢ Reduce the number of inference steps ⇨ Decode more tokens simutaneously

⇨ **Bad accuracy**



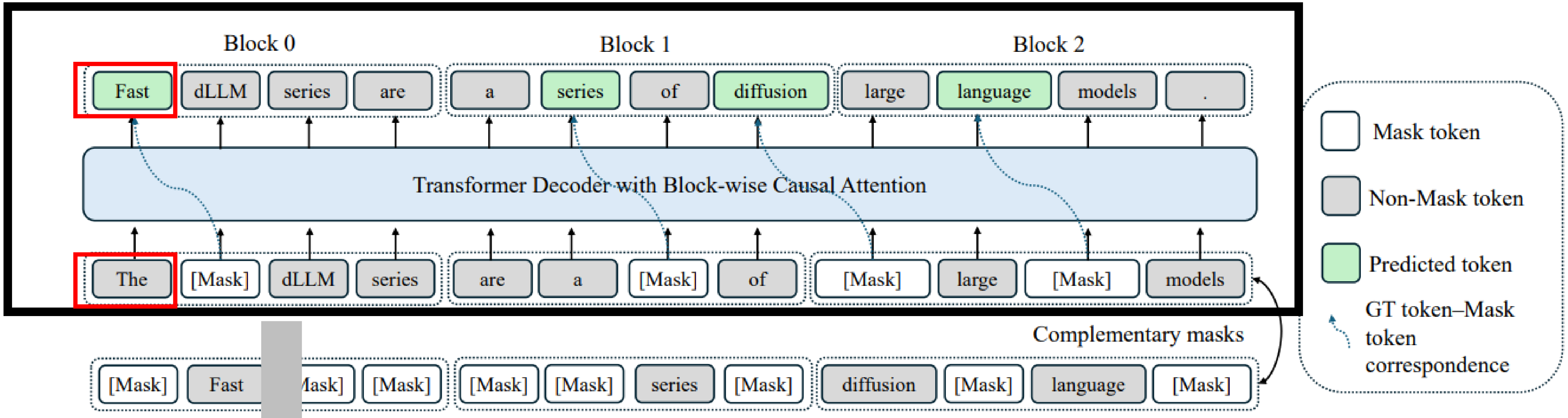


marginal distribution ≠ **joint** distribution

Outline

- Background
- Design
- Evaluations
- Discussion

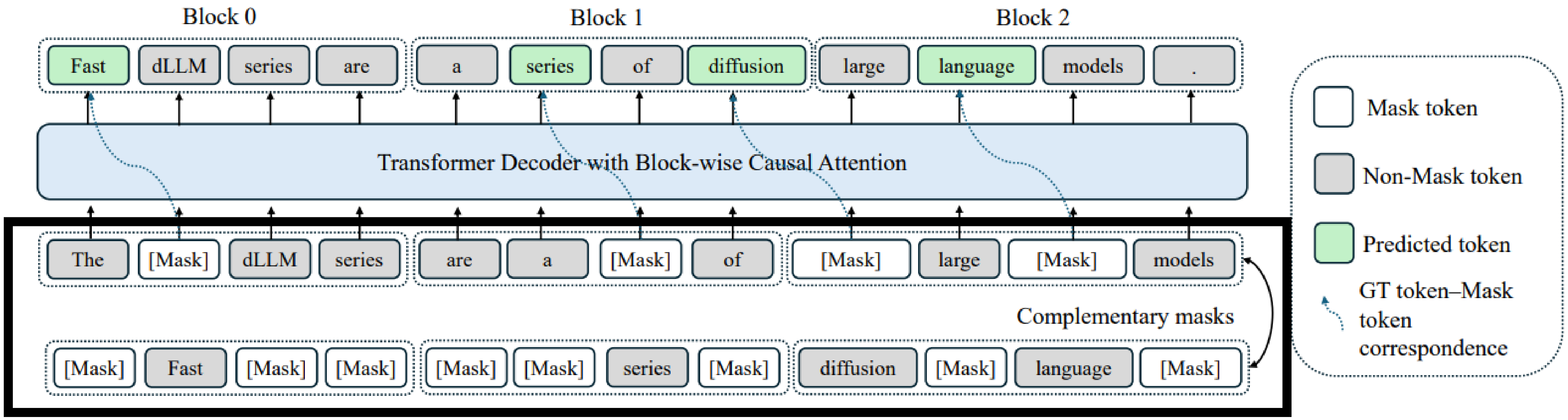
Training Process



❑ Token shift for prediction

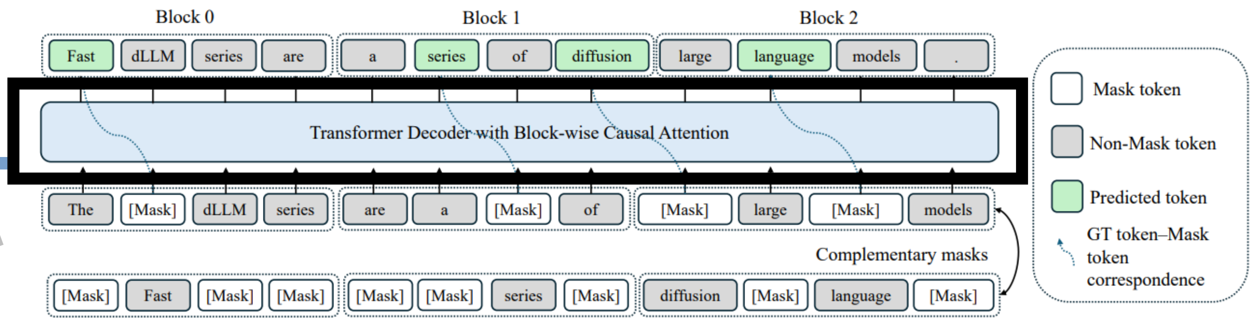
❖ Use hidden state at i-1 to predict x_i

Training Process

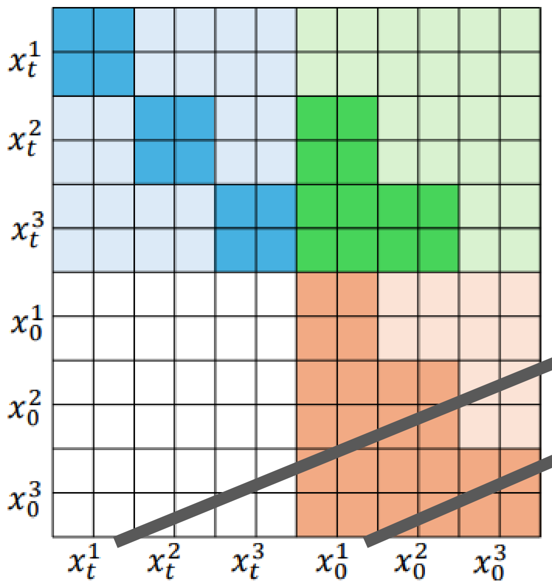


- ❑ Masked token prediction with **complementary** views

Training Process



❑ Block-wise attention



Block Diagonal → bidirectional self-attention

Offset Block Causal → inter-block causal

Block Causal → Traditional AR

Noised sequence

Clean sequence

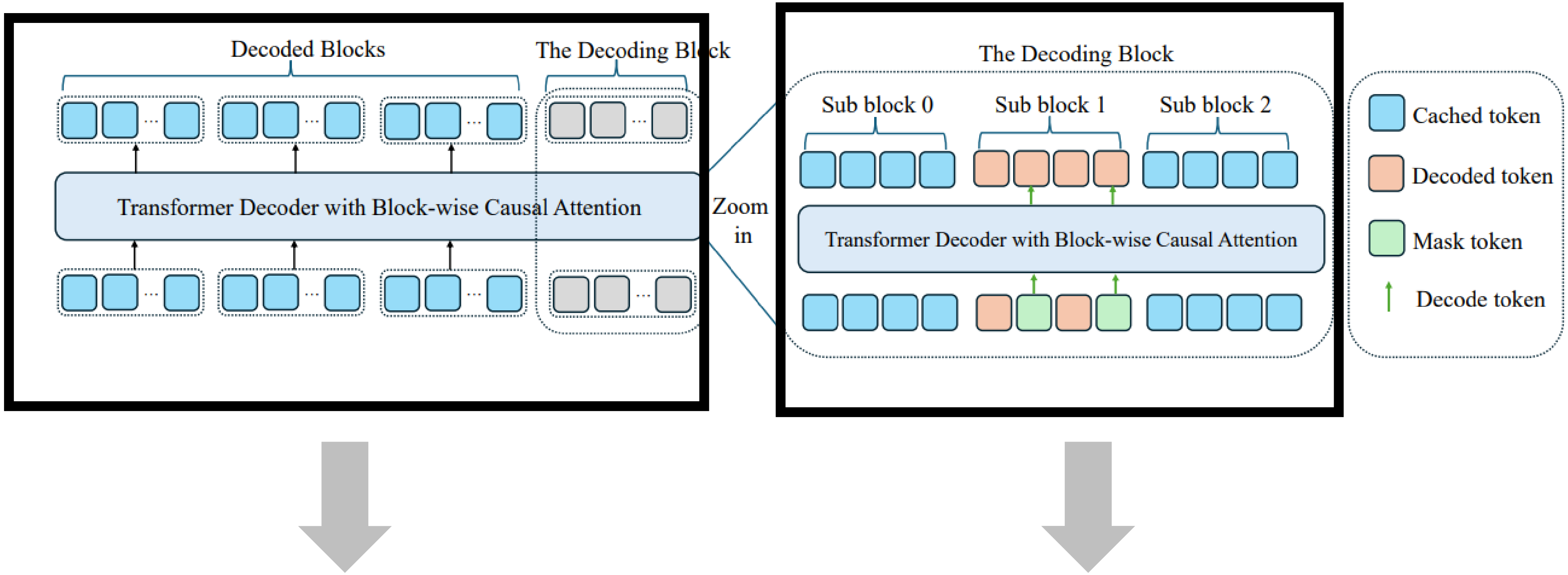
(a) Training-time attention mask.

x^b denote set of tokens in the b-th **block** (rather than the b-th **token**)

❖ employ the **flex-attention** implementation

Inference Pipeline

❑ **Block-wise** AR decoding with caching (Semi-AR)



❑ AR across blocks

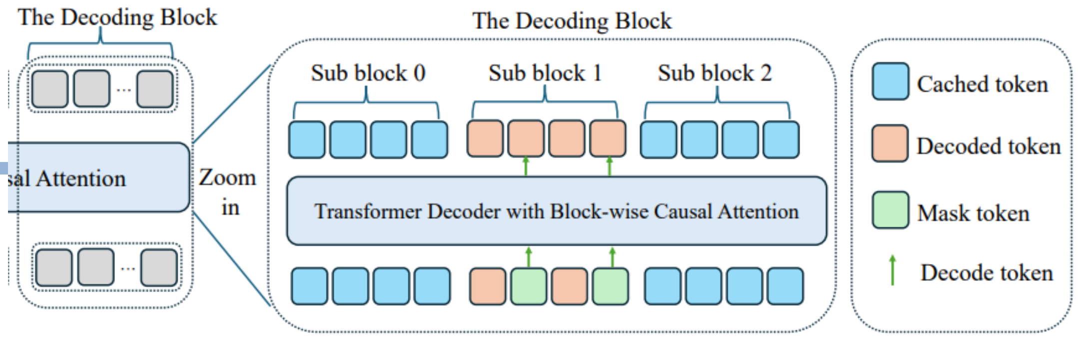
❖ Enable block-level KV cache

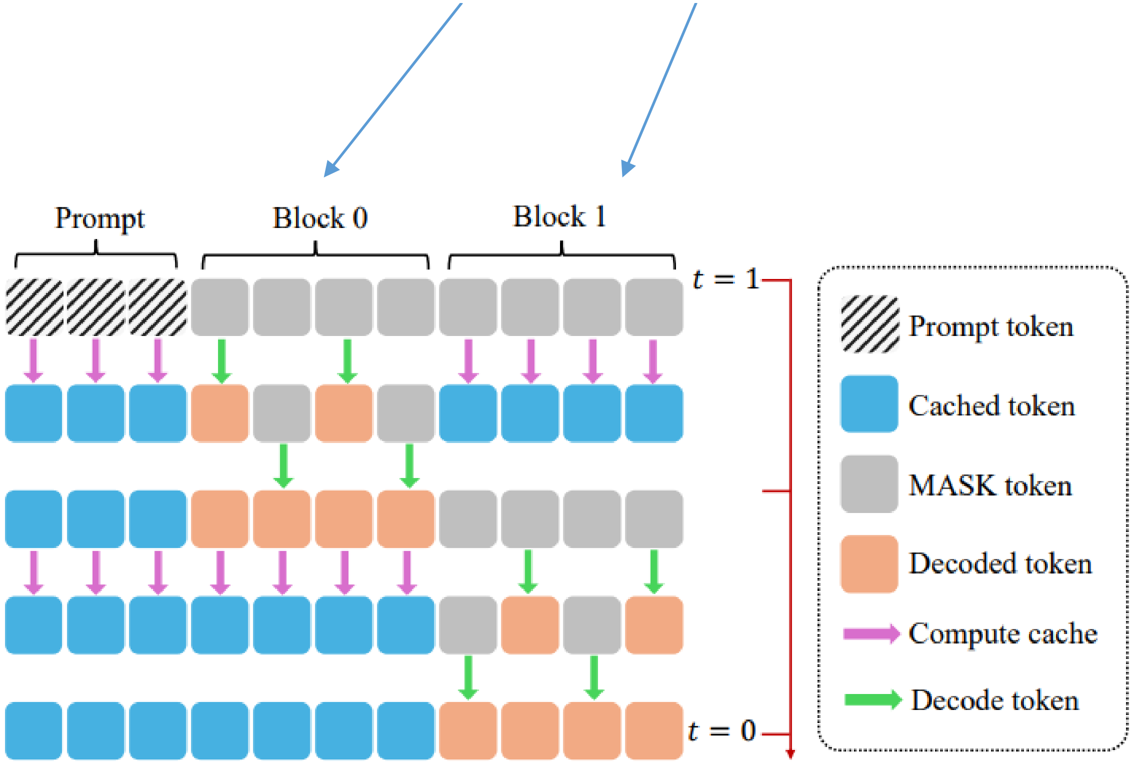
❑ Diffusion within blocks

Diffusion within blocks

Cannot leverage KV cache?

❑ **DualCache**： approximate KV cache



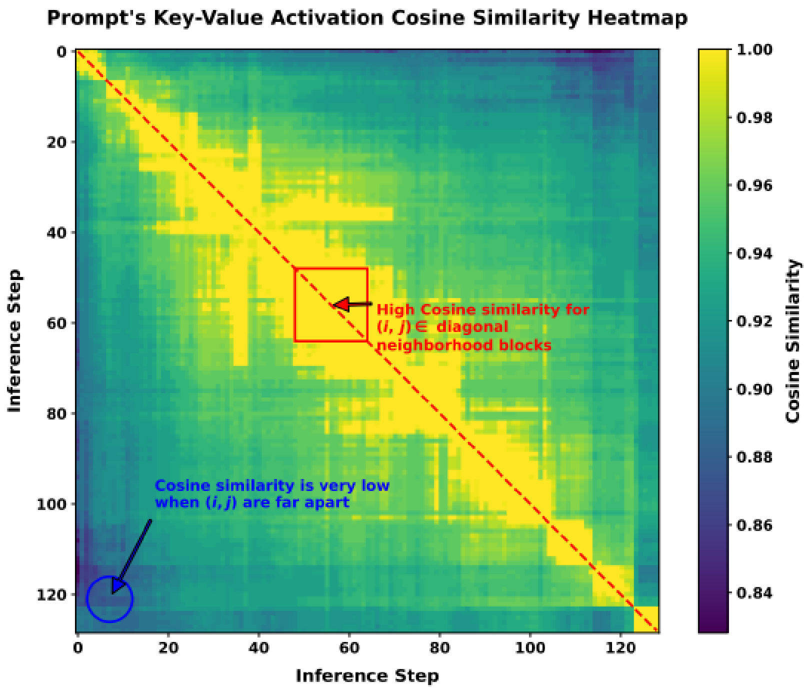


(b) **DualCache**: Bidirectional KV cache contains prefix and suffix Cache.

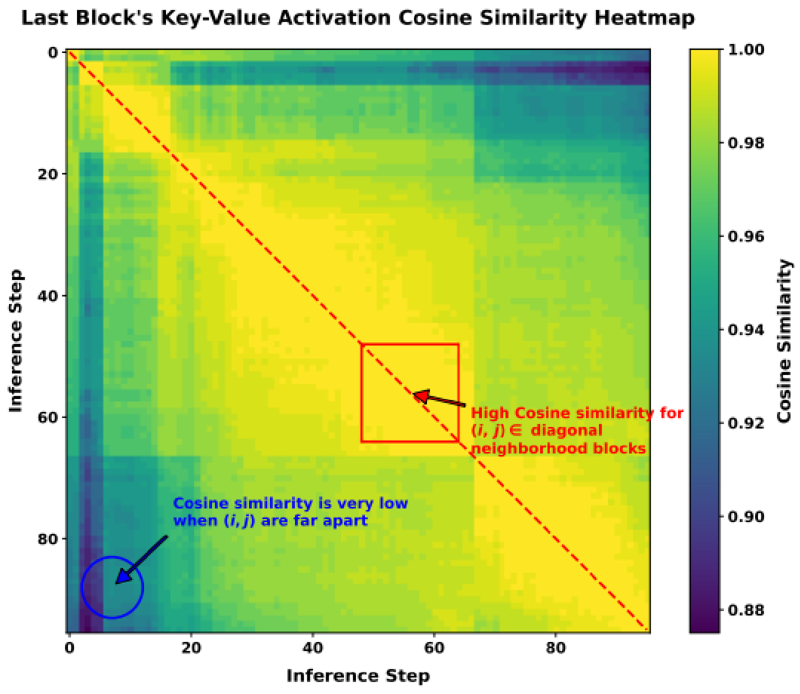
DualCache Intuition

Why works?

- Observation: KV activations exhibit **high similarity** across **adjacent** inference steps **within a block**

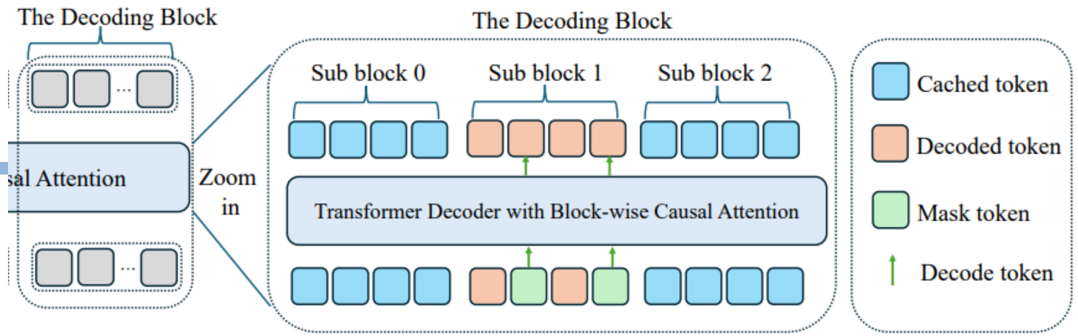


(a) Prompt block



(b) Last block

Diffusion within blocks

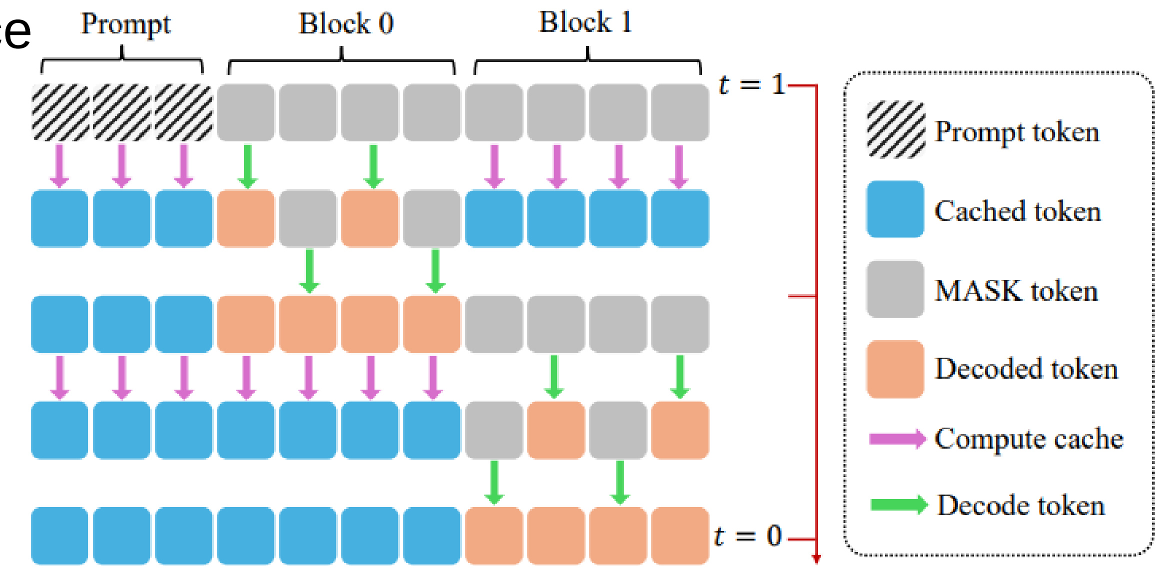


Cannot leverage KV cache?

❑ **DualCache**： approximate KV cache

❖ Cache previous kv in one sub block inference

❖ **Update** when current sub block is finished



(b) **DualCache**: Bidirectional KV cache contains prefix and suffix Cache.

Performance under different cache block size

- **Smaller** block size incur **overhead**
- **larger** block size **diminish** accuracy
- Trade-off

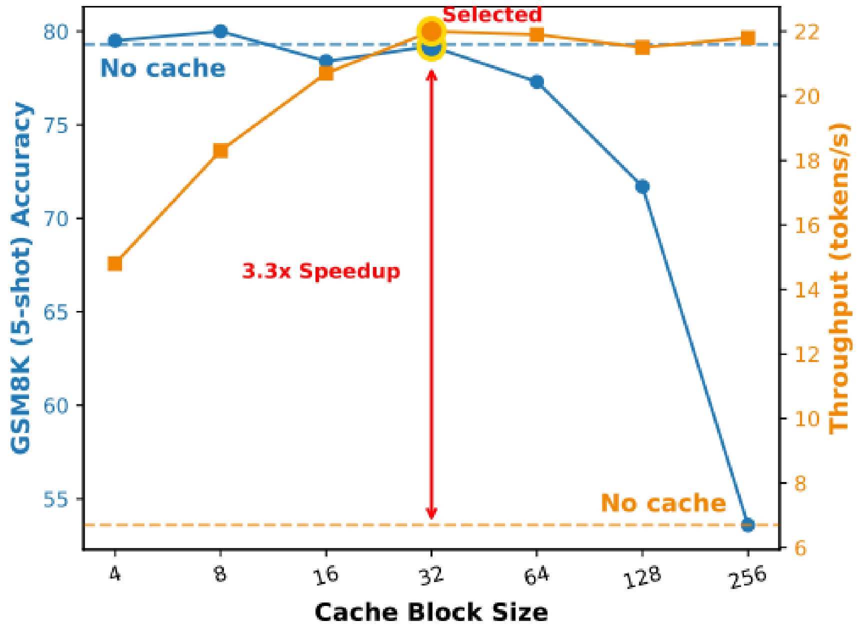
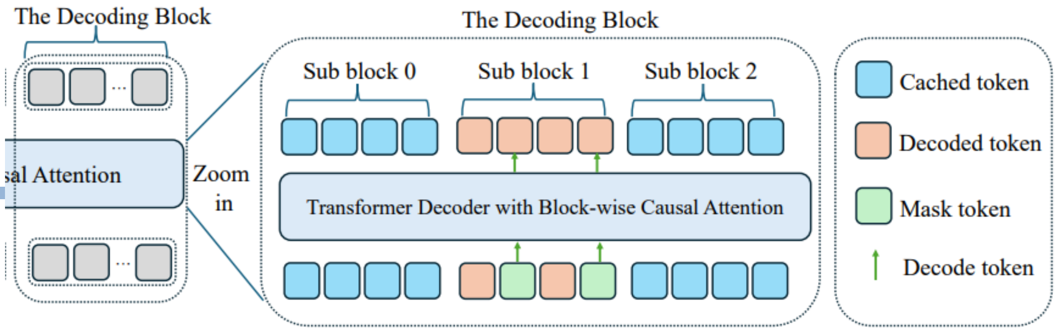


Figure 4 | **Impact of Cache Block Size on Accuracy and Throughput.** The orange line illustrates the effect of varying cache block size on throughput, while the blue line depicts accuracy.

Diffusion within blocks



❑ Confidence-Aware Parallel Decoding

(TL;DR) If the model is confident on many positions, parallel decoding will not introduce errors.

Theorem 1 (Parallel Decoding under High Confidence). *Suppose there exists a specific sequence of tokens* $\boxed{\boldsymbol{x}^*} = (x_{i_1}, \ldots, x_{i_n})$ *such that for each* $j \in \{1, \ldots, n\}$, *the model has high confidence in* $\boxed{x_{i_j}\text{: } p_j(X_{i_j} = x_{i_j}|E) > 1 - \epsilon}$ *for some small* $\epsilon > 0$. *Then, the following results hold:*

1. Equivalence for Greedy Decoding: $\boxed{\textit{If } (n+1)\epsilon \leq 1}$ *(i.e.,* $\epsilon \leq \frac{1}{n+1}$*), then*

$$\operatorname*{argmax}_{\boldsymbol{z}} \boxed{p(\boldsymbol{z}|E)} = \operatorname*{argmax}_{\boldsymbol{z}} \boxed{q(\boldsymbol{z}|E)} = \boldsymbol{x}^*. \tag{4}$$

This means that greedy parallel decoding (selecting $\operatorname{argmax} q$*) yields the same result as greedy sequential decoding (selecting* $\operatorname{argmax} p$*).*

This bound is tight: if $\epsilon > \frac{1}{n+1}$, *there exist distributions* $p(\boldsymbol{X}|E)$ *satisfying the high-confidence marginal assumption for which* $\operatorname{argmax}_{\boldsymbol{z}} p(\boldsymbol{z}|E) \neq \operatorname{argmax}_{\boldsymbol{z}} q(\boldsymbol{z}|E)$.

Diffusion within blocks

Algorithm 1 Block-wise Confidence-aware Parallel Decoding with (Dual) KV Cache

Require: $p_{\boldsymbol{\theta}}$, prompt p_0, answer length L, blocks K, block size B, steps per block T, threshold τ, `use_DualCache`, strategy $\in$ {`threshold`, `factor`}, factor f

$x \leftarrow [p_0;$ `[MASK]`, ..., `[MASK]`$]$

Initialize KV Cache (single or dual) for x (fuse with decoding). *// KV Cache Init*

for $k = 1$ to K **do**

  $s \leftarrow |p_0| + (k-1)B,\ e \leftarrow |p_0| + kB$

  for $t = 1$ to T **do**

    Use cache, run $p_{\boldsymbol{\theta}}$ on $x^{[s,e)}$ if `use_DualCache` else $x^{[s,:)}$ *// Cache Reuse*

    For masked x^i, compute confidence $\boxed{c^i = \max_x p_{\boldsymbol{\theta}}(x^i|\cdot)}$ *// Confidence scoring*

    if `strategy` == `threshold` **then**

      Unmask all i in $[s, e)$ with $c^i \geq \tau$, always unmask max c^i

    else if `strategy` == `factor` **then**

      Sort c^i in descending order as $(c^{(1)}, c^{(2)}, \ldots, c^{(m)})$

      $\boxed{\text{Find largest } n}$ such that $\boxed{(n+1)(1-c^{(n)}) < f}$

      Unmask top-n tokens, always unmask the max c^i

    end if

    if all $x^{[s,e)}$ unmasked **then**

      break

    end if

  end for

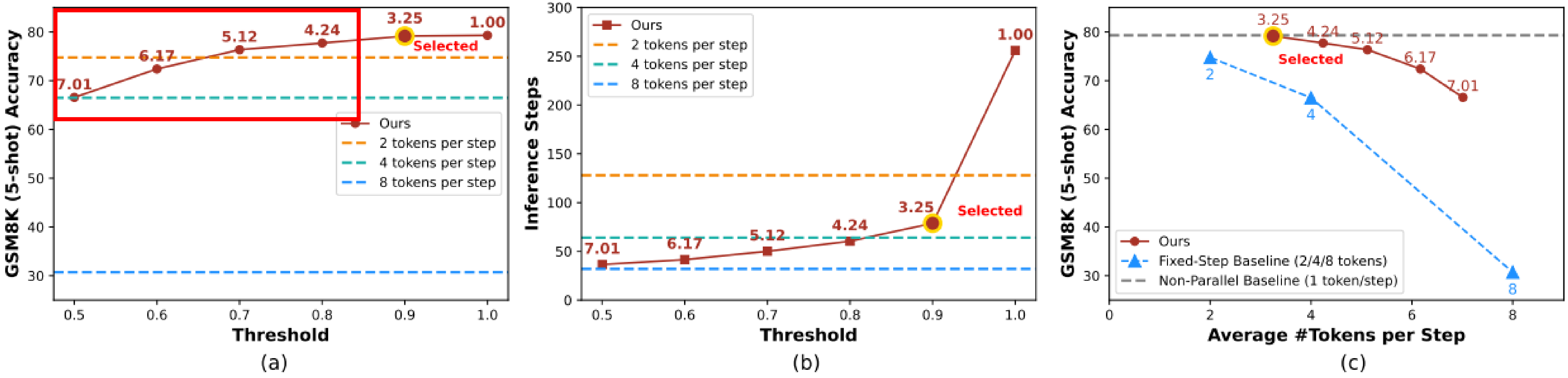
  Update KV cache: **if** `use_DualCache`**:** prefix & suffix; **else:** prefix. *// Cache Update*

end for

return x

Unmask **dynamic** number of tokens

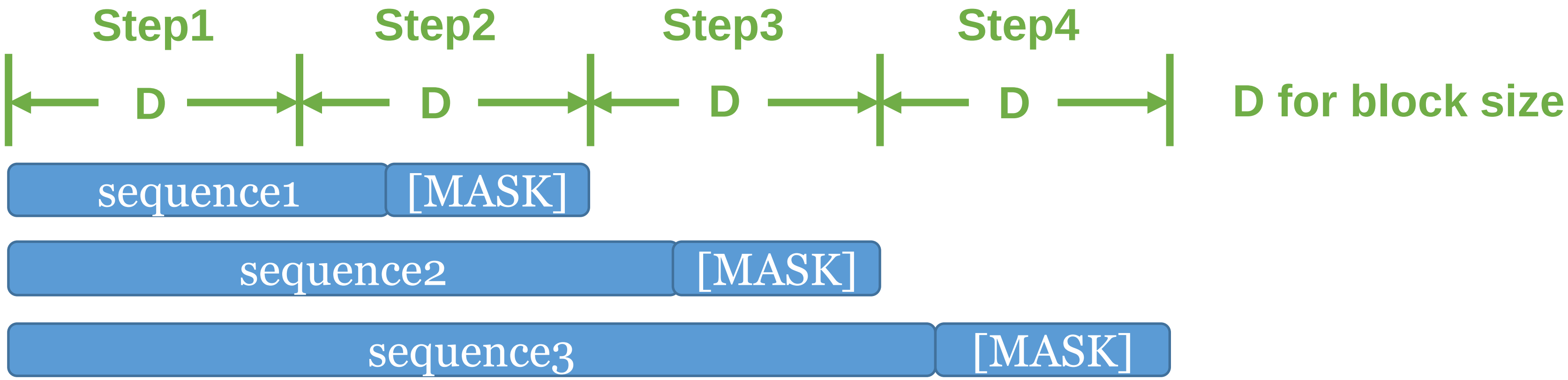
Threshold Curve



❖ Baseline： llada previous fixed-N tokens

❖ Setting： GSM8K, 256 length

Batch decoding with padding



- ❑ Adaptation to block diffusion

Outline

- Background
- Design
- Evaluations
- Discussion

Setups

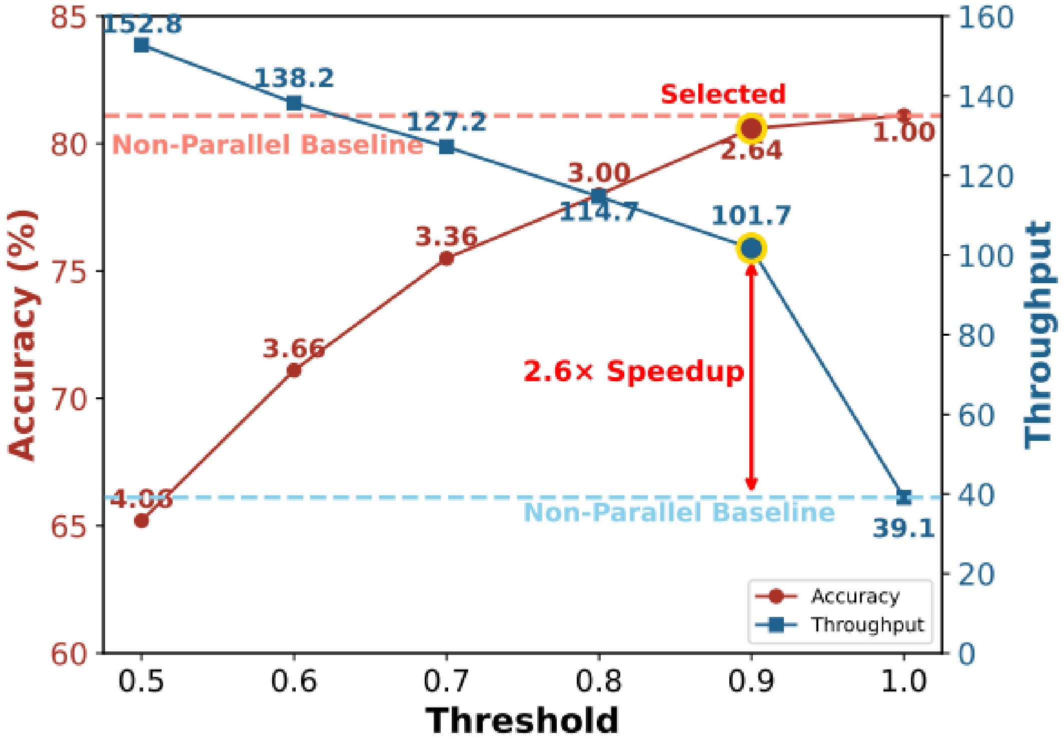
❑ Tuning model:	Qwen-2.5 1.5B and 7B instruct
❑ Training dataset:	LLaMA-Nemotron post-training dataset (batch 256)
❑ Training environment:	64 NVIDIA A100 GPUs
❑ Training configuration:	1.5B: learning rate 2×10^{-5} for 6,000 steps (costs 8h)
	7B: learning rate 1×10^{-5} for 2500 steps (costs 12h)
❑ Block size:	32
❑ Sub-Block size:	8

Benchmark Results

Model	#Params	HumanEval		MBPP		GSM8K	Math	IFEval	MMLU	GPQA	Avg.
		Base	Plus	Base	Plus						
1B Models											
LlaMA-3.2	1.2B	34.1	31.1	34.1	29.4	43.0	23.8	58.9	44.4	24.1	35.9
SmolLM 2	1.7B	34.1	28.7	50.6	46.0	47.7	21.1	55.1	49.1	29.2	40.7
Qwen2.5-1.5B	1.5B	42.1	37.2	48.1	41.3	57.0	46.8	41.2	54.6	30.6	44.3
Qwen2.5-1.5B-Nemo-FT	1.5B	37.2	33.5	53.4	44.4	58.5	43.5	39.4	58.1	31.0	44.3
Fast-dLLM v2	1.5B	43.9	40.2	50.0	41.3	62.0	38.1	47.0	55.1	27.7	**45.0**
7B+ Models											
LLaDA	8B	35.4	31.7	31.5	28.6	78.6	26.6	59.9	65.5	31.8	43.3
LLaDA-1.5	8B	52.4	-	42.8	-	83.3	42.6	58.2	66.0	36.9	-
LLaDA-MoE	7B	61.6	-	70.0	-	82.4	58.7	59.3	67.2	-	-
Dream	7B	57.9	53.7	68.3	56.1	81.0	39.2	62.5	67.0	33.0	57.6
Qwen2.5-7B	7B	51.2	47.6	57.7	49.5	71.4	73.3	70.8	68.7	33.5	58.2
Qwen2.5-7B-Nemo-FT	7B	52.4	48.2	57.1	50.0	84.1	72.0	69.5	68.6	34.2	59.6
Fast-dLLM v2	7B	63.4	58.5	63.0	52.3	83.7	61.6	61.4	66.6	31.9	**60.3**

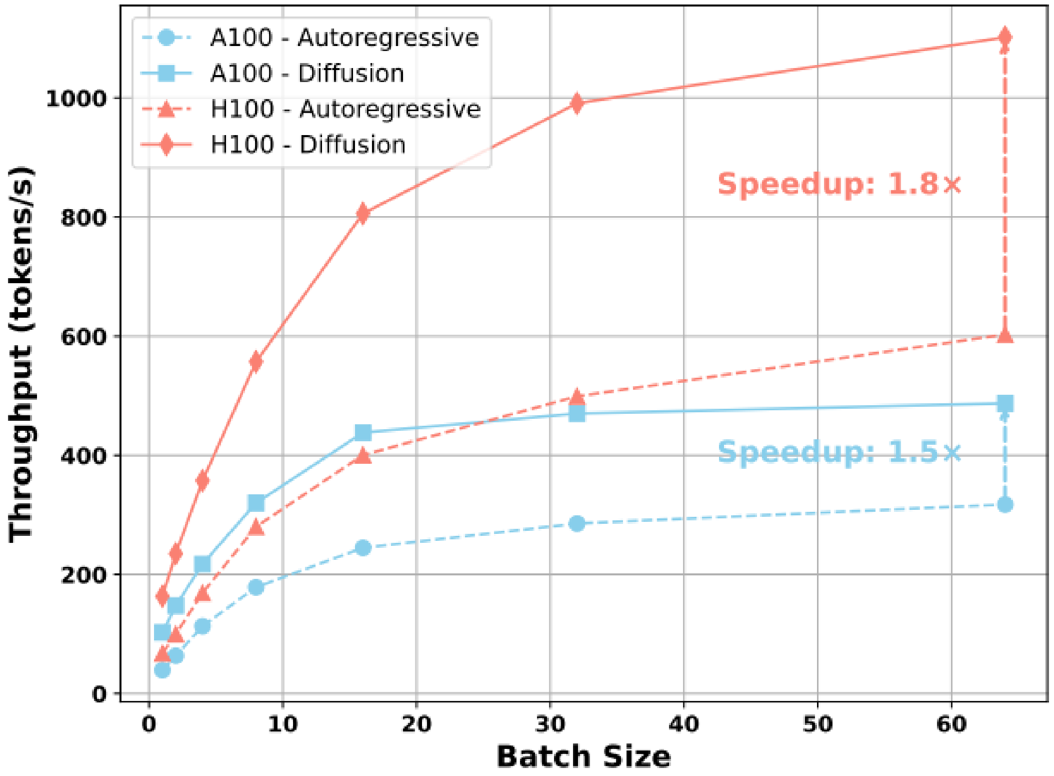
Performance

❑ **Accuracy** and **throughput** under different thresholds on GSM8K



Performance

- ❑ **Throughput** comparison between AR(Qwen2.5-7B-Instruct) and diffusion(Fast-dLLM v2 7B) generation for GSM8K

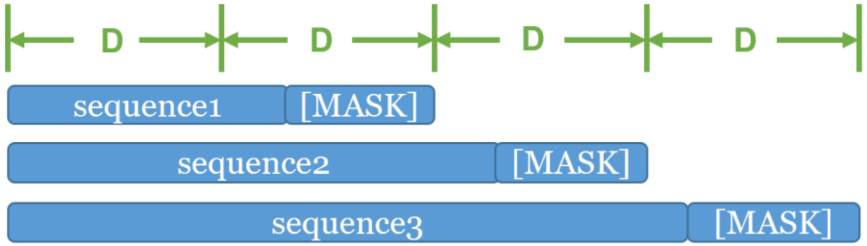


Ablation Study

- Based on Fast-dLLM v2 1.5B

Method	HumanEval		MBPP		GSM8K	Math	IFEval	MMLU	GPQA	Avg.
	Base	Plus	Base	Plus						
Naive token shift	38.4	32.9	44.4	38.6	59.0	37.3	39.9	52.9	**27.9**	41.3
+ pad	38.4	34.1	45.2	38.4	60.1	37.0	45.8	53.5	27.7	42.2
+ pad + CM	**43.9**	**40.2**	**50.0**	**41.3**	**62.0**	**38.1**	**47.0**	**55.1**	27.7	**45.0**

- Naive token shift: use hidden state at i-1 to predict x_i
- Pad:



- CM: complementary mask

Ablation Study

- ❑ Sub-Block size and Block size affect performance

Table 3 | Sub-Block size decoding improves performance, with size 8 being optimal.

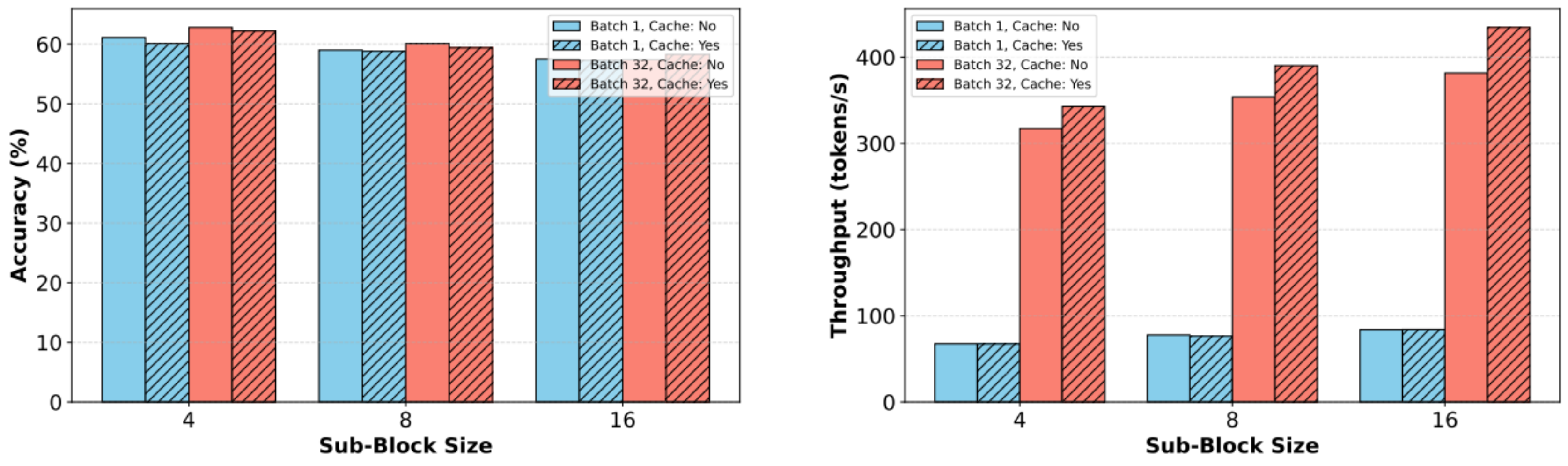
Sub-Block Size	2	4	8	16	32
GSM8K	**62.8**	61.8	<u>62.0</u>	61.3	60.2
HumanEval	42.7	<u>43.3</u>	**43.9**	39.6	38.4
HumanEval+	<u>39.6</u>	**40.2**	**40.2**	36.0	34.8

Table 4 | Inference with mismatched sizes reduces performance.

Block Size	2	4	8	16	32
GSM8K	53.2	56.8	58.5	<u>59.7</u>	**60.2**
HumanEval	37.8	**43.3**	**43.3**	<u>38.4</u>	<u>38.4</u>
HumanEval+	34.1	<u>39.0</u>	**39.6**	34.1	34.8

Ablation Study

❑ Cache is a purely efficiency-enhancing feature



(a) (b)

Figure 6 | Effect of small block size and sub-block cache on model performance. (**a**) Accuracy remains largely unaffected by the use of sub-block cache across different block sizes and batch sizes. (**b**) Throughput increases as small block size grows due to higher decoding parallelism. While sub-block cache has negligible effect when batch size is small, it significantly improves throughput under compute-bound settings (e.g., batch size = 32).

Outline

- Background
- Design
- Evaluations
- **Discussion**

Discussion

- ❑ **Trade-off between accuracy and throughput in MDMs**
  - ❖ Parallel generation inevitably introduces conditional independence
- ❑ **Arbitrary-Order Autoregressive?**
  - ❖ Semi-AR Semi-diffusion
- ❑ **Data-efficient fine-tuning**
  - ❖ Fast-dLLM v2 achieves lossless adaptation with just ~1B tokens, compared to ~500B tokens required by Dream
  - ❖ Diffusion Beats Autoregressive in Data-Constrained Settings (NeurIPS 25)

Thanks!