Fast-dLLM: Training-free Acceleration of Diffusion LLM by Enabling KV Cache and Parallel Decoding

Presenter: Chengyue Wu

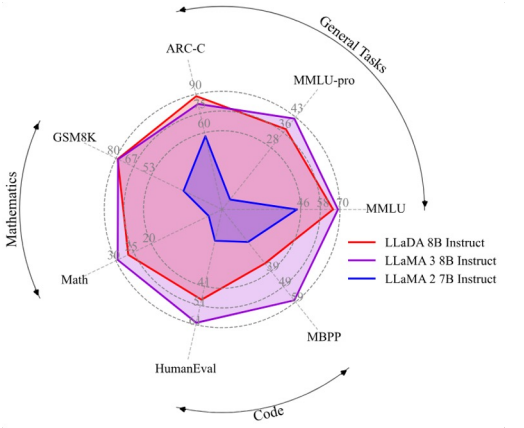
Why Diffusion LLM

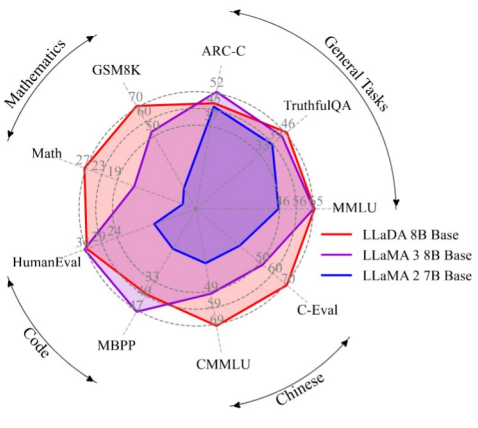
- Mercury and Gemini Diffusion LLM show **superior speed** vs AR LLM with **comparable performance**

	Mercury Coder Mini	Mercury Coder Small	Gemini 2.0 Flash-Lite	Claude 3.5 Haiku	GPT-4o Mini	Qwen 2.5 Coder 7B
Throughput (toks/sec)	1109	737	201	61	59	207
HumanEval	88.0	90.0	90.0	86.0	88.0	90.0
MBPP	77.1	76.6	75.0	78.0	74.6	80.0
EvalPlus	78.6	80.4	77.3	75.1	78.5	79.3
MultiPL-E	74.1	76.2	79.5	72.3	72.0	75.3
LiveCodeBench	17.0	25.0	18.0	31.0	23.0	9.0
BigCodeBench	42.0	45.5	44.4	45.4	46.8	41.4
Fill-in-the-Middle	82.2	84.8	60.1	45.5	60.9	56.1

Open-Sourced Masked Diffusion LLM

- **Comparable performance** vs AR LLM but **MUCH LOWER**





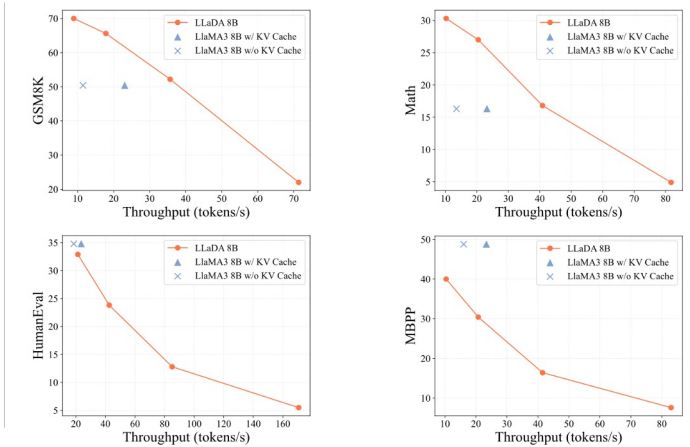


Figure 5: **Analysis of Sampling Efficiency.** The generation length for LLaDA is set to 256, with sampling steps set to 32, 64, 128, and 256 across the figures. LLaDA enables a flexible trade-off between generation quality and sampling speed.

Overview of MDM

- Pretrain: Randomly predict the mask token with ratio t~U(0,1), full attention
- SFT: Only predict the mask token at the response part
- Inference with multi-steps. Each step predict all the tokens while preserve few.

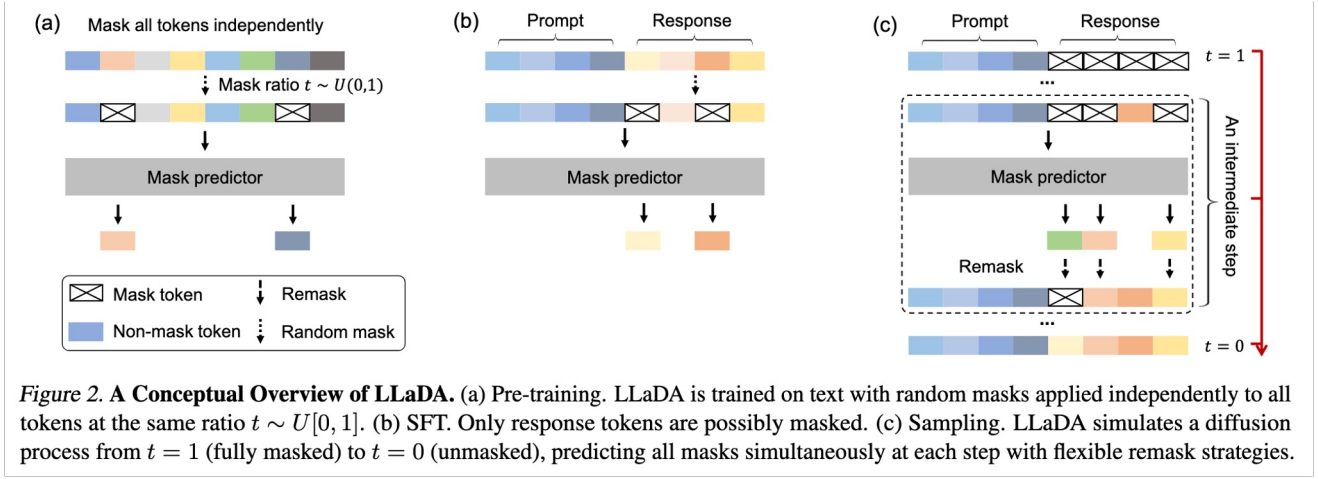
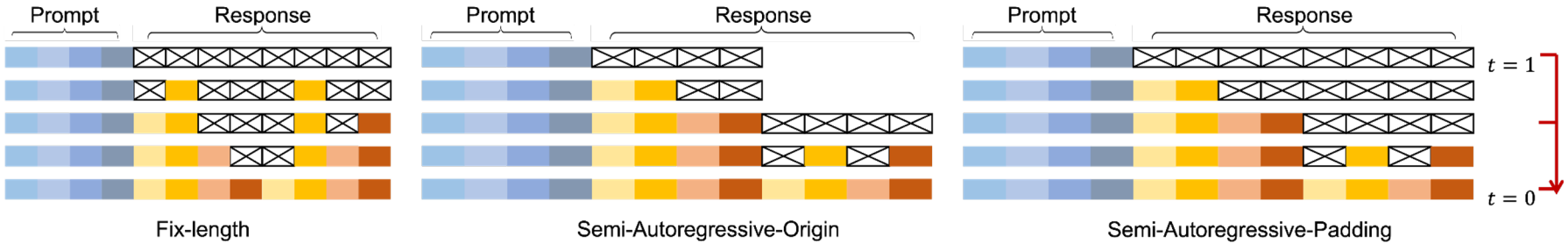


Figure 2. **A Conceptual Overview of LLaDA.** (a) Pre-training. LLaDA is trained on text with random masks applied independently to all tokens at the same ratio $t \sim U[0, 1]$. (b) SFT. Only response tokens are possibly masked. (c) Sampling. LLaDA simulates a diffusion process from $t = 1$ (fully masked) to $t = 0$ (unmasked), predicting all masks simultaneously at each step with flexible remask strategies.

Inference details

- There are 3 ways to do generation for MDM
  - Fix-length: can only generate a fixed number of tokens
  - Semi-AR-origin: generate blocks by blocks, each blocks with fixed number of tokens
  - Semi-AR-Padding: for a fixed number of tokens (may be very long), generate each block from left to right
- All the methods discard tokens after the first <EOS> (end of sentence) token.
- For each step in the above three sampling processes, MDM first predicts all masked tokens simultaneously. A certain proportion of tokens are remasked: randomly remasking or low-confidence remasking.



Limitations

- Efficiency is even worse than AR (for open sourced MDM, not mercury)
  - LLaDA samples with a fixed context length;
  - **LLaDA cannot yet leverage techniques like KV-Cache;**
  - LLaDA achieves optimal performance when the number of sampling steps equals the response length. **Reducing the number of sampling steps leads to a decrease in performance.**
  - Full attention is slower than causal attention
- Masked token prediction is **independent**

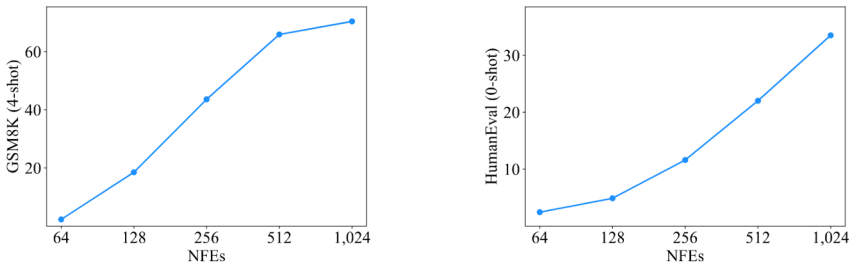
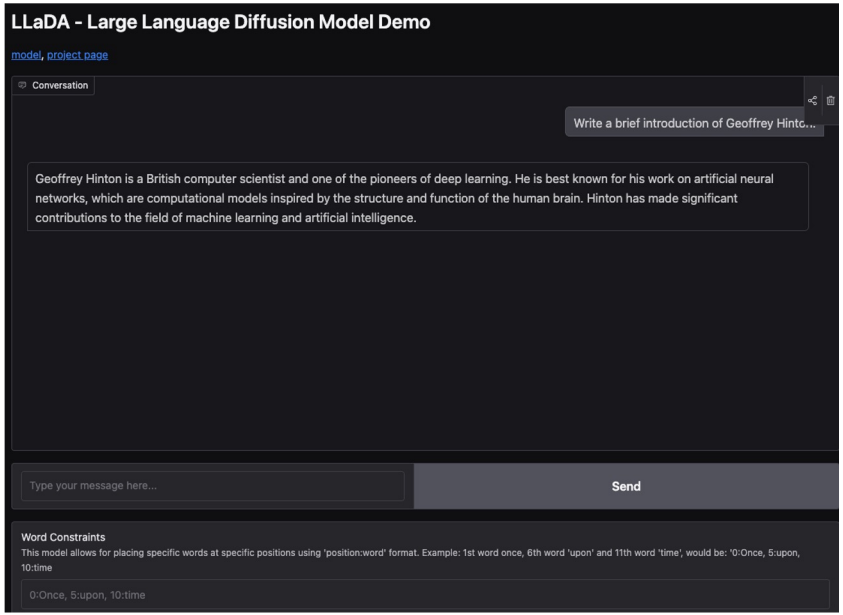
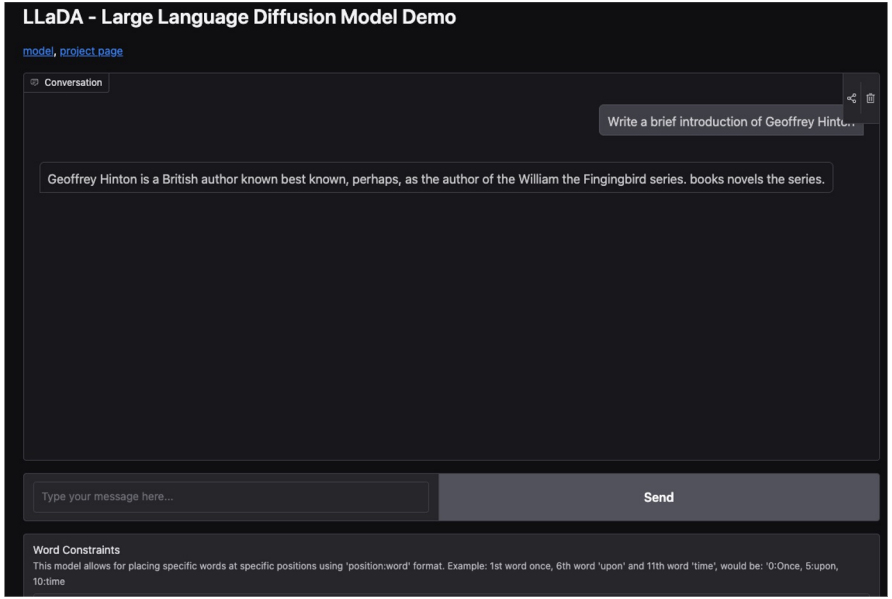


Figure 5. **Analysis of Sampling Steps.**

Independent Sampling Case

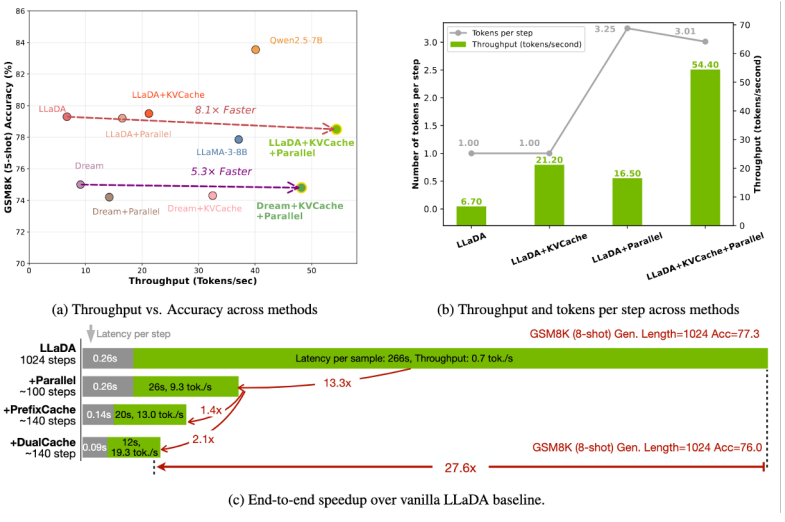
- Prompt: Write a brief introduction of Geoffrey Hinton. Left with 64 steps, right with 8 steps





Acceleration Results

- We improve the parallel tokens **3x** by designing a new **dynamic inference schedule** based on confidence
- Implement a approximate kv cache for diffusion llm and achieve **3x** acceleration
- Combine these two designs, we achieve **9x** speed up under the setting of GSM8k，256 generation length, bsz=1 where one AR baseline is llama3-8B with throughput of 37 tokens/s in A100
- When the gen length and prefill length get longer, we achieve over **27x** acceleration with minimum degradation



(a) Throughput vs. Accuracy across methods

(b) Throughput and tokens per step across methods

(c) End-to-end speedup over vanilla LLaDA baseline.

Acceleration Dimension (1)

- Parallel Sampling (output multiple tokens per step)

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

Require: p_θ, prompt p_0, answer length L, blocks K, block size B, steps per block T, threshold τ, `use_DualCache`

$x \leftarrow [p_0; \texttt{[MASK]}, ..., \texttt{[MASK]}]$
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_θ on $x^{[s,e)}$ if `use_DualCache` else $x^{[s,:)}$ *// Cache Reuse*
    For masked x^i, compute confidence $c^i = \max_x p_\theta(x^i|\cdot)$ *// Confidence scoring*
    Unmask all i in $[s, e)$ with $c^i \geq \tau$, always unmask max c^i *// Parallel decoding*
    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

Mathematical Intuition

Mathematical Proof

Let the known tokens be denoted as the set w.
Let i and j be the indices of two unknown tokens, and let k_1, k_2 be their possible values.

According to the conditions given:

1. $p(i = k_1|w) \geq 1 - \epsilon$ where $\epsilon \to 0$
2. $p(j = k_2|w) \geq 1 - \delta$ where $\delta \to 0$

Goal:
Prove that $p(i = k_1, j = k_2|w) \geq 1 - O(\epsilon + \delta)$

- Main idea: two large marginal probabilities can lead to large combination probabilities (avoid independence)

Proof

By the **probability addition rule** for any two events A, B:

$$p(A \cup B) = p(A) + p(B) - p(A \cap B)$$

Rewriting:

$$p(A \cap B) = p(A) + p(B) - p(A \cup B)$$

Since probabilities cannot exceed 1 ($p(A \cup B) \leq 1$), we have:

$$p(A \cap B) \geq p(A) + p(B) - 1$$

Let A be the event "$i = k_1$", B be the event "$j = k_2$" (both conditioned on w). Then:

$$p(i = k_1, j = k_2 \mid w) \geq p(i = k_1 \mid w) + p(j = k_2 \mid w) - 1$$

Substitute the given lower bounds:

$$p(i = k_1, j = k_2 \mid w) \geq (1 - \epsilon) + (1 - \delta) - 1 = 1 - \epsilon - \delta$$

Therefore,

$$p(i = k_1, j = k_2 \mid w) \geq 1 - (\epsilon + \delta) = 1 - O(\epsilon + \delta)$$

Q.E.D.

Mathematical Proof

- We need to prove the argmax of marginal distribution is equal to the joint distribution

Prior to presenting the theorem, we will define the mathematical notation used in its statement. Let $p_{\boldsymbol{\theta}}(\cdot|E)$ denote the conditional probability mass function (PMF) given by an MDM condition on E (comprising a prompt p_0 and previously generated tokens). Suppose the model is to predict n tokens for positions $i_1, \ldots, i_n$ not in E. Let $\boldsymbol{X} = (X_{i_1}, \ldots, X_{i_n})$ be the vector of n tokens, where each X_{i_j} takes values in vocabulary $\mathcal{V}$. Let $p(\boldsymbol{X}|E) \equiv p_{\boldsymbol{\theta}}(X_{i_1}, \ldots, X_{i_n}|E)$ be the joint conditional PMF according to the model. Let $p_j(X_{i_j}|E) \equiv p_{\boldsymbol{\theta}}(X_{i_j}|E)$ be the marginal conditional PMF for position i_j. Parallel decoding generates tokens using the product of marginals: $q(\boldsymbol{X}|E) = \prod_{j=1}^{n} p_j(X_{i_j}|E)$. The proof of Theorem 1 and relevant discussions are in Appendix A.

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

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

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

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* $\operatorname{argmax}_{\boldsymbol{z}} p(\boldsymbol{z}|E) \neq \operatorname{argmax}_{\boldsymbol{z}} q(\boldsymbol{z}|E)$.

Mathematical Proof (step 1)

Proof. **Step 1: Show that x^* is the unique maximizer of $q(x)$.**

Let $p_j^* = p_j(X_{i_j} = x_{i_j}|E)$. We are given $p_j^* > 1 - \epsilon$. Let $\epsilon_j' = 1 - p_j^* = p_j(X_{i_j} \neq x_{i_j}|E)$. Thus, $\epsilon_j' < \epsilon$. The product-of-marginals probability mass function (PMF) is

$$q(\boldsymbol{z}|E) = \prod_{j=1}^{n} p_j(X_{i_j} = z_j|E).$$

To maximize $q(\boldsymbol{z}|E)$, we must maximize each term $p_j(X_{i_j} = z_j|E)$ independently. The condition $(n+1)\epsilon \leq 1$ implies $\epsilon \leq 1/(n+1)$. Since $n \geq 1$, it follows that $1/(n+1) \leq 1/2$. So, $\epsilon \leq 1/2$. Therefore, for the chosen x_{i_j}:

$$p_j^* = p_j(X_{i_j} = x_{i_j}|E) > 1 - \epsilon \geq 1 - 1/2 = 1/2.$$

This means x_{i_j} is the unique maximizer for $p_j(\cdot|E)$. So,

$$\underset{\boldsymbol{z}}{\operatorname{argmax}}\, q(\boldsymbol{z}|E) = (x_{i_1}, \ldots, x_{i_n}) = \boldsymbol{x}^*.$$

Mathematical Proof (step 2)

Step 2: Show that $\boldsymbol{x}^*$ is the unique maximizer of $p(x)$.

We want to show $p(\boldsymbol{x}^*|E) > p(\boldsymbol{z}|E)$ for all $\boldsymbol{z} \neq \boldsymbol{x}^*$. Using the Bonferroni inequality:

$$p(\boldsymbol{x}^*|E) = p(\cap_{j=1}^n \{X_{i_j} = x_{i_j}\}|E) \geq 1 - \sum_{j=1}^n p(X_{i_j} \neq x_{i_j}|E) = 1 - \sum_{j=1}^n \epsilon'_j.$$

Since $\epsilon'_j < \epsilon$ for all j, we have $\sum_{j=1}^n \epsilon'_j < n\epsilon$. So,

$$p(\boldsymbol{x}^*|E) > 1 - n\epsilon.$$

Now consider any $\boldsymbol{z} = (z_1, \ldots, z_n)$ such that $\boldsymbol{z} \neq \boldsymbol{x}^*$. This means there is at least one index k such that $z_k \neq x_{i_k}$. The event $\{\boldsymbol{X} = \boldsymbol{z}\}$ is a sub-event of $\{X_{i_k} = z_k\}$. So,

$$p(\boldsymbol{z}|E) \leq p_k(X_{i_k} = z_k|E).$$

Since $z_k \neq x_{i_k}$,

$$p_k(X_{i_k} = z_k|E) \leq p_k(X_{i_k} \neq x_{i_k}|E) = \epsilon'_k < \epsilon.$$

Thus,

$$p(\boldsymbol{z}|E) < \epsilon.$$

For $p(\boldsymbol{x}^*|E) > p(\boldsymbol{z}|E)$ to hold, it is sufficient that

$$1 - n\epsilon \geq \epsilon,$$

which simplifies to $1 \geq (n+1)\epsilon$, or $\epsilon \leq \frac{1}{n+1}$. The theorem assumes $(n+1)\epsilon < 1$, which is exactly this condition. The strict inequalities $p(\boldsymbol{x}^*|E) \geq 1 - \sum \epsilon'_j > 1 - n\epsilon$ and $p(\boldsymbol{z}|E) \leq \epsilon'_k < \epsilon$ ensure that $p(\boldsymbol{x}^*|E) > p(\boldsymbol{z}|E)$. Thus,

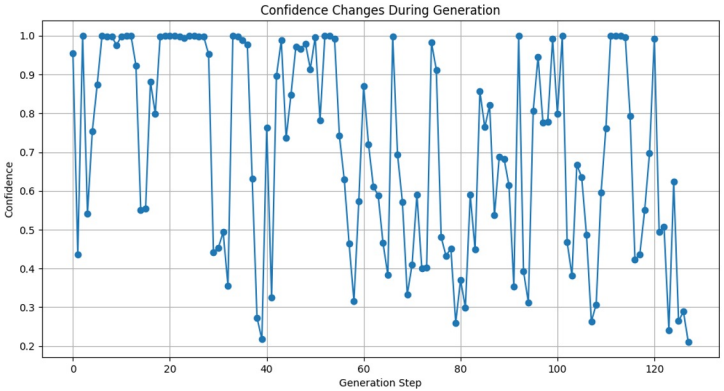
$$\operatorname*{argmax}_{\boldsymbol{z}} p(\boldsymbol{z}|E) = \boldsymbol{x}^*.$$

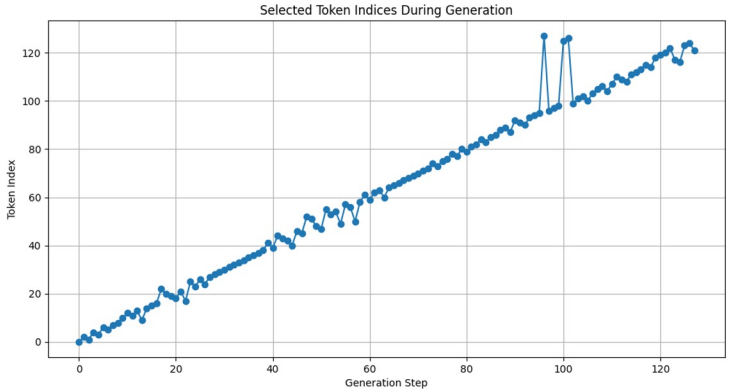
Combined with the argmax of q, this proves the main statement of Part 1:

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

Confidence Qualitative Results

- Analyze the token idx and confidence of each step during 128 steps inference (gen. length=128, block size=32)
- Overall the confidence is relatively high and the order is from left to right





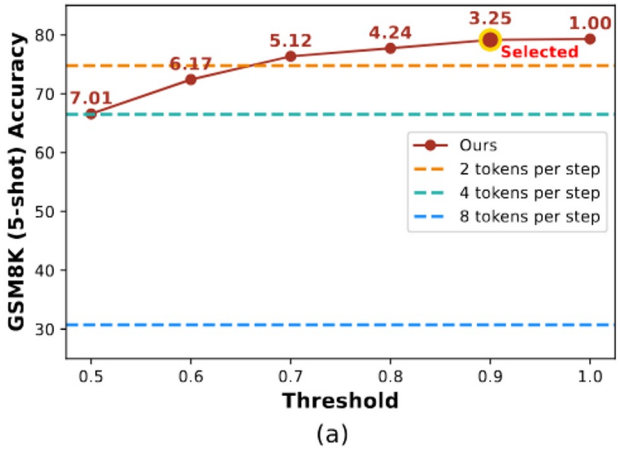
Dynamic Threshold

- We also demonstrate that based on the mathematical proof, we can use a dynamic threshold to select the tokens that have the confidence to be selected “safely” with (n+1)*eps<1.
- The results are not reported in the paper but we find that it raises the throughput from 54 to 78 tokens/sec at GSM8K-5shot with the same accuracy.

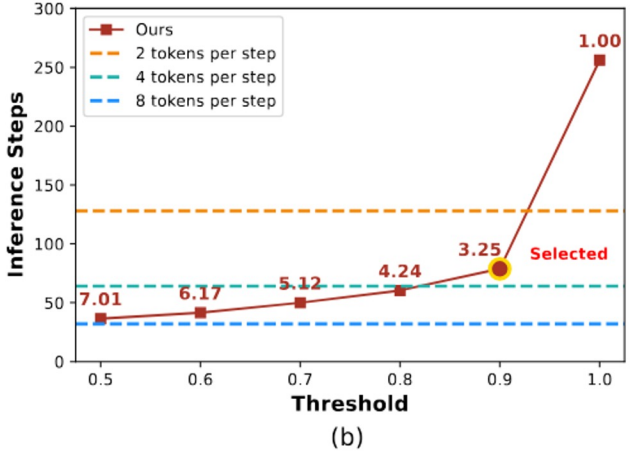
```
factor = 1
for j in range(confidence.shape[0]):
    ns=list(range(1,num_transfer_tokens[j]+1))
    es=[factor/(n+1) for n in ns]
    threshs=[1-e for e in es]

    # at least one token is transferred
    threshs[0]=-1
    sorted_confidence=torch.sort(confidence[j][mask_index[j]],dim=-1,descending=True)[0]
    assert len(sorted_confidence)==len(threshs)
    for top_i in range(len(threshs)):
        if sorted_confidence[top_i]<threshs[top_i]:
            break

    if top_i == 0 or top_i == len(threshs)-1:
        top_i+=1


    _, select_index = torch.topk(confidence[j], k=top_i)
    transfer_index[j, select_index] = True
```

Threshold curve

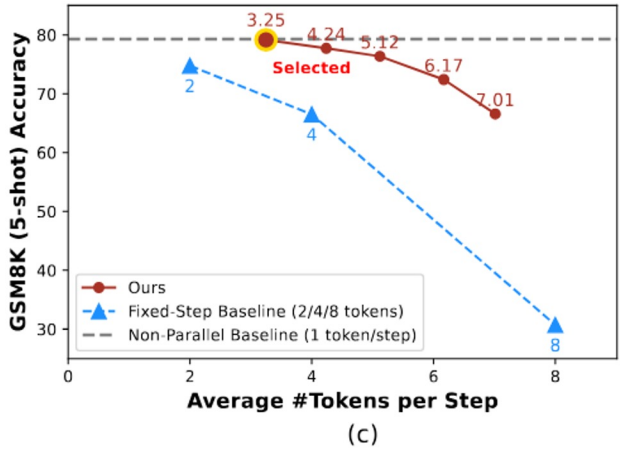
- Under the setting of GSM 8K，256 length, we evaluate different thresholds compared with llada previous fixed-N tokens baseline
- Our method has **great robustness** towards threshold and **improve the performance at large parallel scale.**



(a)



(b)



(c)

Acceleration Dimension (2)

- KV Cache for Diffusion LLM

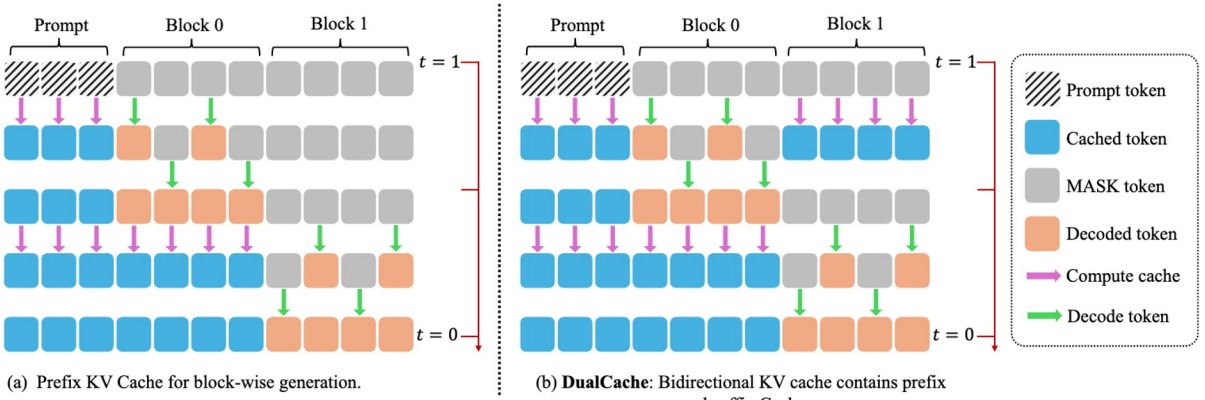
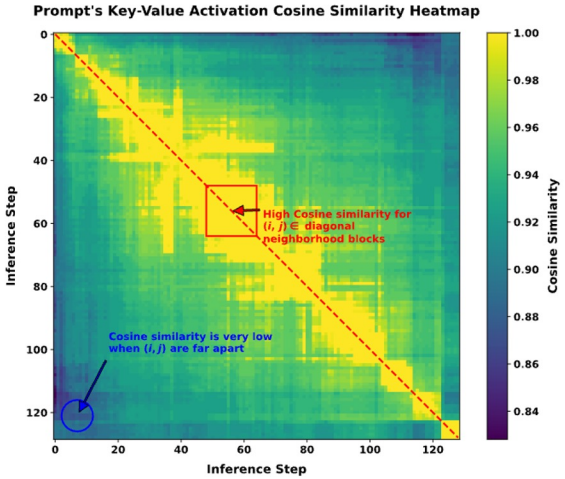


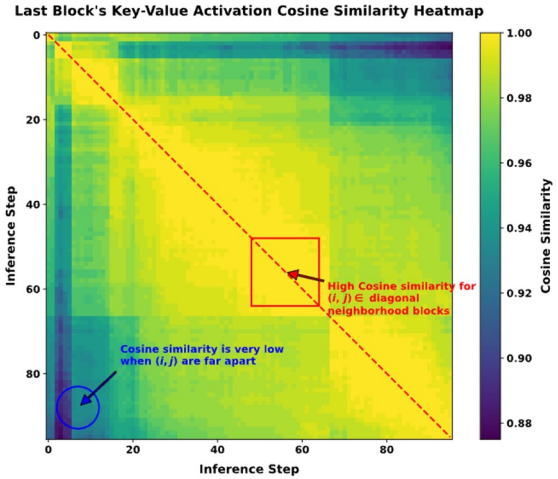
Figure 2 | **Illustration of our Key-Value Cache for Block-Wise Decoding.** (a) During prefix-only caching, the KV cache is computed once for the prompt and reused across multiple decoding steps within each block. The cache is updated after completing a block to maintain consistency, with negligible overhead. (b) DualCache extends this approach by caching both prefix and masked suffix tokens, further accelerating decoding. The high similarity of KV activations across steps allows effective reuse with minimal approximation error.

Cache Intuition

- Visualize the **KV of the prompt** after every inference step and find that **within a block the variance is relatively small,** which means we can **cache previous kv within one block inference** and **update it when the current block is finished.**
- Value at position (i, j) represent f(kv_i, kv_j), where f is a similarity metric, kv_i and kv_j is the kv activation at step i and j. When i is close to j, the similarity is relative high



(a) Prompt block



(b) Last block

Performance under different cache block size

- Smaller block sizes tend to maximize accuracy but incur overhead due to frequent cache updates
- Larger block sizes may diminish accuracy owing to increased context mismatch.
- Block size of 32 achieves the best trade-off

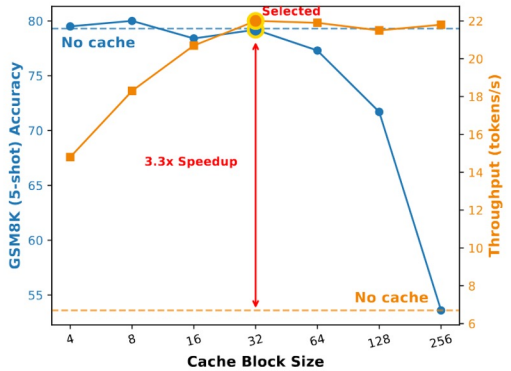
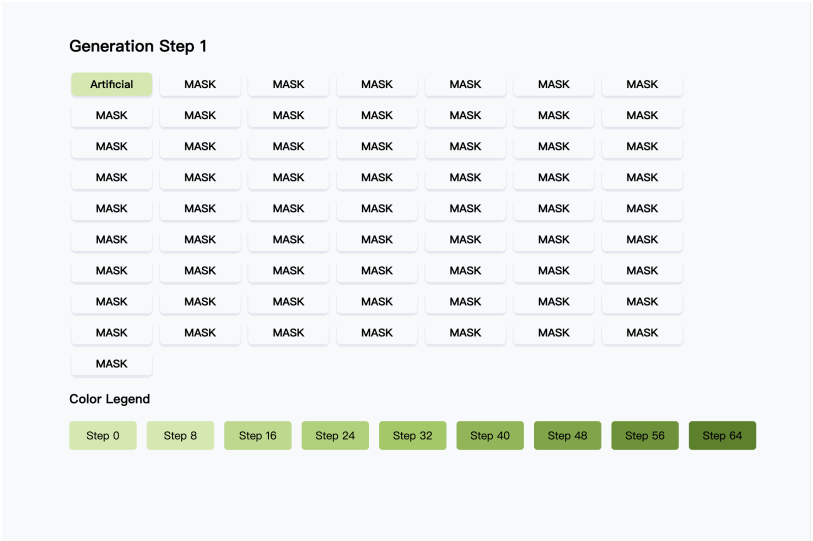
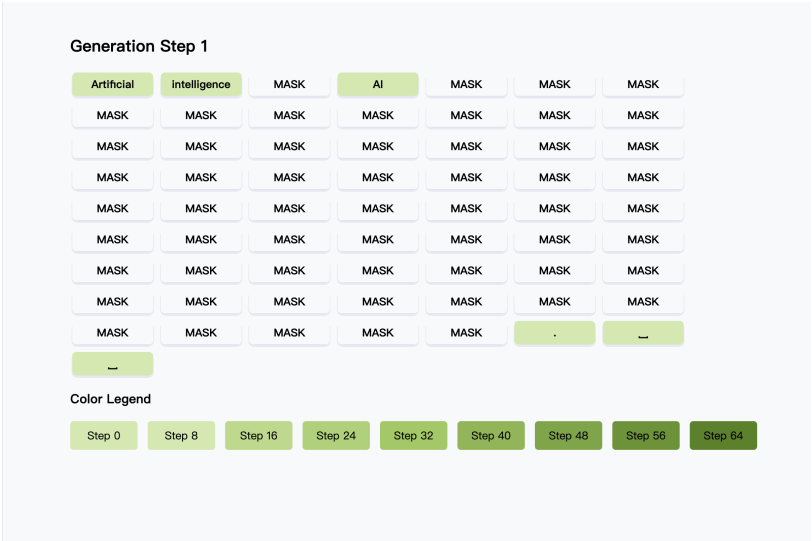


Figure 4 | **Impact of Cache Block Size on Accuracy and Throughput.** The orange line illustrates the effect of varying cache block size on throughput, while the blue line depicts the corresponding impact on accuracy.

Parallel sample visualization

- Visualize parallel sample process showing the **token position** and **numbers of tokens** each step produced
- Left is llada without parallel sampling and right is for parallel samping





Qualitative Results

- Our acceleration well maintains the previous LLaDA performance

Table 5 | Qualitative comparison of responses across methods.

Prompt: A robe takes 2 bolts of blue fiber and half that much white fiber. How many bolts in total does it take?		
Original	**PrefixCache**	**DualCache**
The robe takes 2 bolts of blue fiber. It also takes half that much white fiber, so it takes 2/2 = 1 bolt of white fiber. In total, the robe takes 2 + 1 = 3 bolts of fiber. So, the value is 3	The robe takes 2 bolts of blue fiber. It also takes half that much white fiber, so it takes 2/2 = 1 bolt of white fiber. In total, the robe takes 2 + 1 = 3 bolts of fiber. So, the value is 3	The robe takes 2 bolts of blue fiber. It also takes half that much white fiber, so it takes 2/2 = 1 bolt of white fiber. In total, it takes 2 bolts + 1 bolt = 3 bolts of fiber. The final result is 3

Table 6 | Qualitative comparison of responses with varying block size for DualCache.

Prompt: A robe takes 2 bolts of blue fiber and half that much white fiber. How many bolts in total does it take?		
Block Size 8	**Block Size 16**	**Block Size 32**
The robe takes 2 bolts of blue fiber. It also takes half that much white fiber, so it takes 2/2 = 1 bolt of white fiber. In total, the robe takes 2 + 1 = 3 bolts of fiber. So, the value is 3	The robe takes 2 bolts of blue fiber. It also takes half that much white fiber, so it takes 2/2 = 1 bolt of white fiber. In total, the robe takes 2 + 1 = 3 bolts of fiber. So, the value is 3	The robe takes 2 bolts of blue fiber. It also takes half that much white fiber, so it takes 2/2 = 1 bolt of white fiber. In total, the robe takes 2 + 1 = 3 bolts of fiber. So, the value is 3

Table 7 | Qualitative comparison of responses under different threshold settings.

Prompt: A robe takes 2 bolts of blue fiber and half that much white fiber. How many bolts in total does it take?		
Threshold 0.7	**Threshold 0.8**	**Threshold 0.9**
The robe takes 2 bolts of blue fiber. It also takes half that much white fiber, so it takes 2/2 = 1 bolt of white fiber. In total, it takes takes 2 + 1 = 3 bolts of fiber. So, the value is 3 **(NFE: 9)**	The robe takes 2 bolts of blue fiber. It also takes half that much white fiber, so it takes 2/2 = 1 bolt of white fiber. In total, the robe takes 2 + 1 = 3 bolts of fiber. So, the value is 3 **(NFE: 12)**	The robe takes 2 bolts of blue fiber. It also takes half that much white fiber, so it takes 2/2 = 1 bolt of white fiber. In total, the robe takes 2 + 1 = 3 bolts of fiber. So, the value is 3 **(NFE: 20)**

Quantitative Results

- Our acceleration well maintains the previous LLaDA performance

Table 1 | Comprehensive benchmark results on the LLaDA-Instruct suite. Each cell presents the accuracy and the decoding throughput in tokens per second with relative speedup to the LLaDA baseline (bottom row, blue: tokens per second/orange: relative speedup). The highest throughput and speedup for each configuration are highlighted.

Benchmark	Gen Length	LLaDA	+Cache	+Parallel	+Cache+Parallel (Fast-dLLM)
GSM8K (5-shot)	256	79.3	79.5	79.2	78.5
		6.7 (1×)	21.2 (3.2×)	16.5 (2.5×)	54.4 (8.1×)
	512	77.5	77.0	77.6	77.2
		3.2 (1×)	10.4 (3.3×)	18.6 (5.8×)	35.3 (11.0×)
MATH (4-shot)	256	33.5	33.3	33.4	33.2
		9.1 (1×)	23.7 (2.6×)	24.8 (2.7×)	51.7 (5.7×)
	512	37.2	36.2	36.8	36.0
		8.0 (1×)	19.7 (2.5×)	23.8 (3.0×)	47.1 (5.9×)
HumanEval (0-shot)	256	41.5	42.7	43.9	43.3
		30.5 (1×)	40.7 (1.3×)	101.5 (3.3×)	114.1 (3.7×)
	512	43.9	45.7	43.3	44.5
		18.4 (1×)	29.3 (1.6×)	57.1 (3.1×)	73.7 (4.0×)
MBPP (3-shot)	256	29.4	29.6	28.4	28.2
		6.0 (1×)	17.0 (2.8×)	24.8 (4.1×)	44.8 (7.5×)
	512	14.8	13.4	15.0	13.8
		4.3 (1×)	10.1 (2.3×)	22.3 (5.1×)	39.5 (9.2×)

Quantitative Results

- Our acceleration well maintains the previous Dream performance

Table 2 | Comprehensive benchmark results on Dream-Base variants over four tasks with different generation lengths (256 and 512). Each cell shows accuracy (top row) and decoding throughput in tokens per second with relative speedup to Dream-Base baseline (bottom row, blue: tokens per second/orange: relative speedup). Numbers in yellow indicate the highest throughput and speedup per configuration.

Benchmark	Gen Length	Dream	+Cache	+Parallel	+Cache+Parallel (Fast-dLLM)
GSM8K (5-shot)	256	75.0	74.3	74.2	74.8
		9.1 (1×)	32.5 (3.6×)	14.2 (1.6×)	48.2 (5.3×)
	512	76.0	74.3	73.4	74.0
		7.7 (1×)	25.6 (3.3×)	14.6 (1.9×)	42.9 (5.6×)
MATH (4-shot)	256	38.4	36.8	37.9	37.6
		11.4 (1×)	34.3 (3.0×)	27.3 (2.4×)	66.8 (5.9×)
	512	39.8	38.0	39.5	39.3
		9.6 (1×)	26.8 (2.8×)	31.6 (3.2×)	63.3 (6.5×)
HumanEval (0-shot)	256	49.4	53.7	49.4	54.3
		23.3 (1×)	35.2 (1.5×)	45.6 (2.0×)	62.0 (2.8×)
	512	54.3	54.9	51.8	54.3
		16.3 (1×)	27.8 (1.7×)	29.8 (1.8×)	52.8 (3.2×)
MBPP (3-shot)	256	56.6	53.2	53.8	56.4
		11.2 (1×)	34.5 (3.1×)	31.8 (2.8×)	76.0 (6.8×)
	512	55.6	53.8	55.4	55.2
		9.4 (1×)	26.7 (2.8×)	37.6 (4.0×)	73.6 (7.8×)

Ablations

- Longer prefill and gen. length will raise the acceleration speedup.
- DualCache gains more acceleration under longer gen. Length setting

Table 3 | **Performance and Speedup Comparison on LLaDA Between 5-Shot and 8-Shot Settings at Generation Length 1024.** This table compares the accuracy and throughput speedups of different decoding strategies under 5-shot and 8-shot configurations using a generation length of 1024. The results demonstrate how increased prefill length enhances the effectiveness of caching strategies, particularly for DualCache.

Setting.	LLaDA	Parallel Decoding		
		No Cache	PrefixCache	DualCache
5-shot	77.0 1.1 (1×)	77.4 11.7 (10.6×)	75.2 14.4 (13.1×)	74.7 21.6 (19.6×)
8-shot	77.3 0.7 (1×)	78.0 9.3 (13.3×)	75.7 13.0 (18.6×)	76.0 19.3 (27.6×)

Table 4 | **Impact of Generation Length on Accuracy and Speedup Under 8-Shot for LLaDA.** This table illustrates the effect of varying generation lengths (256, 512, and 1024) on decoding performance and efficiency for different caching strategies under the 8-shot setting. Longer generation lengths lead to higher throughput gains, especially for DualCache, validating the scalability of our approach.

Len.	LLaDA	Parallel Decoding		
		No Cache	PrefixCache	DualCache
256	77.6 4.9 (1×)	77.9 16.4 (3.3×)	77.3 49.2 (10.0×)	76.9 46.3 (9.4×)
512	78.9 2.3 (1×)	78.9 14.0 (6.1×)	74.8 32.0 (13.9×)	75.4 36.4 (15.8×)
1024	77.3 0.7 (1×)	78.0 9.3 (13.3×)	75.7 13.0 (18.6×)	76.0 19.3 (27.6×)

Thanks for listening!