Edit-Based Refinement for Parallel Masked Diffusion Language Models

Houxing Ren Mingjie Zhan Zimu Lu Ke Wang Yunqiao Yang

Haotian Hou Junting Pan Hongsheng Li

CUHK MMLab SenseTime Research

Shenzhen Loop Area Institute CPII under InnoHK

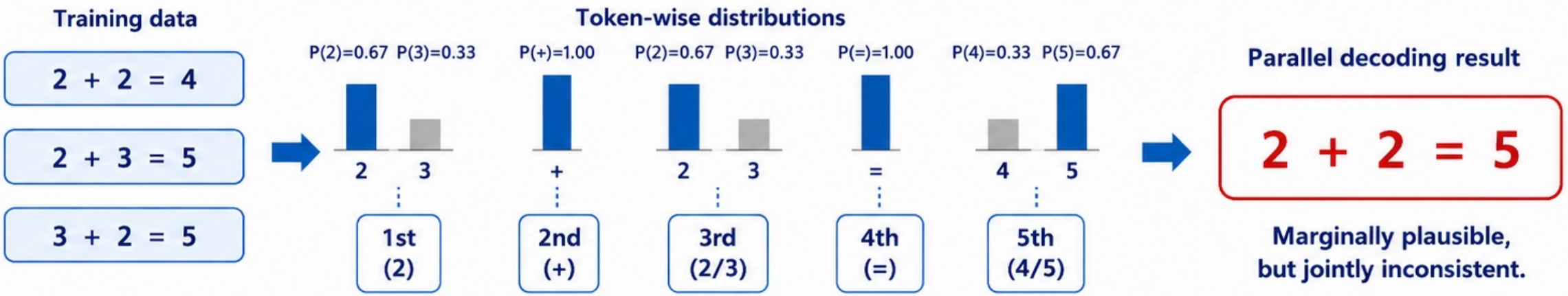
Background

Masked Diffusion Language Models (MDLMs)

- Generate text by progressively unmasking tokens rather than left-to-right decoding
- Enable parallel prediction across positions
- Offer attractive decoding efficiency and scalability

Main Limitation

- Training optimizes token-wise denoising / marginal prediction
- Parallel multi-token decoding predicts several positions simultaneously
- Locally likely tokens may form globally inconsistent sequences



Key challenge:
Bridge the gap between token-level training and sequence-level consistency.

Methods

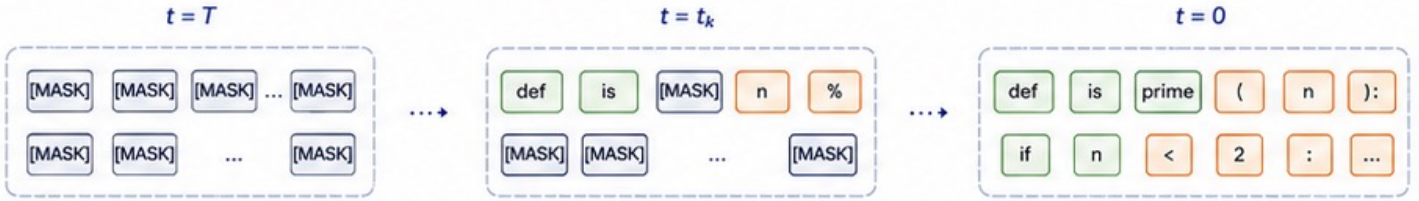
ME-DLM adopts a two-stage framework:
(1) Masked diffusion generates a coarse sequence in parallel; (2) Edit diffusion performs lightweight refinement via minimal edit operations to improve global consistency.

1 Stage 1: Masked Diffusion (Coarse Generation)

Input (Prompt)

Q: Write a function to check whether a number is prime.

Reverse Diffusion Process (T steps)



Coarse Output
(may be inconsistent)

```
def is_prime(n):
    if n < 2:
        return False
    if n % 2 == 0:
        return False
    return True
    ...
```

e.g., missing final loop

- Starting from a fully masked sequence, the model progressively denoises all positions in parallel.
- Fast generation with high efficiency.
- However, the coarse output may contain structural or logical inconsistencies.

2 Stage 2: Edit-Based Diffusion (Refinement)

Coarse Output

```
def is_prime(n):
    if n < 2:
        return False
    if n % 2 == 0:
        return False
    return True
    ...
```

Edit Operations (Token-wise)

For each position i, predict (c_i, n_i)

Replace / Keep	If $c_i \neq [DEL]$, replace x_i with c_i; if $c_i = x_i$, keep unchanged.	x_i = old → c_i = new
Delete	If $c_i = [DEL]$, remove the current token x_i.	x_i = token → $[DEL]$
Insert	Predict n_i after position i; if $n_i \neq c_{i+1}$, insert n_i between c_i and c_{i+1}.	c_i + n_i + c_{i+1}

Edits are predicted in parallel, conditioned on the full sequence.

Refined Output

```
def is_prime(n):
    if n < 2:
        return False
    if n % 2 == 0:
        return False
    for i in range(3, int(n**0.5)+1):
        if n % i == 0:
            return False
    return True
```

- Predict minimal edit operations (replace / delete / insert) to transform the coarse sequence.
- Conditions on the full sequence to model global dependencies.
- Produces a refined sequence with improved consistency and correctness.

Experiments

ME-DLM consistently improves performance, and the gains become larger under more aggressive parallel decoding.

Average Performance Across 4 Benchmarks

Budget	Stage-2	Stage-3	Gain
1/1	55.7	60.0	+4.3
1/2	50.7	55.4	+4.7
1/4	37.7	46.4	+8.7
1/8	19.3	32.6	+13.3

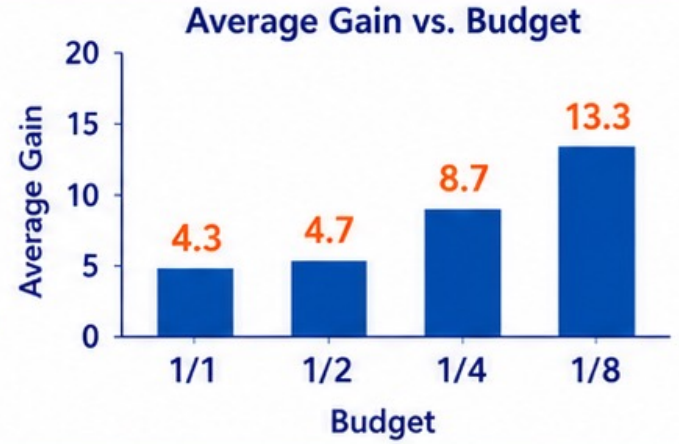
Average over HumanEval, HumanEval+, MBPP, MBPP+, GSM8K, and MATH-500. Lower budget = more aggressive parallel decoding.

Representative Improvements at Budget 1/8

HumanEval	MBPP	GSM8K	MATH-500
+11.6	+4.7	+33.6	+16.6

Key Findings

- Stage-3 outperforms Stage-2 at every budget.
- Gains are strongest in low-step, high-parallelism settings.
- Improvements hold across both code and math tasks.



Takeaway: | Edit-based refinement is most beneficial when parallel decoding is aggressive.

Experiments

Example: code generation under aggressive parallel decoding. A few localized edits turn a flawed draft into a correct program.

Mask Diffusion (Coarse Output)

Prompt: Write a Python function to recursively sum a nested list.

```
def recursive_list_sum(lst):
                                    ← missing total = 0
    for item in lst                 ← missing ':'
        if isinstance(item, list):
            total += recursive_list_sum(item)
        else:
            total += item
    return total
```

Non-executable / structurally inconsistent

Localized Edit Actions

1. **Insert** `total = 0` after the function header.
2. **Replace** `for item in lst` with `for item in lst:`
3. **Preserve** the remaining structure and tokens.

Edits are conditioned on the full sequence and applied with minimal changes.

ME-DLM (Refined Output)

```
def recursive_list_sum(lst):
    total = 0
    for item in lst:
        if isinstance(item, list):
            total += recursive_list_sum(item)
        else:
            total += item
    return total
```

Executable and logically consistent

Takeaway: ME-DLM preserves most tokens and fixes only the critical local errors, restoring global program correctness.

Conclusion

ME-DLM improves sequence-level consistency for parallel MDLMs while preserving the efficiency of parallel diffusion decoding.

Conclusion

1. Parallel MDLMs suffer from a mismatch between token-wise training and joint sequence consistency.
2. ME-DLM addresses this with a two-stage pipeline: coarse mask diffusion plus edit-based refinement.
3. Minimal replace / delete / insert operations, conditioned on the full sequence, correct local inconsistencies.
4. ME-DLM achieves consistent improvements on code and math benchmarks, especially under aggressive parallel decoding.

Future Work

- Explore training-free or lighter-weight mechanisms for enforcing sequence-level constraints.
- Study adaptive edit scheduling and stopping criteria.
- Extend evaluation to broader tasks, longer outputs, and more general instruction-following settings.
- Explore flexible-length refinement and richer edit strategies for more powerful generation.

Thank You!