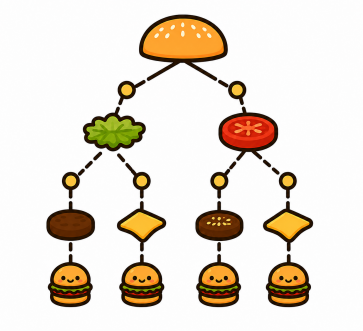
Can I Have Your Order?

Monte-Carlo Tree Search for Slot Filling Ordering in Diffusion Language Models

McDiffuSE

Joshua Ong Jun Leang, Yu Zhao, Mihaela Cătălina Stoian, Wenda Li, Shay B. Cohen, Eleonora Giunchiglia

Imperial College London | **University of Edinburgh**



Core idea: planning the slot-filling order

THESIS

Diffusion language models need a planner for *which slot to fill next*.

McDiffuSE treats slot choice as sequential decision making and uses MCTS to look ahead before committing to an infilling order.

3.2%
over AR baselines
Average gain across six benchmarks.

8.0%
over plan-and-infill
Average gain over ReFusion.

19.45%
MBPP improvement
Largest gain on structured code.

Key point. This is a training-free inference-time search procedure; it changes the slot-ordering policy, not the base model.

Contribution 1: slot ordering as search

CONTRIBUTIONS

The paper turns slot ordering into a deterministic MDP and solves it with MCTS.

Method contribution

- Formulate slot selection in masked diffusion decoding as a state-action problem.
- Adapt neural-guided MCTS using model confidence as the action prior.
- Use stochastic rollouts to estimate long-term trajectory coherence before commitment.

Contributions 2–3: behavior and results

CONTRIBUTIONS

Behavioral finding

- Mostly sequential ordering works, but rare non-sequential deviations matter.
- Exploration breadth is more important than simply increasing simulations.
- Coding tasks benefit strongly because code has hard structural dependencies.

Empirical result

McDiffuSE outperforms all evaluated diffusion baselines and matches or exceeds autoregressive baselines on five of six benchmarks under the reported setting.

BACKGROUND

Why slot ordering matters

Background needed to motivate the method.

Autoregressive decoding has a fixed order

DECODING PARADIGMS

AR models condition on a single left-to-right trajectory.

- The next token is always conditioned on every previous token.
- This makes local dependency handling straightforward.
- But it imposes a rigid generation order and can become long and inefficient for reasoning traces.

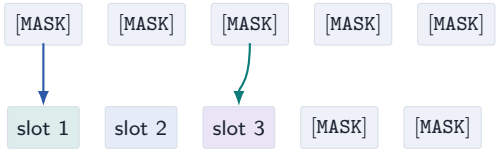
One legal path: left to right

Masked diffusion models relax the generation order

MDMS

MDMs can generate through iterative denoising rather than a fixed left-to-right path.

- The output begins as masked slots or masked tokens.
- Each iteration decides what to reveal and then denoises that part.
- This opens a larger space of possible generation schedules.



Many legal schedules

Plan-and-infill makes the schedule explicit

EXISTING FRAMEWORK

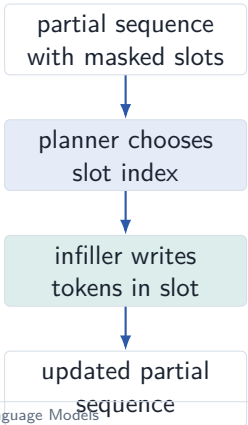
Each denoising iteration has a planner and an infiller.

Planning phase

Select one currently masked sub-sequence or slot to fill next.

Infilling phase

Generate the tokens inside that selected slot autoregressively.



Inter-slot dependencies are the bottleneck

FAILURE MODE

Generating a locally confident slot can make later slots harder.

Why confidence-only planning fails

- Slot confidence is local.
- Later slots may depend on earlier structural commitments.
- A wrong early slot can create unmodelled dependencies and propagate errors.

Why search helps

- Search can simulate consequences before commitment.
- Low-prior choices can be evaluated if they lead to globally coherent completions.
- Final action is chosen by accumulated evidence, not a one-step confidence score.

Slot ordering is a combinatorial problem

PLANNING CHALLENGE

For K slots, a complete ordering is a permutation.

$$\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_K)$$
$$\sigma_t \in \{1, \ldots, K\}$$
$$\sigma_i \neq \sigma_j \quad \text{for } i \neq j$$

The planner chooses the next slot index at every step.

Combinatorial pressure

- A greedy rule sees one step.
- Exhaustive search is impossible for realistic K.
- MCTS gives targeted lookahead under a compute budget.

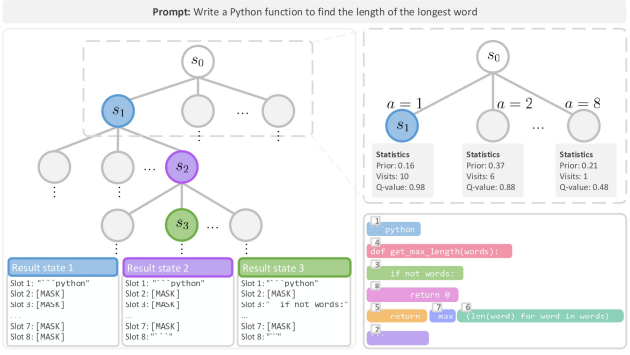
METHOD

Slot selection as decision making

Slot ordering as a Markov decision process solved with MCTS.

Method overview: MCTS inside every denoising step

MCDIFFUSE



The search tree evaluates possible slot choices before committing to the next infill.

McDiffuSE changes the planner, not the base model

TRAINING-FREE

McDiffuSE is an inference-time wrapper around plan-and-infill decoding.

Kept from ReFusion

- Slot-based plan-and-infill structure.
- Autoregressive generation inside each slot.
- Model confidence scores for generated tokens.

Changed by McDiffuSE

- Planner becomes an MCTS controller.
- Confidence is used as a prior and reward signal.
- Rollouts estimate future coherence before choosing a slot.

Objects in the decision problem

PROBLEM SETUP

The generated response is split into K contiguous, disjoint slots.

$$\mathbf{x} = (\mathbf{x}^1, \ldots, \mathbf{x}^K)$$
$$\mathbf{x}^k = \text{slot } k \text{ with } L \text{ tokens}$$
$$\sigma = \text{slot ordering}$$

Generation rule

- Tokens within a slot are generated autoregressively.
- Slots may be generated in arbitrary order.
- The planner searches over the order, not the token content directly.

State: ordering prefix and partial sequence

MDP FORMULATION

A state stores both the ordering prefix and the partial sequence.

$$
\begin{aligned}
s_t &= (\sigma_{:t}, \mathbf{x}_t) \\
\sigma_{:t} &= (\sigma_1, \ldots, \sigma_t) \\
\mathbf{x}_t &= (\mathbf{x}_t^1, \ldots, \mathbf{x}_t^K)
\end{aligned}
$$

Interpretation

If slot k has not yet appeared in $\sigma_{:t}$, then $\mathbf{x}_t^k$ is still $[\texttt{MASK}]$.

Initial state: all slots masked

STATE EXAMPLE

The root represents no committed slot decisions.

$$\sigma_{:0} = \emptyset$$

$$\mathbf{x}_0 = [\texttt{MASK}]^K$$

[MASK]	[MASK]	[MASK]	[MASK]	[MASK]	[MASK]
slot 1	slot 2	slot 3	slot 4	slot 5	slot 6

At the first decision point, every slot is admissible.

Action: choose one unfilled slot

ACTION SPACE

At step t, the action set shrinks to the remaining slots.

$$\mathcal{A}_t = \{k \mid k \in \{1, \ldots, K\} \setminus \{\sigma_1, \ldots, \sigma_t\}\}$$

Action meaning

$a = \sigma_{t+1}$ means "fill slot a next."

Important consequence

The action is structural: it changes the context available to all later slot predictions.

Transition: deterministic given the chosen slot

TRANSITION FUNCTION

Given the selected slot, the successor state is uniquely determined by argmax infilling.

$$
\begin{aligned}
s_{t+1} &= (\sigma_{:t} \oplus \sigma_{t+1}, \mathbf{x}_{t+1}) \\
\mathbf{x}_{t+1}^{\sigma_{t+1}} &= \text{model prediction for selected slot} \\
\mathbf{x}_{t+1}^{k} &= \mathbf{x}_{t}^{k} \quad \text{for } k \neq \sigma_{t+1}
\end{aligned}
$$

Why deterministic?

The paper uses argmax decoding within the slot for the transition, so search evaluates ordering decisions rather than sampling token alternatives.

Reward: slot-level confidence

REWARD FUNCTION

The reward is the mean probability assigned to the generated slot tokens.

$$\mathcal{R}(s_t, a) = \frac{1}{L} \sum_{i=1}^{L} \mathbb{P}_\theta \left(\mathbf{x}_{t+1}^{\sigma_{t+1}}[i] \mid \mathbf{x}_t, \mathbf{x}_{t+1}^{\sigma_{t+1}}[< i], \text{Prompt} \right)$$

Interpretation

It is a confidence proxy for the quality of the slot filled after taking action a.

Confidence as a prior, not a greedy objective

REWARD DESIGN

The method avoids turning confidence into a purely greedy rule.

Risk

High confidence can favor locally easy continuations that constrain later slots badly.

Mitigation

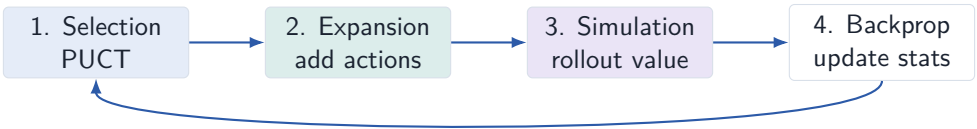
Confidence enters as a prior and as part of the value estimate; PUCT and rollouts decide whether it should be trusted.

This design is the main difference between a confidence planner and a search planner.

MCTS cycle adapted to slot planning

SEARCH LOOP

Every simulation runs selection, expansion, simulation, and backpropagation.



After N_{sim} simulations, choose the root child with the highest visit count.

Selection: PUCT with a confidence prior

SELECTION

PUCT combines empirical value with a confidence-derived policy prior.

$$\mathrm{PUCT}(s,a) = \underbrace{Q(s,a)}_{\text{exploitation}} + \underbrace{c \cdot P(a \mid s) \cdot \frac{\sqrt{N_s}}{1+N_s^a}}_{\text{prior-guided exploration}}$$

$Q(s,a)$

Mean return from simulations that selected action a.

$P(a \mid s)$

Prior from normalized slot-level confidence.

c

Exploration constant; crucial in the analysis.

Expansion: confidence scores become priors

EXPANSION

Every admissible unfilled slot becomes a child of the current state.

$$P(a \mid s_t) = \frac{\mathcal{R}(s_t, a)}{\sum_{a' \in \mathcal{A}_t} \mathcal{R}(s_t, a')}$$

What gets initialized

For each new state-action pair: visits N_s, edge visits N_s^a, and accumulated value $W(s, a)$ begin at zero.

Why this matters

The model's own belief guides early exploration, but search can override it when lookahead finds better trajectories.

Simulation estimates future coherence

ROLLOUT

A leaf is evaluated by mixing immediate confidence with a stochastic future rollout.

$$V(s_t, a) = \lambda \cdot \mathcal{R}(s_t, a) + (1 - \lambda) \cdot \mathbb{E}_{\pi_{\text{roll}}}[G]$$

Immediate term

How confident is the model in the selected slot now?

Rollout term

If we keep filling remaining slots, does this choice lead to a coherent trajectory?

Rollout policy: confidence-weighted sampling

ROLLOUT POLICY

Remaining slots are sampled from a temperature-scaled confidence distribution.

$$\pi_{\text{roll}}(a \mid s_r) = \frac{\exp(\mathcal{R}(s_r, a)/\tau)}{\sum_{a' \in \mathcal{A}_r} \exp(\mathcal{R}(s_r, a')/\tau)}$$

Why stochastic?

It samples plausible futures without collapsing to the single most confident next slot at every rollout step.

Role of τ

Lower τ sharpens confidence preference; higher τ increases diversity.

Rollout return: mean reward over the trajectory

ROLLOUT RETURN

The rollout score summarizes the quality of the remaining completion path.

$$G = \frac{1}{T} \sum_{i=1}^{T} \mathcal{R}(s_i, \tilde{a}_i), \qquad \tilde{a}_i \sim \pi_{\text{roll}}(\cdot \mid s_i)$$

What G captures

Whether the remaining slots stay probable after the current structural commitment.

What G does not need

Ground-truth references at inference time.

Value mixing: $\lambda = 0.3$ favors the rollout

VALUE MIXING

The chosen value function intentionally prioritizes long-term rollout feedback.

$\lambda = 0.3$

Immediate slot confidence remains useful, but gets less weight than future trajectory quality.

$1 - \lambda = 0.7$

Most of the value comes from whether the subsequent completion remains coherent.

The appendix grid search finds best MATH500 accuracy at $\tau = 0.5$, $\lambda = 0.3$.

Backpropagation: updating edge statistics

BACKPROPAGATION

Every visited edge gets a stronger empirical estimate after each simulation.

$$
\begin{aligned}
N_s &\leftarrow N_s + 1 \\
N_s^a &\leftarrow N_s^a + 1 \\
W(s,a) &\leftarrow W(s,a) + V(s,a) \\
Q(s,a) &= \frac{W(s,a)}{N_s^a}
\end{aligned}
$$

Effect

If a low-prior action repeatedly leads to high rollout values, its Q and visit counts can overcome the initial confidence bias.

Decision: robust (most-visited) child selection

DECISION RULE

After search, McDiffuSE chooses the most visited root child.

$$a_t^* = \arg\max_{a \in \mathcal{A}_t} N_{s_t}^a$$

During search

PUCT includes an exploration bonus.

At decision time

The exploration bonus is removed; the selected slot is the action most supported by simulations.

The full algorithm per decoding step

ALGORITHM

At every decoding step, run search from the current partial sequence.

1. Initialize root state $s_t = (\sigma_{:t}, \mathbf{x}_t)$.
2. Compute slot-level confidences for each admissible action.
3. Normalize confidence into priors $P(a \mid s_t)$.
4. Run N_{sim} MCTS simulations using PUCT.
5. Select a_t^* by maximum root visit count.
6. Fill slot a_t^*, update state, and repeat until no masks remain.

Worked example: confidence prefers the wrong first slot

RUNNING EXAMPLE

The greedy prior wants the function definition; search prefers the code-fence structure.

Prompt

Write a Python function to find the length of the longest word.

Greedy prior

$P(a = 2 \mid s_0) = 0.37$ favors slot 2: `def get_max_length(words):`

Search value

$Q(s_0, a = 1) = 0.98$ for slot 1 vs. $Q(s_0, a = 2) = 0.88$ for slot 2.

Decision

Start with slot 1: `"‘python`. This sets structure and improves downstream coherence.

Some slots set structure that later slots depend on

STRUCTURAL DEPENDENCY

Some slots set the grammar that later slots rely on.

- In Python generation, headers, indentation, imports, and closing delimiters are not independent.
- A locally plausible function header can be less valuable if the surrounding structure is missing.
- Search allows a lower-prior structural slot to be selected first when it improves future slots.

Method intuition

McDiffuSE is useful when the right first slot is the one that makes the rest of the output easier, not necessarily the one the model is most confident about now.

Roles of the prior, reward, and rollout

METHOD INTUITION

Confidence becomes a search guide, not an oracle.

Prior

"Where should search look first?"

Reward

"Was this slot likely under the model?"

Rollout

"Does the rest of the completion stay coherent?"

One-line answer

The method does not trust local confidence blindly; it asks whether local confidence survives the future.

Hyperparameters used in the main setup

IMPLEMENTATION

The method has four search-specific knobs.

Hyperparameter	Main role	Paper setting
λ	immediate vs. future value	0.3
c	exploration strength	50 in main text analysis
N_{sim}	simulation budget	256 in main text analysis
τ	rollout sampling temperature	0.5

Appendix implementation details also report a practical configuration with 30 simulations, slot size 4, 32 serial blocks, slot threshold 0.5, token threshold 0.6, and $c = 10.0$.

Compute cost scales with the simulation budget

COST MODEL

N_{sim} costs forward passes; c mostly changes the selection policy.

Increasing simulations

- More rollout evaluations.
- Roughly linear time increase.
- Can over-concentrate if exploration is too low.

Increasing exploration constant

- Pushes search toward lower-prior branches.
- Negligible direct overhead.
- Crucial for escaping local confidence bias.

EXPERIMENTS

Does better slot planning improve reasoning?

Six benchmarks, diffusion and autoregressive baselines, matched prompting.

Benchmarks span math, science QA, and code

BENCHMARKS

The benchmark suite tests several dependency structures.

Math reasoning	General/science QA	Code generation
GSM8K MATH500	ARC Challenge GPQA-Diamond	MBPP HumanEval

Metric

Pass@1 accuracy, averaged over three independent runs with reported standard errors.

Baselines isolate the effect of the planner

BASELINES

McDiffuSE is compared against AR models, MDMs, and ReFusion planner variants.

Autoregressive	Diffusion	Planner variants
Llama-3.1 8B	LLaDA 8B	ReFusion + Sequential
Qwen2.5 7B	Dream 7B	ReFusion + Random
Qwen3 8B	ReFusion 7B	McDiffuSE

Main result: McDiffuSE has the best average accuracy

AGGREGATE

Search-based slot planning reaches 68.52% average accuracy.

Model family / model	Average accuracy	Readout
Qwen3 8B	65.33%	best AR baseline average
ReFusion 7B	60.52%	plan-and-infill baseline
ReFusion + Sequential	58.29%	fixed left-to-right slot order
ReFusion + Random	43.10%	random slot order collapses
McDiffuSE	**68.52%**	best overall average

McDiffuSE improves by roughly 8.0 points over ReFusion and 3.2 points over the best AR average.

Per-benchmark results: McDiffuSE leads on five of six

PER-BENCHMARK

Model	GSM8K	MATH500	ARC	GPQA	MBPP	HumanEval
Qwen3 8B	**88.21**	46.20	88.40	23.13	69.81	76.24
ReFusion 7B	85.64	42.90	87.98	30.45	54.12	62.05
ReFusion + Seq.	77.56	38.90	87.09	33.43	50.58	62.18
ReFusion + Rand.	66.87	29.10	80.98	27.63	21.40	32.60
McDiffuSE	87.91	**47.80**	**88.68**	**34.78**	**73.57**	**78.37**

The only benchmark where McDiffuSE is not the top score is GSM8K, where it remains close to Qwen3.

Largest gains appear on code generation

TASK EFFECT

Structured program synthesis benefits most from strategic slot planning.

MBPP

+19.45%

Absolute gain reported over ReFusion.

HumanEval

+16.32%

Strong improvement on function-level code synthesis.

Why code benefits

- Variable declarations, headers, indentation, and control flow constrain later text.
- Bad early structure can make remaining slots syntactically impossible.
- MCTS explicitly tests future consequences.

Math and QA gains are smaller but consistent

REASONING

McDiffuSE also improves MATH500, ARC, and GPQA-Diamond.

Benchmark	ReFusion	McDiffuSE	Absolute gain
MATH500	42.90	47.80	+4.90
ARC Challenge	87.98	88.68	+0.70
GPQA-Diamond	30.45	34.78	+4.33

The paper interprets the code-heavy gains as evidence that MCTS planning is especially useful when slot dependencies are strict and structural.

Random slot ordering sharply degrades accuracy

PLANNER ABLATION

A bad slot planner can destroy performance even with the same base model.

ReFusion + Random

43.10%

Average accuracy across the six benchmarks.

ReFusion baseline

60.52%

The planner matters before we even add MCTS.

Takeaway

Slot ordering is not an implementation detail; it is a central inference decision.

McDiffuSE produces more compact reasoning

TOKEN EFFICIENCY

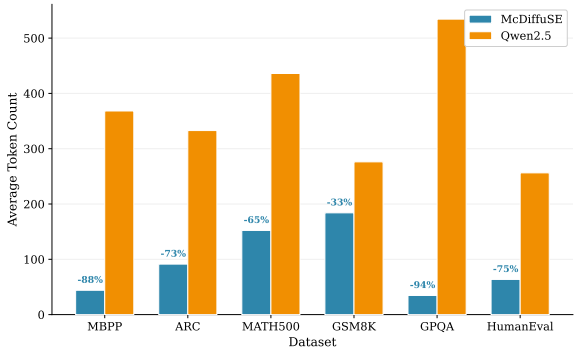
Compared with Qwen2.5-7B, McDiffuSE uses far fewer tokens while improving accuracy.

Dataset	Token reduction	Accuracy gain	Comment
MBPP	-88.05%	+8.39	strongest compact-code effect
ARC	-72.55%	+0.52	same accuracy regime, fewer tokens
MATH500	-65.08%	+2.40	shorter math traces
GSM8K	-33.39%	+2.20	arithmetic remains more sequential
GPQA	-93.53%	+17.84	compact answer path
HumanEval	-75.19%	+6.59	code efficiency

All reductions are reported as statistically significant with $p < 0.001$.

Token efficiency versus accuracy

EVIDENCE



McDiffuSE reduces generated token count substantially while keeping or improving accuracy.

Compute-matched search baselines fall short

COMPUTE CONTROL

Naively spending more compute is not enough.

Model	GSM8K	MATH500	ARC	GPQA	MBPP	HumanEval	Avg.
ReFusion	84.91	54.00	89.76	32.32	68.20	78.66	67.98
Best-of-N	79.76	45.60	88.65	30.81	57.20	68.90	61.82
Beam Search	81.65	49.40	89.42	31.82	59.40	71.34	63.84
DiffuSearch	**88.10**	**62.20**	**91.98**	**36.87**	**78.40**	**89.63**	**74.53**

The appendix names the improved method DiffuSearch in this compute-matched table; it refers to the same MCTS-style slot planning idea.

Comparison with long-context autoregressive models

EXTENDED CONTEXT

McDiffuSE remains competitive with far shorter maximum length.

Model	Context	GSM8K	MATH500	GPQA
McDiffuSE	1024	**90.63**	54.60	37.19
Llama3.1 7B	8096	84.91	39.40	37.71
Qwen2.5 7B	32768	89.99	63.40	35.94
Qwen3 8B	32768	88.17	**68.00**	**41.96**

The limitation is visible too: MATH500 still rewards the very long-context AR setup.

ANALYSIS

Why does the method work?

Mechanisms behind the improvements.

Most decisions remain sequential

SEQUENTIALITY

McDiffuSE does not abandon left-to-right structure.

Coding tasks

91.1%
of slot-ordering decisions follow a sequential pattern.

Math reasoning tasks

93.8%
of decisions are sequential.

The method mostly exploits the reliable sequential prior, then deviates when lookahead justifies it.

Rare non-sequential decisions matter disproportionately

KEY FINDING

Among cases where McDiffuSE succeeds and the sequential baseline fails, non-sequential choices are common.

Failure-recovery subset

13.2%

of the dataset: McDiffuSE succeeds while the sequential AR-style baseline fails.

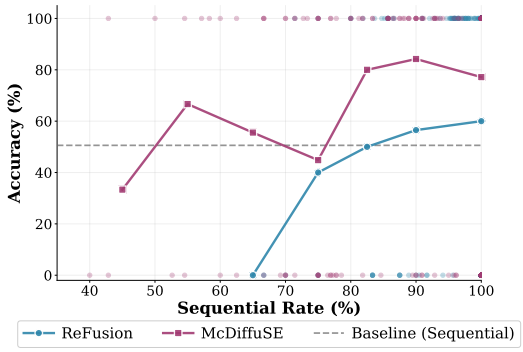
Non-sequential involvement

60.7%

of those recovered cases include at least one non-sequential decision.

Accuracy versus sequentiality on MBPP

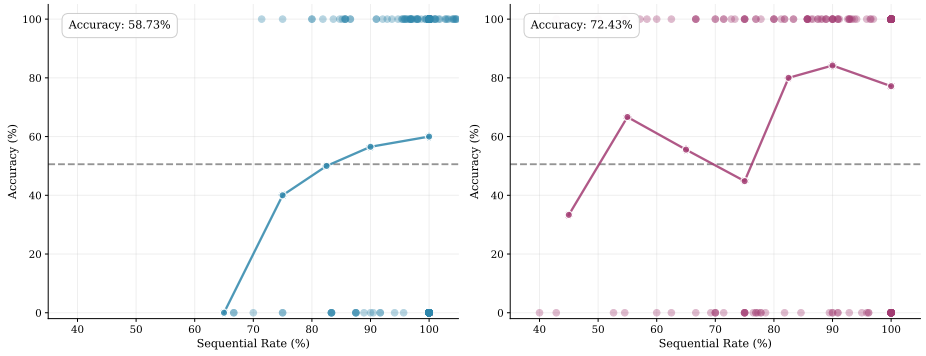
ANALYSIS FIGURE



Accuracy often peaks at high but not perfect sequentiality: mostly sequential with strategic deviations.

MBPP: accuracy peaks below full sequentiality

APPENDIX FIGURE

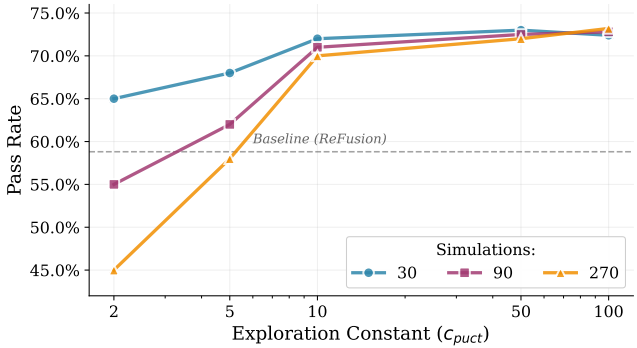


The appendix reports McDiffuSE around 72.43% on MBPP in this analysis and an approximately 8.2% non-sequential exploration rate.

Exploration matters more than simulation depth

SEARCH BEHAVIOR

Increasing c consistently helps; increasing N_{sim} alone can hurt.



Low exploration plus more simulations can reinforce the model's locally confident but globally myopic choices

Policy entropy explains premature convergence

POLICY ENTROPY

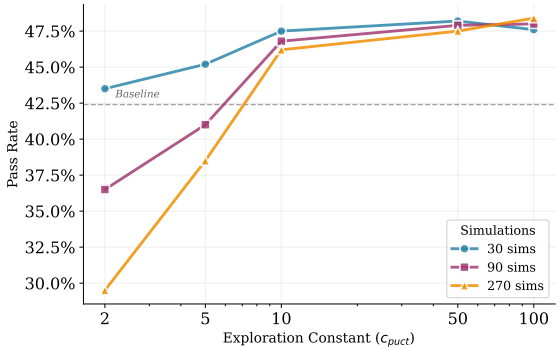
Config	Mean H (bits)	Std H	Median H	Concentration
$c = 2,\ N = 30$	1.4062	0.3785	1.4824	0.4958
$c = 2,\ N = 270$	1.1842	0.4043	1.1492	0.6118
$c = 100,\ N = 30$	1.4176	0.3771	1.5058	0.4954
$c = 100,\ N = 270$	1.4340	0.3768	1.5211	0.4879

Readout

With low exploration, more simulations lower entropy and increase concentration; with high exploration, entropy stays broad.

MATH500 shows the same exploration trend

APPENDIX ANALYSIS

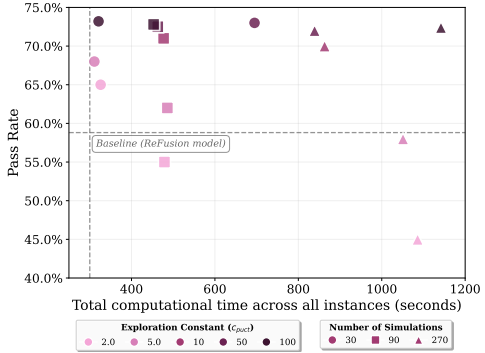


Higher exploration constants improve MATH500 performance, while increasing simulations has diminishing returns.

Generation time tracks simulation budget

EFFICIENCY

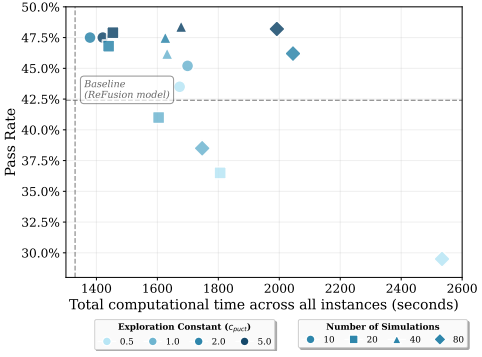
Cost rises mainly with N_{sim}, not with c.



For MBPP, high-exploration low-budget settings can capture large gains without the full cost of deep simulations

MATH500 shows the same cost pattern

EFFICIENCY

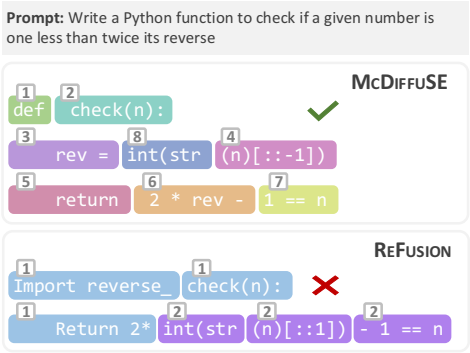


The paper reports a roughly linear relationship between simulations and time; changing exploration has negligible overhead.

Qualitative example: avoiding a local optimum

MBPP EXAMPLE

ReFusion commits to a high-confidence import; McDiffuSE starts with the function structure.



Lookahead catches that the import creates a syntactically broken trajectory for this prompt.

Compute overhead is moderate relative to gains

FLOPS

McDiffuSE uses about 35% more compute than ReFusion on average.

Model	GSM8K	MATH	ARC	GPQA	MBPP	HEval	Avg.
ReFusion	6.18E16	2.78E16	1.00E16	2.97E15	1.16E16	4.22E15	1.95E16
McDiffuSE	8.95E16	3.62E16	8.21E15	2.76E15	1.53E16	5.60E15	2.63E16

The largest compute increases appear on GSM8K, MATH500, MBPP, and HumanEval; ARC and GPQA are marginally lower in the reported table.

Hyperparameter grid: best at $\tau = 0.5$, $\lambda = 0.3$

TUNING

$\tau \backslash \lambda$	0.0	0.3	0.5	0.7	0.9	1.0
0.1	43.20	44.40	43.80	44.00	43.40	43.00
0.3	44.80	47.00	46.20	46.40	45.20	44.60
0.5	45.60	**48.20**	47.20	47.40	45.80	44.80
0.7	45.00	47.20	46.40	46.60	45.40	44.20
1.0	43.60	44.80	44.20	44.20	43.60	43.00

The best grid point balances stochastic rollout diversity with coherent slot ordering.

Mechanistic summary of the analysis

MECHANISM

The method works by selectively overriding local confidence.

Default	Exception	Search knob
Follow sequential structure most of the time.	Use non-sequential moves when they repair structural dependencies.	Use enough exploration to reach low-prior but high-value branches.

Positioning in related work

POSITIONING

McDiffuSE connects MCTS reasoning methods with text diffusion decoding.

MCTS for reasoning

Existing work uses MCTS for math, coding, and LLM reasoning, often in autoregressive settings.

This paper

First application, to the authors' knowledge, of MCTS to masked diffusion models for text generation slot ordering.

MCTS for diffusion

Prior MCTS-style work also appears in visual diffusion generation.

Core novelty

The search action is not a token; it is the ordering decision over slots.

Limitations

LIMITATIONS

The strongest story is not that MCTS is free; it is that the extra compute is useful.

- Inference overhead is real and mainly controlled by the simulation budget.
- The method depends on slot-based plan-and-infill decoding, so ReFusion is the main base model.
- Diffusion reasoning length remains constrained by current MDM designs.
- MATH500 long-context AR baselines still outperform McDiffuSE in the extended-context comparison.

Practical implications

TAKEAWAY

For MDMs, generation order is a controllable inference-time resource.

Before this framing	After this framing
Slot ordering can look like a heuristic detail of the decoder.	Slot ordering becomes a decision problem with priors, value estimates, exploration, and robustness criteria.

This opens a path to adaptive compute allocation in diffusion language generation.

Conclusion

FINAL MESSAGE

McDiffuSE makes masked diffusion decoding more coherent by planning the slot order.

- It formulates slot ordering as a deterministic MDP.
- It uses confidence-guided PUCT and stochastic rollouts to evaluate future consequences.
- It improves over diffusion baselines and is competitive with AR baselines.
- Its gains come from mostly sequential generation plus rare, strategically useful non-sequential choices.
- Its key search lesson is breadth: exploration can matter more than extra simulations.

CORE TAKEAWAY

McDiffuSE improves masked diffusion decoding by planning slot order.

It does not replace the model; it changes how the model decides what to fill next.

Thank you