PLAN-AND-PATCH: DIFFUSION LANGUAGE MODELS FOR AGENTIC PLANNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Planning is increasingly important for long-horizon agents, where successful execution requires coordinating subgoals, tool use, and intermediate outcomes over many steps. Yet assumptions made during planning may be invalidated by the environment, tools may return unexpected results, or actions may fail. Effective agents must therefore not only generate plans, but also revise them. Such revisions often affect only part of a plan, leaving the preceding and subsequent structure intact. Rather than regenerate the entire plan and risk unnecessary changes, repair can regenerate the affected region conditioned on the preserved prefix and suffix. We introduce PLAN-AND-PATCH, a plan-and-act framework in which a diffusion language model (dLLM) generates a structured, program-like plan through parallel unmasking and repairs it by filling in selected regions while keeping the surrounding steps fixed. We compare DreamReasoner-8B and Qwen3-8B as diffusion and autoregressive (AR) planners. On Natural Plan without task-specific training, diffusion (53.7%) achieves nearly twice the plan repair success rate of AR (27.0%). After task-specific training on agentic benchmarks, ALFWorld and TextCraft, the planners achieve similar observed success in plan generation, while diffusion reduces mean plan-generation latency by 39–46% relative to AR. Our results show that PLAN-AND-PATCH provides a framework for faster plan generation and effective plan repair in long-horizon agents.

1 INTRODUCTION

Planning helps long-horizon agents coordinate subgoals, tool use, and intermediate outcomes. Plan-and-act methods first construct a multi-step plan, then turn its steps into environment actions (Wang et al., 2023a; Sun et al., 2023; Erdogan et al., 2025). This separation can improve task success over workflows without an explicit planning stage (Prasad et al., 2024; Sun et al., 2023; Wu et al., 2026a; Peng et al., 2026; Zhang et al., 2025). Program-like representations make the plan's structure explicit through subgoals and executable steps (Singh et al., 2023a; Liang et al., 2023; Wang et al., 2024; 2023b; Zhang et al., 2025). Yet assumptions made during planning may be invalidated by the environment, tools may return unexpected results, or actions may fail. Such failures can arise because the planner has incomplete knowledge of the environment, misjudges action preconditions, or relies on conditions that change during execution (Wang et al., 2023b; Sun et al., 2023; Erdogan et al., 2025). Effective agents must therefore both generate plans and revise them in response to environment feedback.

Planning requires reasoning about dependencies among actions and their long-term consequences for achieving the goal (Bachmann & Nagarajan, 2024). LLM planners address this challenge by anticipating future states (Hao et al., 2023) and decomposing complex tasks into simpler subgoals (Prasad et al., 2024; Singh et al., 2023a). Plan format also affects the performance of an agent significantly. Zambrano et al. (2026) compare sequential subgoals, narrative, pseudocode, and checklists, finding that web-agent performance depends on both representation and planner–executor pairing. We propose a **structured plan representation** with explicit subgoals and executable steps to provide meaningful units for revision. A replacement must remain compatible with the preceding and subsequent plan context.

Such revisions often affect only part of a plan. However, existing plan-and-act agents commonly revise globally (Prasad et al., 2024; Erdogan et al., 2025; Shinn et al., 2023). In this work, we model

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

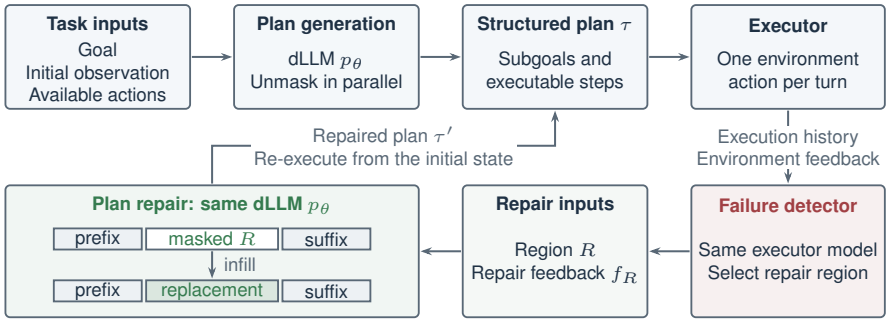


Figure 1: **The PLAN-AND-PATCH framework.** The same planner generates plans and infills selected regions. After failed execution, the executor acts as the failure detector, selecting a region using execution history and environment feedback. The planner infills it conditioned on repair feedback, with the prefix and suffix fixed. Figure A.2 gives a worked example.

plan repair as an infilling problem, identifying the affected region and regenerating it conditioned on the preserved prefix and suffix.

While autoregressive (AR) models can be adapted to infilling through explicit fill-in-the-middle training objectives (Bavarian et al., 2022; Fried et al., 2023), masked diffusion language models (dLLMs) naturally learn to reconstruct masked regions conditioned on their surrounding context (Austin et al., 2021; Lou et al., 2024; Sahoo et al., 2024). Recent work demonstrates competitive code generation (Singh et al., 2023b; Gong et al., 2025) and code infilling (Wu et al., 2026b; Inception Labs et al., 2025) with diffusion language models, including higher generation throughput at comparable coding performance (Inception Labs et al., 2025). These results motivate exploring diffusion models for structured plan generation and repair. We therefore ask,

Can we design an agentic planning framework that exploits diffusion language models' native capabilities for parallel generation and infilling?

To address this question, we introduce PLAN-AND-PATCH (Figure 1), a plan-and-act framework that uses a single dLLM trained for both plan generation and repair. The planner generates a complete structured plan through parallel unmasking, and a separate executor translates its steps into environment actions. If execution fails, the executor also serves as the failure detector, using the current plan, execution history, and environment feedback to select a repair region and explain the required revision. The planner uses this repair feedback to infill the region while preserving the surrounding plan text.

We evaluate diffusion and AR planners on ALFWorld for household task execution (Shridhar et al., 2021) and TextCraft for dependency-based crafting (Prasad et al., 2024), using task-specific training, a shared structured plan representation, and a common executor. The planners achieve similar observed performance in plan generation and repair, with a slight edge to AR, while diffusion reduces mean plan-generation latency by 39–46%. On Natural Plan for travel and scheduling (Zheng et al., 2024) without task-specific training, diffusion (53.7%) nearly doubles the plan repair success rate of AR (27.0%). Our main contributions are as follows.

1. We introduce PLAN-AND-PATCH, a plan-and-act framework that jointly trains a single diffusion language model for complete-plan generation and plan repair, modeling repair as infilling conditioned on surrounding plan context and repair feedback.
2. We develop a data construction pipeline that combines synthetic repair examples, generated through structured perturbations of verified plans, with naturally occurring execution failures. Together with successful trajectories, these examples support joint training for generation and repair.
3. We present a training recipe and curriculum for joint plan generation and repair with diffusion language models, using objective-specific block-size sampling and decoding. We describe the training choices that support joint generation and repair, and analyze how plan representation, executor choice, and repair-region selection affect performance.

2 Related Work

Diffusion language models and agents. Masked diffusion language models recover masked tokens from surrounding context, enabling flexible generation order and bidirectional infilling (Austin et al., 2021; Lou et al., 2024; Sahoo et al., 2024). Building on this flexibility, advances in scaling, post-training, and decoding have improved generation quality, reasoning, and inference efficiency (Nie et al., 2025; Ye et al., 2025b; Zhu et al., 2025; Zhao et al., 2025; Wang et al., 2025; Wu et al., 2025; Ma et al., 2026). Their practical efficiency depends on context length, batching, and the number of sampling steps (Kim et al., 2025). These models have also been applied to agents, both as world models that predict environment responses (Deshpande, 2026) and as ReAct-style agents that interleave action generation with environment observations (Yao et al., 2023; Zhen et al., 2026; Zhao et al., 2026; Bie et al., 2026). In these settings, Lu et al. (2026) identify two systematic failure modes, "retry loops" in embodied agents and "imprecise tool-call formats" in tool-calling agents. Agent-specific training improves reliability and efficiency (Zhao et al., 2026; Bie et al., 2026). Under matched fine-tuning, Zhen et al. (2026) also compare diffusion and AR models as agents. We study these models in a plan-and-act setting, focusing on complete-plan generation and plan repair through infilling guided by repair feedback.

Plan repair and program repair. Plan-and-act agents construct multi-step plans and translate their steps into environment actions (Wang et al., 2023a; Sun et al., 2023; Erdogan et al., 2025). When execution fails, these plans can be revised using environment feedback (Prasad et al., 2024; Song et al., 2023; Shinn et al., 2023; Madaan et al., 2023; Huang et al., 2023; Wang et al., 2023b; Liu et al., 2023; Wang et al., 2024). An important choice is which part of the plan to revise. PIVOT and ContextFlow preserve a validated prefix and rewrite a suffix (Zhang et al., 2026a; Guo et al., 2026). Atomic Task Graph preserves validated graph regions and repairs the affected subgraph (Zhang et al., 2026b), while L-ICL localizes the first constraint violation and supplies a corrective example (Kumar & Cohen, 2026). To assess repair, controlled replay helps distinguish the effect of changing a plan from variation across executions (Luan et al., 2026). Our evaluation holds failed plans and repair regions fixed across diffusion and AR models, and separately assesses region selection using execution history.

A similar use of feedback appears in program repair, where test failures guide fault localization and patch validation (Le Goues et al., 2019; 2012; Xia & Zhang, 2022). This setting is also captured by LiveCodeBench, which evaluates code self-repair given an incorrect program, a failing test case, and execution feedback (Jain et al., 2025). To replace part of a program while retaining the surrounding code, AR models can use fill-in-the-middle serialization or training to condition on both the prefix and suffix (Bavarian et al., 2022; Fried et al., 2023; Ahmad et al., 2025). Masked diffusion naturally supports this infilling operation and has been applied to code generation, editing, and repair (Singh et al., 2023b; Gong et al., 2025; Xue et al., 2026; Singh et al., 2025). To accommodate replacements of different lengths, diffusion methods use deletion or expansion tokens, or predict the required span length before denoising (Wu et al., 2026b; Xu et al., 2026). PLAN-AND-PATCH applies this formulation to plan repair, where repair feedback guides infilling of selected plan regions while the surrounding text remains fixed.

Diffusion models for planning. In continuous control, diffusion models generate trajectories and may replan as the environment evolves (Janner et al., 2022; Ajay et al., 2023; Zhou et al., 2023). In discrete domains, evaluations on Countdown, Sudoku, and trip planning demonstrate their ability to generate solutions with dependencies across steps (Ye et al., 2025a;b). Studies of graph planning further show how later path tokens can guide earlier decisions (Trainin et al., 2026). A complementary approach separates planning from execution, with Latent-DARM connecting a diffusion planner to an AR executor through a latent communication interface for reasoning tasks (Berrayana et al., 2026). Our work combines plan generation and repair in interactive environments, using execution history for region selection and repair feedback for infilling.

3 Preliminaries

We introduce the structured plan representation, then review masked diffusion training and the decoding procedures used by PLAN-AND-PATCH.

(a) Complete executable plan

```
[PLAN_BEGIN]
 [GOAL] craft pink concrete powder [/GOAL]
 [THINK] ... [/THINK]
 [SG_BEGIN op=SEQ] sg1
  [SG_INTENT] prepare all materials needed for pink
   concrete powder [/SG_INTENT]
  [SG_BEGIN op=SEQ] sg1.1
   [SG_INTENT] prepare pink dye by fetching pink tulip
    and crafting it [/SG_INTENT]
   [STEP_BEGIN] sg1.1.step1
    [ACTION] fetch [/ACTION] [ARGS] 1 pink tulip [/ARGS]
    [EXPECTED_OBS] Got 1 pink tulip [/EXPECTED_OBS]
   [STEP_END] [STEP_BEGIN] sg1.1.step2
    [ACTION] craft [/ACTION] [ARGS] 2 pink dye using
     1 pink tulip [/ARGS]
    [EXPECTED_OBS] Crafted 2 minecraft:pink_dye
     [/EXPECTED_OBS] [STEP_END] [SG_END]
  [SG_BEGIN op=SEQ] sg1.2
   [SG_INTENT] fetch sand and gravel [/SG_INTENT]
   [STEP_BEGIN] sg1.2.step1
    [ACTION] fetch [/ACTION] [ARGS] 4 sand [/ARGS]
    [EXPECTED_OBS] Got 4 sand [/EXPECTED_OBS]
   [STEP_END] [STEP_BEGIN] sg1.2.step2
    [ACTION] fetch [/ACTION] [ARGS] 4 gravel [/ARGS]
    [EXPECTED_OBS] Got 4 gravel [/EXPECTED_OBS]
    [STEP_END] [SG_END] [SG_END]
 [SG_BEGIN op=SEQ] sg2
  [SG_INTENT] craft pink concrete powder from pink dye,
   sand, and gravel [/SG_INTENT]
  [STEP_BEGIN] sg2.step1
   [ACTION] craft [/ACTION] [ARGS] 8 pink concrete powder
    using 1 pink dye, 4 sand, 4 gravel [/ARGS]
   [EXPECTED_OBS] Crafted 8 minecraft:pink_concrete_powder
    [/EXPECTED_OBS] [STEP_END] [SG_END]
[PLAN_END]
```

(b) The same plan, parsed

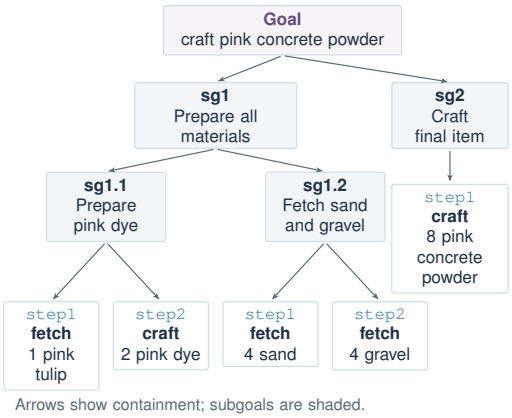


Figure 2: **A TextCraft plan and its ordered hierarchy.** All five executable steps are shown; reasoning is elided. Arrows show containment; execution visits leaves in depth-first, left-to-right order.

Definition 1 (Structured plan)**.** *A structured plan consists of a goal and an ordered hierarchy of subgoals and executable steps. Each subgoal contains an ordered list of steps or nested subgoals. Each leaf step specifies an action, its arguments, and an expected observation. The plan is serialized as a token sequence $\tau \in \mathcal{V}^L$, where $\mathcal{V}$ is the vocabulary and $L \geq 1$ is its token length. Every subgoal and step has an identifier that locates its contiguous token span in τ; we call these spans addressable. Reasoning and intent fields also have explicit delimiters.*

Masked diffusion language models. Representing structured plans as token sequences lets us formulate plan generation and infilling as conditional text generation. Let Q denote the input context, $A = (A_1, \ldots, A_T)$ a target sequence of T tokens, and p_θ a token model with parameters θ. An AR model generates tokens from left to right, with conditional log-likelihood

$$\log P_\theta(A \mid Q) = \sum_{i=1}^{T} \log p_\theta(A_i \mid A_{<i}, Q).$$

A masked diffusion model instead learns to reconstruct masked tokens from their surrounding context. During training, it draws a mask rate $t \sim \mathrm{Unif}(0, 1)$ and independently replaces each target token with `[MASK]` with probability t. Let $\widetilde{A}_t$ be the corrupted sequence and M_t the set of masked positions. The model minimizes the masked-token objective (Sahoo et al., 2024; Lou et al., 2024)

$$\mathcal{L}_{\mathrm{MDM}}(\theta) = \mathbb{E}_{A,Q,t,\widetilde{A}_t}\left[-\frac{1}{t}\sum_{i \in M_t} \log p_\theta(A_i \mid \widetilde{A}_t, Q)\right].$$

The expectation is over training context–target pairs, the mask rate, and the resulting corruption.

Decoding schedules. We use DreamReasoner's confidence-based unmasking procedure (Wu et al., 2026c), which commits sufficiently confident token predictions in parallel while keeping resolved tokens fixed. Generation decodes blocks of b positions from left to right using a fixed confidence threshold. Repair jointly infills the selected spans using DreamReasoner's RelaxedConfidence procedure, which lowers the threshold when nearby positions are reliable. Decoding settings and forward-pass statistics appear in Section A.4.1.

Each selected span is replaced by a padded sequence of masked tokens, called its canvas segment. Following deletion-token approaches (Wu et al., 2026b), we end each replacement before the first

`[delete]` token in its segment, allowing replacements of different lengths. Section 4.1 specifies the canvas allocation. The AR baseline jointly generates all replacements in one left-to-right output sequence, using fill-in-the-middle (FIM) serialization to condition on the preserved prefix, suffix, and repair feedback.

4 THE PLAN-AND-PATCH FRAMEWORK

PLAN-AND-PATCH follows a plan-and-act architecture (Sun et al., 2023; Erdogan et al., 2025), with one diffusion planner jointly trained for plan generation and repair (Figure 1). The planner generates a complete structured plan, which a separate executor translates into environment actions. After failure, the executor selects a repair region and supplies repair feedback. The planner infills this region while preserving the surrounding plan text.

4.1 PLAN GENERATION AND REPAIR

Structured plan representation. The hierarchy in Definition 1 and Figure 2 provides explicit boundaries for selecting and replacing plan text. Step and subgoal identifiers locate spans for masking during training or repair. A repair region may include a step or an enclosing subgoal, together with associated reasoning and intent fields. This supports repair at the step or subgoal level while preserving the surrounding text.

Task 1: Plan generation. Given a task goal, the initial environment observation, and the available actions, the planner generates a complete plan using the generation procedure in Section 3. The executor follows the plan strictly, carrying out its leaf steps in order. The diffusion and AR planners receive these same inputs and use the same structured plan representation.

Failure detection. An execution fails if it terminates without achieving the task goal. The *execution history* h records ordered actions and their *environment feedback*, comprising observations and error messages returned by the environment. We separately prompt the executor to select steps in the failed plan τ for revision using h, or to abstain. Selected steps and associated fields define a repair region $R \subseteq [L]$, where $[L] = \{1, \ldots, L\}$. The detector supplies *repair feedback* f_R, combining environment feedback with an explanation of why the selected text needs revision. Failure may be observed at a different step from the one requiring repair.

This procedure requires no reference plan (Section A.3.2). For training and a separate reference-assisted evaluation, a language-model judge uses a reference plan to select repair regions and supply repair feedback. In that evaluation, both planners receive the same failed plans, selected regions, and repair feedback.

Task 2: Plan repair. The planner receives the failed plan τ, selected region R, and repair feedback f_R. For a contiguous region, write $\tau = \tau_{\text{pre}} \| \tau_R \| \tau_{\text{post}}$, where $\|$ denotes concatenation. Conditioned on the preserved prefix, suffix, and repair feedback, the planner generates a replacement ρ to form

$$\tau' = \tau_{\text{pre}} \| \rho \| \tau_{\text{post}}. \tag{1}$$

The replacement may differ in length from the selected span. For multiple spans, both planners jointly generate replacements, preserving their order and all intervening text; R indexes tokens in the original plan.

The diffusion planner allocates a total of $K_0 = 2|R|$ replacement positions across the selected spans, where $|R|$ counts tokens in the original repair region. The planner then applies the infilling procedure in Section 3. Each span has its own padded canvas segment, and its replacement ends before the first `[delete]` token in that segment. The diffusion and AR planners receive the same preserved plan context and repair feedback. In our interactive evaluation, each repaired plan is executed from the task's initial environment state (Section A.1). Figure A.2 shows an ALFWorld example.

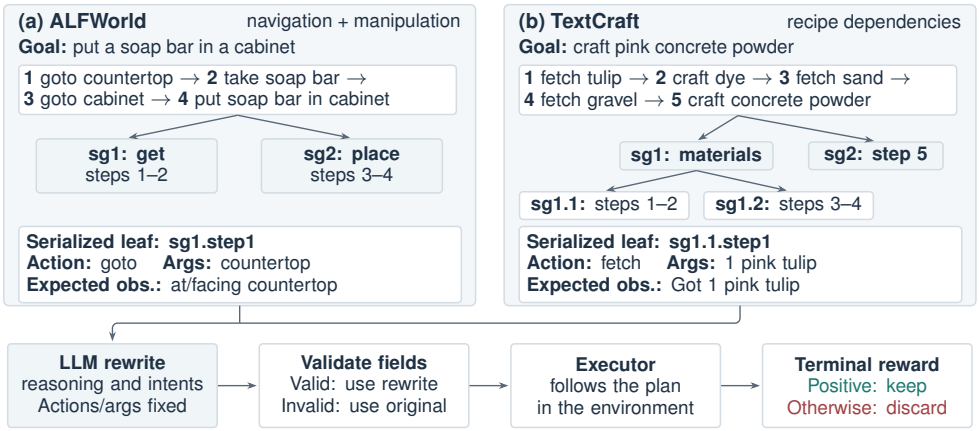


Figure 3: **Reference-plan construction.** ALFWorld groups navigation–manipulation pairs; TextCraft groups ingredient preparation before the final craft. Numbers map actions to subgoals; leaf boxes show serialized fields. Both domains share the bottom pipeline. Actions and observations are abbreviated.

4.2 Reference plans and repair examples

We partition each environment's tasks into disjoint *train-gen*, *train-rep*, *dev*, and *test* sets. The first two supply complete-plan targets and repair examples, respectively; *dev* determines training convergence and *test* is reserved for evaluation.

Reference-plan construction. We replay successful action sequences, storing each action, its arguments, and observed feedback as a leaf step (Figure 3). Fixed domain rules group leaves into subgoals. An LLM rewrites only reasoning and intents; invalid rewrites revert to the original plan. A separate executor checks for positive terminal reward. Accepted *train-gen* plans provide generation targets; *train-rep* plans provide repair references (Section A.1).

Repair examples. We train generation-only DreamReasoner-8B and Qwen3-8B planners per environment on *train-gen* and execute them on *train-rep*, retaining failed plans and their execution histories. A reference-assisted judge selects defective regions and supplies repair feedback; corresponding reference text provides the replacement targets. We supplement these real failures with structured perturbations of verified training plans, changing an object, receptacle, intent, or prerequisite. The original text in the perturbed region supplies the replacement target. We reject examples with edits outside the selected region or replacements appearing verbatim in the visible context. Both model families receive the same generation and repair examples.

4.3 Joint training and decoding

Retaining both capabilities. Each final planner starts from its original DreamReasoner-8B or Qwen3-8B checkpoint and is trained jointly on generation and repair for its environment (Figure 4). In preliminary experiments with both diffusion and AR planners, sequentially fine-tuning a generation-only planner on repair examples left repair performance roughly unchanged but substantially degraded complete-plan generation, primarily through invalid formatting. This finding motivated joint training on the combined generation and repair examples.

Targets and block sizes. Generation conditions on the goal, initial observation, and action grammar; its target is the complete plan. Repair conditions on the failed plan's preserved context and repair feedback; only replacement targets contribute to the repair loss. For diffusion, target tokens are corrupted and supervised using the masked-token objective in Section 3. DreamReasoner trains with progressively larger blocks (Wu et al., 2026c). We adapt this block-size curriculum with separate sampling distributions for generation and repair (Figure 4). Generation assigns 61.2% of its sampling weight to blocks of at most 16 tokens, whereas repair assigns 49.0% to the $4{,}096$-token block. This emphasizes short blocks for constructing plans and larger blocks for infilling supplied

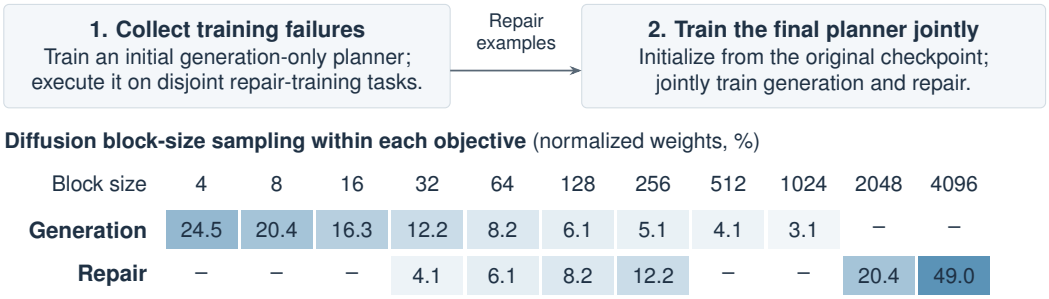


Figure 4: **Joint training with task-specific block sizes.** The initial planner supplies failures for data construction; the final planner starts from the original checkpoint and learns generation and repair jointly. We adapt DreamReasoner's block-size curriculum (Wu et al., 2026c) with separate sampling distributions for generation and repair. The lower panel shows weights normalized within each objective; dashes denote unsampled sizes. Inference uses the separate settings in Table A.11.

regions with fixed surrounding context. The AR planner learns plan generation and repair on the same examples, using FIM serialization for repair.

Inference. On ALFWorld and TextCraft, diffusion generation uses left-to-right blocks of 4 tokens with confidence threshold 0.95. Repair uses full-canvas infilling with RelaxedConfidence, threshold floor and neighbour-support cutoff 0.7, and neighbourhood radius 4 (Section 3 and Table A.11). Small generation blocks limit how much of the plan is resolved together; repair uses the preserved prefix and suffix throughout infilling. The AR planner jointly generates replacements for the selected spans in one left-to-right sequence. Section A.4 specifies optimization, padding, and prompts.

5 Experiments

We evaluate whether the jointly trained planners generate successful plans at lower latency, how repair depends on region selection, and how performance changes with the executor and plan representation. Natural Plan provides a separate evaluation of generation and repair without task-specific training.

5.1 Experimental settings

Benchmarks and data. ALFWorld (Shridhar et al., 2021) requires household navigation and manipulation; TextCraft (Prasad et al., 2024) requires dependency-constrained crafting. Task success means that executing the plan achieves the environment's goal. Table A.1 reports task splits and repair corpora constructed by Section 4.2. Task counts precede reference-plan execution filtering (Figure 3); Table A.2 reports candidate acceptance rates.

The *train-gen* and *train-rep* partitions contain 2,665/888 ALFWorld and 3,824/1,268 TextCraft tasks, respectively. Repair corpora contain 699 real plus 763 synthetic examples for ALFWorld, and 267 real plus 1,661 synthetic examples for TextCraft.

Models and optimization. The diffusion planner Diff-Plan is DreamReasoner-8B (Wu et al., 2026c), adapted from Qwen3-8B-Base. The AR planner AR-Plan is Qwen3-8B. Both have 8B parameters and share a tokenizer, task splits, training examples, and plan representation. We fully fine-tune each planner per environment with AdamW until development-set performance converges. Their prior training, decoding, and training compute differ. Section 4.3 and Table A.11 give block sizes and optimization settings. A shared Qwen3-8B executor receives no task-specific fine-tuning. ReAct (Yao et al., 2023) uses the same Qwen3-8B model to act without a separate plan.

Evaluation protocol. Generation uses 134 ALFWorld and 100 TextCraft test tasks, with limits of 60 planned leaf steps and 1,000 episode tokens. For repair, we pool failed plans from both planners:

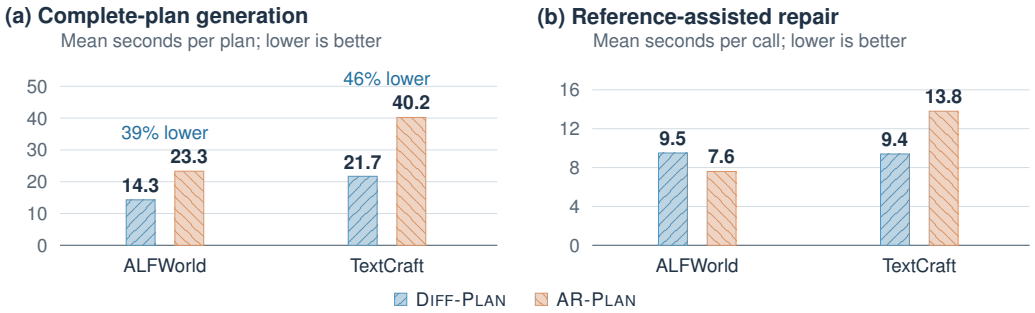


Figure 5: **Diffusion reduces mean plan-generation latency by** 39–46%**.** Values are synchronized, unbatched calls on one A100 in bf16; prompt construction, parsing, and environment execution are excluded. Repair latency varies by environment.

148 ALFWorld and 71 TextCraft failures. Each example is identified by task and generating planner, so one task may contribute two failed plans. Reference-assisted region selection uses all 219 failures. History-based selection uses the 175 failures with step-level reference annotations (126 ALFWorld, 49 TextCraft), which support evaluation of the selected regions; the detector itself sees only the plan and execution history. Both planners receive the same failed plans, selected regions, and repair feedback. Each example permits one repair attempt followed by execution from the initial state. Missing regions and abstentions count as unsuccessful. Section A.1 gives the complete population definitions.

5.2 PLAN QUALITY AND LATENCY

Table 1: **Task success under the Qwen3-8B executor (%).** Generation: 134/100 tasks; reference-assisted/history-based repair: 148/71 and 126/49 failures (ALFWorld/TextCraft), with one attempt and restart. †End-to-end provides generation-plus-repair success including recovery on each planner's own failures using the history-based repair. Dashes denote no repair stage.

Environment	Model	Plan generation	Reference-assisted repair	History-based repair	End-to-end†
ALFWorld	ReAct	38.1%	–	–	–
	DIFF-PLAN	44.0%	28.4%	14.3%	59.9%
	AR-PLAN	45.5%	27.7%	17.5%	60.6%
TextCraft	ReAct	48.0%	–	–	–
	DIFF-PLAN	63.0%	15.5%	24.5%	68.7%
	AR-PLAN	66.0%	18.3%	30.6%	72.2%

Joint training retains generation and repair. All 234 generated plans from each final planner pass the parser's format checks. Both planners outperform ReAct in generation, with a slight edge to AR. The planners differ by at most 3 percentage points in generation and reference-assisted repair (Table 1). These results show that the joint training recipe retains both generation and repair performance in a single checkpoint (Figure 4). Section A.2 reports paired generation outcomes and confidence intervals.

Parallel decoding reduces generation latency. Diffusion reduces mean generation latency by 39–46% across the two environments (Figure 5). Latency measures synchronized, unbatched bf16 decoding on one A100. Repair is faster than plan generation for both planners. Diffusion therefore generates plans faster at similar observed task success. Section A.4.1 reports replacement tokens committed per forward pass.

Repair without task-specific training. Natural Plan (Zheng et al., 2024) evaluates travel and scheduling plans using programmatic constraint validators, without an executor. Repair examples blank regions of reference plans; both planners receive the same context, repair feedback, and approximate replacement length. We use the original checkpoints; diffusion uses block size 4 for generation and 256 for repair.

AR has higher generation success, while diffusion nearly doubles repair success; both judges agree with these rankings (Table 2). Diffusion's repair advantage is concentrated in trip planning. Its flight-graph structure favours flexible decoding, analogous to diffusion's reverse decoding on synthetic Star-Path tasks (Trainin et al., 2026). Section A.3.3 gives domain results and uncertainty estimates.

Table 2: **Natural Plan without task-specific training.** Acceptance rates (%) under the benchmark validator and two judges. Generation uses 1,000 tasks; repair infills reference-plan regions in 2,661 examples from 696 tasks.

Task	Model	Validator	GPT-5	Sonnet 5
Generation	DIFF-PLAN	33.0%	20.2%	31.8%
	AR-PLAN	48.6%	34.0%	46.1%
Repair	DIFF-PLAN	53.7%	72.9%	81.6%
	AR-PLAN	27.0%	42.1%	53.4%

5.3 WHAT LIMITS EXECUTION AND REPAIR?

Region selection. On the same 175 eligible failures, with repair feedback fixed, history-based selection reduces pooled repair success relative to reference-assisted selection by 12 percentage points for diffusion and 8.6 for AR (Table A.9). These results show that selecting the repair region correctly is important for successful plan repair.

Alternative structured representation. We compare two structured formats: delimited plans and prose plans with phase headings and numbered steps. We retrain both planners, keeping task goals, executable fields, splits, and decoding settings fixed. Under Qwen3-8B, prose plans improve generation success by 17.9–22.4 percentage points on ALFWorld and 6–12 on TextCraft (Table A.7). Diffusion's observed reference-assisted repair success is lower for prose than delimited plans: 21.3% versus 28.4% on ALFWorld and 5.0% versus 15.5% on TextCraft, evaluated on their respective failure sets. AR has higher prose-plan repair success than diffusion in both environments. Plan format thus affects generation and repair differently: explicit boundaries may simplify repair-region selection and support recovery. Section A.3.1 gives complete results and executor comparisons.

Error analysis. Despite AR's slight aggregate lead, its successes do not subsume diffusion's: diffusion solves 23/134 ALFWorld and 13/100 TextCraft tasks that AR fails (Table 3). Cooling favors AR and cleaning favors diffusion. Among tasks failed by only one planner, failure at a leaf step is the most common stopping reason for both; invalid-action limits are also frequent in ALFWorld (Section A.2.1).

Table 3: **Pairwise error analysis of generated plans.** Counts partition the test tasks. In each "only" column, the other planner succeeds.

Environment (tasks)	Both succeed	Both fail	Only diffusion fails	Only AR fails
ALFWorld (134)	36	50	25	23
TextCraft (100)	50	21	16	13

6 CONCLUSION AND FUTURE WORK

PLAN-AND-PATCH combines complete-plan generation with repair through infilling, preserving unselected text. On ALFWorld and TextCraft, the planners achieve similar observed generation success, while diffusion reduces mean generation latency by 39–46%. History-based repair enables recovery without reference plans, with higher pooled success for AR. Without task-specific training on Natural Plan, AR achieves higher generation success and diffusion higher repair success. These findings support structured plans as an interface for parallel generation and repair. Several directions remain for future work, including jointly learning repair-region selection and infilling from execution feedback, adapting repair scope and decoding effort to uncertainty, and extending the framework to longer planning horizons while maintaining consistency between local edits and global task goals.

AI USE STATEMENT

We used generative AI tools to assist with experiment implementation, manuscript writing, and synthetic training-data creation. Writing assistance included editing text and formatting results. We also used language models as judges in the evaluation procedures described in Sections A.3.2 and A.4.3. Language models also assisted with training-data construction, as they rewrote reasoning and intent fields of reference plans while preserving executable actions, and identified repair regions using failed executions and reference plans (Section A.1). Candidate reference plans were checked by execution. For Natural Plan, we report program-validator scores alongside two language-model judges (Sections A.3.3 and A.4.3). The authors take responsibility for the final content of this work, including text, claims, and artifacts produced with AI assistance.

REPRODUCIBILITY STATEMENT

Section 4 describes plan generation and localized repair. Sections A.1 and 5.1 specify the benchmarks, task partitions, training-data construction, and evaluation protocols. Section A.4 provides training and decoding settings and prompts. Section A.2 reports paired outcomes and timing conditions; Sections A.3.2 and A.3.3 specify region-selection and infilling evaluations, including filtering and the bootstrap procedure.

ETHICS STATEMENT

This work evaluates language-model planning and repair in simulated environments and established planning benchmarks. These evaluations measure task success under the reported protocols; they do not establish the safety of autonomous actions in real-world settings. Deployment would require checks on permitted actions and human oversight appropriate to the consequences of failure. Language-model judges can introduce errors and biases into supervision and evaluation. We report execution-based outcomes for interactive tasks and program-validator scores alongside language-model judgments for Natural Plan to make the dependence on evaluation criteria explicit.

REFERENCES

Wasi Uddin Ahmad, Somshubra Majumdar, and Boris Ginsburg. From output to evaluation: Does raw instruction-tuned code LLMs output suffice for fill-in-the-middle code generation? *arXiv preprint arXiv:2505.18789*, 2025.

Anurag Ajay, Yilun Du, Abhi Gupta, Joshua B. Tenenbaum, Tommi Jaakkola, and Pulkit Agrawal. Is conditional generative modeling all you need for decision-making? In *International Conference on Learning Representations (ICLR)*, 2023. arXiv:2211.15657.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. arXiv:2107.03006.

Gregor Bachmann and Vaishnavh Nagarajan. The pitfalls of next-token prediction. In *International Conference on Machine Learning (ICML)*, 2024.

Mohammad Bavarian, Heewoo Jun, Nikolas Tezak, John Schulman, Christine McLeavey, Jerry Tworek, and Mark Chen. Efficient training of language models to fill in the middle. *arXiv preprint arXiv:2207.14255*, 2022.

Lina Berrayana, Ahmed Heakl, Abdullah Sohail, Thomas Hofmann, Salman Khan, and Wei Chen. Latent-DARM: Bridging discrete diffusion and autoregressive models for reasoning. *arXiv preprint arXiv:2603.09184*, 2026.

Tiwei Bie, Jiansheng Cai, Maosong Cao, Xiang Cao, Binbin Chen, Fuyuan Chen, Kun Chen, Yuqi Ding, Lun Du, Zhuocheng Gong, Yanmei Gu, Kaiyuan Guan, Xiaoxu Guo, Kairong Han, Zenan Huang, Zijin Huang, Zhenzhong Lan, Chengxi Li, Jianguo Li, Zhe Li, Ruiqi Liang, Xinyao Lin, Huabin Liu, Lin Liu, Xinyi Liu, Xuejie Liu, Guoshan Lu, Yuan Lu, Yuxin Ma, Chenhui Mao, Xiaojing Qi, Kaida Qiu, Xiaoyu Shi, Yiding Tian, Huanyu Wang, Qi Wang, Rui Wang, Shuaidi Wang, Lanning Wei, Tao Wu, Siyang Xiao, Yipeng Xing, Chenkai Xu, Song Yan, Ying Yan, Jingyi Yang, Yichun Yang, Liangyu Zha, Tianze Zhang, Yifan Zhang, Junbo Zhao, Wenqian Zhao, Da Zheng, Jianyuan Zhong, Jiahao Zhou, Junlin Zhou, Tianyu Zhou, Liwang Zhu, and Yihong Zhuang. LLaDA2.2: Enabling agentic diffusion language models via levenshtein editing. Technical report, Inclusion AI / Ant Group, 2026. https://github.com/inclusionAI/LLaDA2.X/blob/main/LLaDA2_2_tech_report.pdf.

Darshan Deshpande. Masked diffusion language models are strong and steerable text-based world models for agentic RL. *arXiv preprint arXiv:2607.16204*, 2026.

Lutfi Eren Erdogan, Nicholas Lee, Sehoon Kim, Suhong Moon, Hiroki Furuta, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. Plan-and-Act: Improving planning of agents for long-horizon tasks. *arXiv preprint arXiv:2503.09572*, 2025.

Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Wen-tau Yih, Luke Zettlemoyer, and Mike Lewis. InCoder: A generative model for code infilling and synthesis. In *International Conference on Learning Representations (ICLR)*, 2023. arXiv:2204.05999.

Shansan Gong, Ruixiang Zhang, Huangjie Zheng, Jiatao Gu, Navdeep Jaitly, Lingpeng Kong, and Yizhe Zhang. DiffuCoder: Understanding and improving masked diffusion models for code generation. *arXiv preprint arXiv:2506.20639*, 2025.

Shuhan Guo, Kun Zhang, Haifei Liu, Xingyu Gao, Yongqi Zhang, Yaqing Wang, and Quanming Yao. ContextFlow: Hierarchical task-state alignment for long-horizon embodied agents. *arXiv preprint arXiv:2605.19314*, 2026.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning through planning with language models. In *Conference on Robot Learning (CoRL)*, volume 205 of *Proceedings of Machine Learning Research*, pp. 1769–1782, 2023. URL https://proceedings.mlr.press/v205/huang23c.html.

Inception Labs, Samar Khanna, Siddhant Kharbanda, Shufan Li, Harshit Varma, Eric Wang, Sawyer Birnbaum, Ziyang Luo, Yanis Miraoui, Akash Palrecha, Stefano Ermon, Aditya Grover, and Volodymyr Kuleshov. Mercury: Ultra-fast language models based on diffusion. *arXiv preprint arXiv:2506.17298*, 2025.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida I. Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations (ICLR)*, 2025.

Michael Janner, Yilun Du, Joshua B. Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. In *International Conference on Machine Learning (ICML)*, 2022. arXiv:2205.09991.

Minseo Kim, Coleman Hooper, Aditya Tomar, Chenfeng Xu, Mehrdad Farajtabar, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. Beyond next-token prediction: A performance characterization of diffusion versus autoregressive language models. *arXiv preprint arXiv:2510.04146*, 2025.

Aditya Kumar and William W. Cohen. Localizing and correcting errors for LLM-based planners. *arXiv preprint arXiv:2602.00276*, 2026.

Claire Le Goues, ThanhVu Nguyen, Stephanie Forrest, and Westley Weimer. GenProg: A generic method for automatic software repair. *IEEE Transactions on Software Engineering*, 38(1):54–72, 2012.

Claire Le Goues, Michael Pradel, and Abhik Roychoudhury. Automated program repair. *Communications of the ACM*, 62(12):56–65, 2019.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2023. arXiv:2209.07753.

Zeyi Liu, Arpit Bahety, and Shuran Song. REFLECT: Summarizing robot experiences for failure explanation and correction. In *Conference on Robot Learning (CoRL)*, 2023. arXiv:2306.15724.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. In *International Conference on Machine Learning (ICML)*, 2024. ICML 2024 Oral; arXiv:2310.16834.

Qingyu Lu, Liang Ding, Kanjian Zhang, Jinxia Zhang, and Dacheng Tao. The bitter lesson of diffusion language models for agentic workflows: A comprehensive reality check. In *Proceedings of the Association for Computational Linguistics (ACL), Main Conference*, 2026. arXiv:2601.12979.

Zhongwen Luan, Xiaoyu Zhang, Ming Hu, Yue Yang, Jiongchi Yu, and Xiaohong Chen. Repair or resample? rethinking failure debugging in LLM multi-agent systems. *arXiv preprint arXiv:2608.25920*, 2026.

Linrui Ma, Yufei Cui, Kai Han, and Yunhe Wang. Diffusion In Diffusion: Reclaiming global coherence in semi-autoregressive diffusion. *arXiv preprint arXiv:2601.13599*, 2026.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. arXiv:2303.17651.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. *arXiv preprint arXiv:2502.09992*, 2025.

Jiangweizhi Peng, Yuanxin Liu, Ruida Zhou, Charles Fleming, Zhaoran Wang, Alfredo Garcia, and Mingyi Hong. HiPER: Hierarchical plan–execute RL for multi-turn LLM agents. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2026. URL https://openreview.net/forum?id=RJWQcgd3hX.

Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. ADaPT: As-needed decomposition and planning with language models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, 2024. arXiv:2311.05772.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T. Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2406.07524.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations (ICLR)*, 2021. arXiv:2010.03768.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. ProgPrompt: Generating situated robot task plans using large language models. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2023a. arXiv:2209.11302.

Mukul Singh, José Cambronero, Sumit Gulwani, Vu Le, Carina Negreanu, and Gust Verbruggen. CodeFusion: A pre-trained diffusion model for code generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 11697–11708, 2023b. doi: 10.18653/v1/2023.emnlp-main.716. URL https://aclanthology.org/2023.emnlp-main.716/.

Mukul Singh, Gust Verbruggen, Vu Le, and Sumit Gulwani. Diffusion is a code repair operator and generator. *arXiv preprint arXiv:2508.11110*, 2025.

Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M. Sadler, Wei-Lun Chao, and Yu Su. LLM-Planner: Few-shot grounded planning for embodied agents with large language models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023. arXiv:2212.04088.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. AdaPlanner: Adaptive planning from feedback with language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. arXiv:2305.16653.

Itamar Trainin, Shauli Ravfogel, Omri Abend, and Amir Feder. Discrete diffusion models exploit asymmetry to solve lookahead planning tasks. *arXiv preprint arXiv:2602.19980*, 2026.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research (TMLR)*, 2024. arXiv:2305.16291.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*, 2023a. arXiv:2305.04091.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. ScienceWorld: Is your agent smarter than a 5th grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2022. arXiv:2203.07540.

Yinjie Wang, Ling Yang, Bowen Li, Ye Tian, Ke Shen, and Mengdi Wang. Revolutionizing reinforcement learning framework for diffusion large language models. *arXiv preprint arXiv:2509.06949*, 2025.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023b. arXiv:2302.01560.

Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. Fast-dLLM: Training-free acceleration of diffusion LLM by enabling KV cache and parallel decoding. *arXiv preprint arXiv:2505.22618*, 2025.

Wenyi Wu, Sibo Zhu, Kun Zhou, and Biwei Huang. Planner Matters! an efficient and unbalanced multi-agent collaboration framework for long-horizon planning. *arXiv preprint arXiv:2605.02168*, 2026a.

Zirui Wu, Lin Zheng, Zhihui Xie, Jiacheng Ye, Jiahui Gao, Shansan Gong, Yansong Feng, Zhenguo Li, Wei Bi, Guorui Zhou, and Lingpeng Kong. DreamOn: Diffusion language models for code infilling beyond fixed-size canvas. In *International Conference on Learning Representations (ICLR)*, 2026b. arXiv:2602.01326.

Zirui Wu, Lin Zheng, Jiacheng Ye, Shansan Gong, Xueliang Zhao, Yansong Feng, Wei Bi, and Lingpeng Kong. DreamReasoner-8B: Block-size curriculum learning for diffusion reasoning models. *arXiv preprint arXiv:2606.19257*, 2026c.

Chunqiu Steven Xia and Lingming Zhang. Less training, more repairing please: Revisiting automated program repair via zero-shot learning. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pp. 959–971, 2022.

Haobo Xu, Sirui Chen, Yuanchen Bei, Lingjie Chen, Yuchen Yan, Dongqi Fu, Jingrui He, and Hanghang Tong. Predict, don't iterate: Efficient adaptive-length infilling for diffusion language models. *arXiv preprint arXiv:2609.02108*, 2026.

Anton Xue, Litu Rout, Constantine Caramanis, and Sanjay Shakkottai. AnCoder: Anchored code generation via discrete diffusion models. *arXiv preprint arXiv:2602.17688*, 2026.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023. arXiv:2210.03629.

Jiacheng Ye, Jiahui Gao, Shansan Gong, Lin Zheng, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Beyond autoregression: Discrete diffusion for complex reasoning and planning. In *International Conference on Learning Representations (ICLR)*, 2025a. arXiv:2410.14157.

Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7b: Diffusion large language models. *arXiv preprint arXiv:2508.15487*, 2025b.

Alejandra Zambrano, Sara Vera Marjanovic, Imene Kerboua, Xing Han Lù, and Leila Kosseim. Does the way you plan matter? an empirical study of planning representations for LLM web agents. *arXiv preprint arXiv:2605.29927*, 2026.

Tuo Zhang, Alin-Ionut Popa, Yan Xu, Rui Song, and Dimitrios Dimitriadis. PIVOT: Bridging planning and execution in LLM agents via trajectory refinement. *arXiv preprint arXiv:2605.11225*, 2026a.

Yue Zhang, Sihan Chen, Ziwen Huang, Hanyun Cui, Kangye Ji, and Zhi Wang. Atomic Task Graph: A unified framework for agentic planning and execution. *arXiv preprint arXiv:2607.01942*, 2026b.

Zhenyu Zhang, Tianyi Chen, Weiran Xu, Alex Pentland, and Jiaxin Pei. ReCAP: Recursive context-aware reasoning and planning for large language model agents. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. arXiv:2510.23822.

Jiahao Zhao, Shaoxuan Xu, Zhongxiang Sun, Fengqi Zhu, Jingyang Ou, Yuling Shi, Chongxuan Li, Xiao Zhang, and Jun Xu. DLLM-Searcher: Adapting diffusion large language model for search agents. *arXiv preprint arXiv:2602.07035*, 2026.

Siyan Zhao, Devaansh Gupta, Qinqing Zheng, and Aditya Grover. d1: Scaling reasoning in diffusion large language models via reinforcement learning. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2025. arXiv:2504.12216.

Huiling Zhen, Weizhe Lin, Renxi Liu, Kai Han, Yiming Li, Yuchuan Tian, Hanting Chen, Xiaoguang Li, Xiaosong Li, Chen Chen, Xianzhi Yu, Mingxuan Yuan, Youliang Yan, Peifeng Qin, Jun Wang, Yu Wang, Dacheng Tao, and Yunhe Wang. DLLM agent: See farther, run faster. *arXiv preprint arXiv:2602.07451*, 2026.

Huaixiu Steven Zheng, Swaroop Mishra, Hugh Zhang, Xinyun Chen, Minmin Chen, Azade Nova, Le Hou, Heng-Tze Cheng, Quoc V. Le, Ed H. Chi, and Denny Zhou. NATURAL PLAN: Benchmarking llms on natural language planning. *arXiv preprint arXiv:2406.04520*, 2024.

Siyuan Zhou, Yilun Du, Shun Zhang, Mengdi Xu, Yikang Shen, Wei Xiao, Dit-Yan Yeung, and Chuang Gan. Adaptive online replanning with diffusion models. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2023. arXiv:2310.09629.

Fengqi Zhu, Rongzhen Wang, Shen Nie, Xiaolu Zhang, Chunwei Wu, Jun Hu, Jun Zhou, Jianfei Chen, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. LLaDA 1.5: Variance-reduced preference optimization for large language diffusion models. *arXiv preprint arXiv:2505.19223*, 2025.

A SUPPLEMENTARY MATERIAL

The appendix is organized as follows. Section A.1 defines the data and evaluation populations. Section A.2 reports the complete primary results. Section A.3 collects representation, executor, region-selection, Natural Plan, and long-horizon analyses. Section A.4 records training, decoding, and prompt details.

A.1 PROTOCOLS AND DATA

ALFWorld combines household navigation and manipulation; TextCraft requires dependency-constrained crafting. Natural Plan contains trip, meeting, and calendar scheduling problems with programmatic validators that check whether a completed plan satisfies the task constraints. ScienceWorld is used only for the long-horizon generation analysis in Section A.3.4.

A.1.1 TASK SPLITS AND TRAINING EXAMPLES

ALFWorld and TextCraft are partitioned by task identity into disjoint *train-gen*, *train-rep*, *dev*, and *test* sets. *train-gen* supplies complete-plan targets. Generation-only DreamReasoner-8B and Qwen3-8B checkpoints, trained separately per environment on that split, produce failures on *train-rep*. The *dev* split determines training convergence. Primary interactive evaluation uses *test*. Both model families use the same task splits and training examples.

Table A.1: **Task splits and repair-training examples.** Task counts precede reference-plan execution filtering. Repair examples combine failed planner executions and structured perturbations of verified plans.

	Task split				Repair-training examples		
Environment	train-gen	train-rep	dev	test	real	synthetic	total
ALFWorld	2,665	888	140	134	699	763	1,462
TextCraft	3,824	1,268	50	100	267	1,661	1,928

Dataset-specific reference construction. The benchmark task specifies the goal; the subgoal decomposition uses fixed, domain-specific rules. ALFWorld references start from expert demonstrations. We group navigation followed by manipulation into subgoals. TextCraft uses a deterministic solver that expands recipe dependencies backward from the target item; these dependencies define the subgoal hierarchy. In both environments, actions are replayed to record observations. We render each action, its arguments, and its observed environment feedback as a leaf step, then serialize these steps in execution order. This constructs the initial plan deterministically from the source sequence and replayed feedback. Claude Sonnet 5 with default model settings rewrites reasoning and intent fields from the grouped plan and its evidence while preserving actions and arguments. The returned fields are rendered and validated; an invalid rewrite falls back to the original structured plan. Claude Sonnet 5 with default model settings follows each candidate as the verification executor in the live environment. Positive terminal reward is required for acceptance (Figure 3).

Table A.2: **Reference-plan candidates.** *Train* is the union of *train-gen* and *train-rep*. Counts precede execution filtering; verified *train-gen* candidates supply generation targets, while verified *train-rep* references provide repair targets for failed plans. Mean, median, and maximum refer to the number of leaf steps; success is measured by execution.

Environment	Split	Candidates	Success	Mean	Median	Max
ALFWorld	train	3,553	91.0%	5.8	6	8
	dev	140	90.7%	5.7	6	8
	test	134	91.0%	5.6	6	8
	all	3,827	91.0%	5.8	6	8
TextCraft	train	5,092	99.8%	9.8	7.0	66
	dev	50	100%	9.9	7.0	91
	test	100	99.0%	8.0	6.5	33
	all	5,242	99.8%	9.8	7.0	91

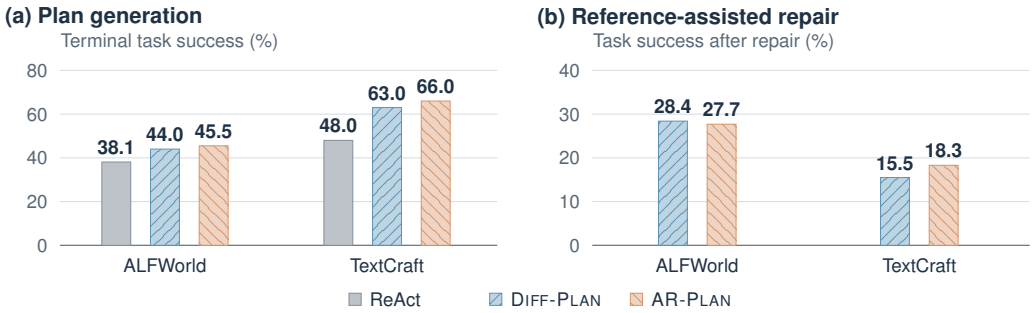


Figure A.1: **Observed success in the two interactive benchmarks.** (a) Task success after plan generation, on 134 ALFWorld and 100 TextCraft tasks. (b) Task success after reference-assisted repair, on 148 and 71 pooled failed plans. Diffusion–AR differences are at most three percentage points; higher is better.

For repair training, we retain failed plans with their execution history and first observed failure. A reference-assisted judge identifies the smallest addressable repair region associated with the failure. The corresponding text from the verified reference plan provides the replacement target. Structured perturbations supplement these real failures. They modify fields of the parsed plan, such as an object, receptacle, intent, or prerequisite. An example is rejected if the edit changes text outside the selected region or if its replacement occurs verbatim in the remaining prompt.

We query Claude Sonnet 5 (`us.anthropic.claude-sonnet-5`) once with default model settings as the reference-assisted judge. It receives the goal, failed plan, execution history, failure observation, and reference plan, and is instructed to return the identifiers of the smallest set of steps whose differences from the reference explain the observed failure, followed by a brief explanation.

A.1.2 Interactive evaluation populations

Generation is evaluated on 134 ALFWorld and 100 TextCraft test tasks. The resulting failures are pooled across planners, giving 148 failed plans in ALFWorld (75 from DIFF-PLAN and 73 from AR-PLAN) and 71 in TextCraft (37 from DIFF-PLAN and 34 from AR-PLAN). Each repair example is identified by its task and generating planner, so a task failed by both planners contributes two examples. All failed plans remain in the primary reference-assisted repair population, including those with no usable repair region, which count as unsuccessful.

History-based region selection is evaluated on $126/148$ ALFWorld and $49/71$ TextCraft failed plans with a step labelled by the reference-assisted judge. These labels provide the step-level reference targets needed to evaluate region selection. The other 44 failed plans (22 from each environment) lack such a label and are excluded from this comparison. On this subset of 175 failed plans, reference-assisted repair achieves task success with DIFF-PLAN (29.1%) and AR-PLAN (29.7%), compared with DIFF-PLAN (24.2%) and AR-PLAN (24.7%) on all 219 primary failures. A stored failure-detector output is available for every included example, and abstentions remain in the denominator.

Each failed plan permits one repair attempt. Both planners infill the selected region, and the repaired plan is executed from the task's initial environment state rather than resumed from the failure. The Qwen3-8B executor uses temperature 0.6 and top-p 0.95. Reported repair success is terminal task success after this repair-and-restart procedure.

A.2 Complete primary results

Table 1 reports success rates and evaluation denominators for generation and both repair protocols. Figure A.1 plots the generation and reference-assisted repair results.

Table A.3: **Paired outcomes.** Generation compares the planners on the same test tasks. Reference-assisted repair compares them on pooled failed plans, which may include two plans from one task. Columns indicate which planners succeed.

	Plan generation				Reference-assisted repair			
Environment	both	DIFF-PLAN only	AR-PLAN only	neither	both	DIFF-PLAN only	AR-PLAN only	neither
ALFWorld	36	23	25	50	24	18	17	89
TextCraft	50	13	16	21	6	5	7	53

A paired normal approximation based on tasks with different outcomes across planners gives 95% confidence intervals of $[-11.6, 8.6]$ and $[-13.5, 7.5]$ percentage points for the diffusion-minus-AR generation difference on ALFWorld and TextCraft. These intervals describe held-out task variation conditional on the trained checkpoints and recorded execution outcomes. Repair examples are grouped by task, since a task may contribute a failed plan from each planner.

Mean generation latency is lower for DIFF-PLAN (14.3 seconds on ALFWorld and 21.7 on TextCraft) than for AR-PLAN (23.3 and 40.2 seconds, respectively). For reference-assisted repair, AR-PLAN (7.6 seconds) is faster than DIFF-PLAN (9.5 seconds) on ALFWorld, while DIFF-PLAN (9.4 seconds) is faster than AR-PLAN (13.8 seconds) on TextCraft. Measurements use synchronized, unbatched bf16 decoding on one A100 and exclude prompt construction, parsing, and environment execution.

A.2.1 SHARED AND MODEL-EXCLUSIVE FAILURES

We compare the two generated plans on each primary test task, using the same Qwen3-8B executor and evaluation protocol as Table 1. A shared failure means both plans fail; a diffusion-only failure means the diffusion plan fails and the AR plan succeeds, and conversely for an AR-only failure. Of 134 ALFWorld tasks, 50 are shared failures, 25 diffusion-only failures, and 23 AR-only failures. The corresponding counts on 100 TextCraft tasks are 21, 16, and 13 (Table A.3).

Shared difficulties and differences by task. Table A.4 breaks down ALFWorld outcomes by task family. Both planners fail on $12/18$ light-inspection tasks and $10/17$ two-object placement tasks, compared with $3/24$ simple placement tasks. Their similar aggregate success rates also conceal differences across task families: cooling has more diffusion-only failures (9 versus 2), while cleaning has more AR-only failures (7 versus 3). These are descriptive comparisons on small task subsets, conditional on the evaluated checkpoints and executions.

Table A.4: **Paired ALFWorld generation failures by task family.** Each task contributes to at most one failure column; tasks where both planners succeed are omitted from those columns but included in the total.

Task family	Tasks	Both fail	Only diffusion fails	Only AR fails
Light inspection	18	12	3	0
Simple placement	24	3	2	5
Clean then place	31	12	3	7
Cool then place	21	6	9	2
Heat then place	23	7	4	6
Two-object placement	17	10	4	3
Total	134	50	25	23

Stopping reasons when only one planner fails. Table A.5 reports the executor's recorded stopping reason on tasks where exactly one generated plan succeeds. Failure at a leaf step is the most frequent reason for both planners in both environments. The invalid-action limit also occurs frequently in ALFWorld ($9/25$ diffusion-only and $8/23$ AR-only failures). Completing the plan without terminal success occurs in $4/25$ diffusion-only ALFWorld failures and none of the 23 AR-only failures; in TextCraft it occurs for both planners ($3/16$ and $1/13$). Thus, these model-exclusive failed tasks mostly share stopping-reason categories. The labels describe how execution stops, and the stopping-reason breakdown covers tasks where exactly one planner succeeds.

Table A.5: **Stopping reasons on tasks failed by only one planner.** Columns identify the failing planner; the other planner succeeds on each counted task. Counts cover every model-exclusive failure.

	ALFWorld		TextCraft	
Recorded stopping reason	Diffusion	AR	Diffusion	AR
Failure at a leaf step	10	13	10	7
No-progress limit	2	2	2	3
Plan completed without terminal success	4	0	3	1
Invalid-action limit	9	8	1	2
Total	25	23	16	13

Failure locations across both planners. We analyze the 148 ALFWorld and 71 TextCraft failed executions from the primary generation evaluation. These diagnostics pool failures from both planners, so a task failed by both contributes two executions. A failing leaf step is identified in 106 ALFWorld and 14 TextCraft executions; the remaining 42 and 57 failures have no identified failing leaf step in this diagnostic. This coverage differs from the reference-assisted judge's repair-region annotations used in Table A.9.

Table A.6: **Where failed executions stop.** Action and reference-comparison counts cover failures with an identified leaf step and describe the observed failure location.

Diagnostic	ALFWorld	TextCraft
All failed executions	148	71
Identified failing leaf step	106	14
Unidentified failing leaf step	42	57
Action at the identified failing leaf step		
`take`	89	0
`fetch`	0	12
`goto`	7	0
`put`	7	0
`clean`, `cool`, `heat`	1 each	0
`craft`	0	2
Identified failing leaf step compared with the reference		
Identical	103	9
Different	2	5
Absent from the reference	1	0

Object acquisition accounts for $89/106$ identified failures in ALFWorld and $12/14$ in TextCraft. The failing leaf step matches the reference in $103/106$ and $9/14$ cases. These counts describe agreement at the failing leaf step. The action distribution is conditional on identifying that step, covering $106/148$ ALFWorld and $14/71$ TextCraft failed executions.

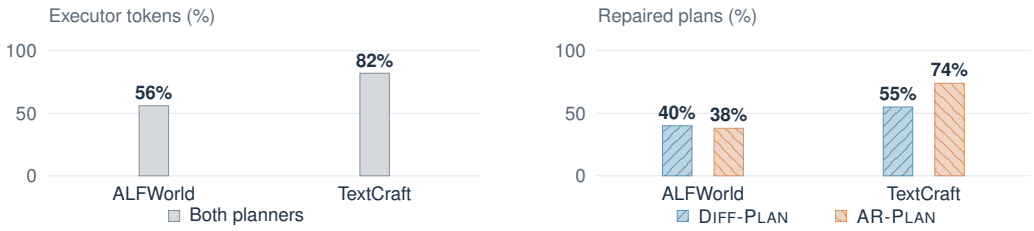
A.2.2 Worked repair and preservation

Figure A.2 illustrates reference-assisted repair of an ALFWorld plan. The selected region includes a leaf step and its associated reasoning and intent fields; executable steps outside the region remain unchanged.

We analyze 104 ALFWorld and 38 TextCraft failed executions with at least one completed step. Figure A.3a reports the pooled fraction of executor tokens consumed before failure: 56% on ALFWorld and 82% on TextCraft. After reference-assisted repair, the fractions of diffusion and AR plans that parse and preserve every completed step unchanged are 40% and 38% on ALFWorld, and 55% and 74% on TextCraft (Figure A.3b). These are measurements of plan-text preservation; repaired plans are executed from the initial state. The fraction of repairs that parse and preserve completed steps is higher for AR on TextCraft and similar across planners on ALFWorld.



Figure A.2: **Plan repair in ALFWorld.** In this training example, a reference-assisted judge selects one leaf step, the reasoning block, and three intent fields for regeneration. Infilling changes `countertop` to `toilet` and updates the expected observation. The other seven leaf steps remain fixed. The repaired plan succeeds.



(a) Executor tokens before failure.

(b) Preservation of completed steps.

Figure A.3: **Interaction before failure and preservation after repair.** We analyze 104 ALFWorld and 38 TextCraft failed executions with at least one completed step. Plans come from both planners and use the Qwen3-8B executor. (a) Pooled fraction of executor tokens consumed before the recorded failure point. (b) Fraction of plans after reference-assisted repair that parse and preserve every completed step, shown separately for each planner. Preservation compares completed-step text before and after repair; repaired plans are executed from the initial state.

A.3 Sensitivity Analyses

A.3.1 Representation and Executor

Alternative plan representation. We retrain both planners in each environment using an alternative structured plan representation with phase headings and numbered prose steps in place of structural delimiters. We refer to these two structured formats as prose and delimited plans, respectively. Task goals, executable fields, task splits, and model-specific decoding methods remain unchanged. For each representation, Qwen3-8B and Claude Sonnet 5 execute the same stored plans on the same tasks with the same budgets.

Table A.7: **Generation under two representations and executors.** Entries are task success rates. Within each representation, both executors receive identical stored plan text.

		Qwen3-8B executor		Sonnet 5 executor	
Environment	Planner	Delimited	Prose	Delimited	Prose
ALFWorld	DIFF-PLAN	44.0%	61.9%	96.3%	95.5%
	AR-PLAN	45.5%	67.9%	94.8%	94.8%
TextCraft	DIFF-PLAN	63.0%	69.0%	94.0%	94.0%
	AR-PLAN	66.0%	78.0%	94.0%	96.0%

With Qwen3-8B, prose plans improve task success in both environments (Table A.7). The increase on ALFWorld is from 44.0% to 61.9% for DIFF-PLAN and from 45.5% to 67.9% for AR-PLAN. Holding plan text fixed, Sonnet 5 achieves higher success than Qwen3-8B for both planners and representations, showing that executor choice also affects task success.

For reference-assisted repair of prose plans, we evaluate only failures for which the judge selects a usable repair region. This subset differs from the primary repair population, which retains failures without a selected region. Table A.8 reports counts before and after selection. As in the primary evaluation, a task failed by both planners contributes two distinct failed plans.

Table A.8: **Reference-assisted repair of prose plans.** "Failures" gives counts from DIFF-PLAN and AR-PLAN, in that order; "Eligible" counts plans with a selected repair region. Both planners receive the same failed plans and regions. Repair entries report task success after repair and restart.

Environment	Executor	Failures	Eligible	DIFF-PLAN repair	AR-PLAN repair
ALFWorld	Qwen3-8B	$51 + 43 = 94$	$46 + 43 = 89$	19/89 (21.3%)	49/89 (55.1%)
TextCraft	Qwen3-8B	$31 + 22 = 53$	$14 + 6 = 20$	1/20 (5.0%)	6/20 (30.0%)
ALFWorld	Sonnet 5	$6 + 7 = 13$	13	4/13 (30.8%)	6/13 (46.2%)
TextCraft	Sonnet 5	$6 + 4 = 10$	5	2/5 (40.0%)	1/5 (20.0%)

With Qwen3-8B, AR-PLAN achieves higher prose-plan repair success in both environments. With Sonnet 5, the ordering differs across environments, although the TextCraft comparison includes only five failed plans. Repair outcomes depend on both representation and executor.

A.3.2 REPAIR FROM EXECUTION HISTORY

We prompt the Qwen3-8B executor separately to act as the history-based failure detector. It receives the goal, current plan, execution history, and environment feedback, without a reference plan. In one call, it selects one or more step identifiers and supplies repair feedback, or abstains. For this comparison, repair feedback is held fixed across selection methods. Section A.1 defines the evaluated subset.

Table A.9: **Repair under reference-assisted and history-based region selection.** Entries are task success rates after repair. Paired columns use the same failed plans and repair feedback; abstentions remain in the history-based denominator.

	DIFF-PLAN		AR-PLAN	
Environment	Reference-assisted	History-based	Reference-assisted	History-based
ALFWorld ($n = 126$)	31.7%	14.3%	32.5%	17.5%
TextCraft ($n = 49$)	22.4%	24.5%	22.4%	30.6%
Pooled ($n = 175$)	29.1%	17.1%	29.7%	21.1%

Relative to reference-assisted selection, history-based selection lowers repair success on ALFWorld and increases it on TextCraft for both planners.

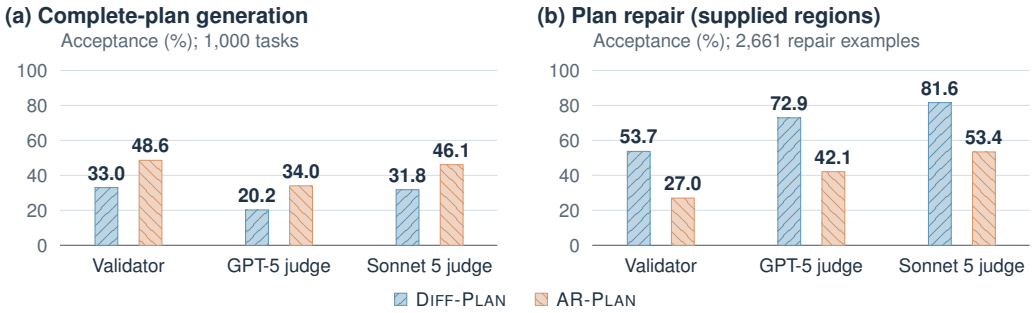


Figure A.4: **Plan generation and repair on Natural Plan.** Both models are evaluated without task-specific fine-tuning. Generation uses 1,000 tasks; repair infills supplied regions in 2,661 examples from 696 tasks. Bars show acceptance rates under the benchmark validator and two language-model judges. Section A.3.3 gives the filtering protocol and complete results.

A.3.3 NATURAL PLAN GENERATION AND REPAIR

We evaluate `Dream-org/DreamReasoner-8B` and `Qwen/Qwen3-8B` without task-specific fine-tuning. Generation uses 1,000 tasks with reference plans that pass the benchmark validators, comprising 400 trip-planning, 300 meeting-scheduling, and 300 calendar-scheduling tasks. Reasoning is disabled for both checkpoints; DIFF-PLAN uses block size 4 for generation and 256 for repair.

Each planner receives the benchmark's `prompt_5shot` field, containing the task instruction, five released demonstrations, the test task, and the final `SOLUTION:` prefix. For repair, we blank one or two regions of the reference plan, which follows the benchmark's structured natural-language format. Both planners receive the same preserved prefix, suffix, repair feedback, and approximate replacement length, and decode only the missing text.

We construct five repair candidates per reference plan. Three constructions remove a single region, either a random step, a phrase within a step, or a constraint-determined value. The other two remove a pair of regions, either mutually constrained regions or regions in different parts of the plan. We exclude candidates whose target text occurs verbatim in the visible plan or prompt.

Of the 5,000 candidates, we retain 2,661 for which restoring the reference text passes the validator and filling every blank with a fixed nonsensical string fails. This invalid-replacement control therefore has 0% acceptance on the retained examples by construction. A separate do-nothing control removes all blank markers without inserting text. Its validator acceptance falls from 46.3% on all candidates to 1.3% on the retained examples. All repair results below use the retained population.

The benchmark validator checks each completed plan directly, without an executor. Two LLM judges also assess completed plans, receiving only the task and candidate plan, without the reference plan or validator output (Section A.4.3). For repair, we additionally report exact replacement recovery, which requires all replacements to match the reference text after Unicode normalization, whitespace collapse, and removal of leading and trailing punctuation.

Table 2 reports validator and judge acceptance. Exact replacement recovery is 22.7% for DIFF-PLAN and 5.0% for AR-PLAN.

Table A.10: **Natural Plan repair by domain and construction.** Entries are validator acceptance rates on the 2,661 retained examples. Calendar references contain only a few words specifying a meeting day and time; no calendar candidates pass the repair filters, so this domain is evaluated only for generation.

Domain	n	DIFF-PLAN	AR-PLAN
Trip planning	1,261	93.3%	35.7%
Meeting scheduling	1,400	18.1%	19.2%
Calendar scheduling	0	–	–

Construction	n	DIFF-PLAN	AR-PLAN
Random step	545	52.7%	40.7%
Random word span	491	65.0%	28.5%
Constraint assignment	482	69.5%	47.9%
Causally linked regions	541	34.2%	0.7%
Compound nonlocal regions	602	50.5%	20.3%

Natural Plan: reconstruct one constraint-determined phrase

Task. Visit 4 European cities in 17 days on direct flights: Stockholm 2 days (conference days 1–2), Dubrovnik 7 days, Vienna 5 days, Mykonos 6 days (friends there days 12–17). Direct flights: Dubrovnik–Vienna, Stockholm–Vienna, Vienna–Mykonos, Stockholm–Dubrovnik.

Repair feedback. A short phrase inside one step states a value that the task's constraints do not permit.

Plan supplied to both planners. One region is blanked; the remainder is stated correct.

```
Day 1-2: Arriving in Stockholm and visit Stockholm for 2 days.
Day 2: Fly from Stockholm to Dubrovnik.  Day 2-8: Visit Dubrovnik for 7 days.
Day 8: Fly from Dubrovnik to Vienna.  Day 8-12: Visit Vienna for 5 days.
                                  Mykonos.   (about 30 characters, 6 words)
Day 12-17: Visit Mykonos for 6 days.
```

DIFF-PLAN: exact replacement

```
Day 12: Fly from Vienna to
```

The inserted phrase joins the preserved word `Mykonos.` to form the required flight.

AR-PLAN: incorrect replacement

```
Day 12-17: Visit Mykonos for 6 days.
```

The inserted phrase repeats the following line and leaves an extra `Mykonos.`

Figure A.5: **An example of plan repair on Natural Plan.** Both planners receive the same task, repair feedback, preserved prefix and suffix, and approximate replacement length. The blank is the prefix of a line, so its replacement must join the preserved suffix. The diffusion output restores the reference phrase; the AR output repeats the following line. This example illustrates the within-task comparison in Figure A.4.

The validator and judges agree on the model ordering within each operation, despite their different acceptance criteria and absolute success rates. The aggregate repair advantage is concentrated in trip planning, while DIFF-PLAN has higher validator acceptance under each of the five constructions. A percentile bootstrap with 50,000 resamples clustered by the 696 retained source tasks gives a 95% interval of $[23.8, 29.6]$ percentage points for the aggregate DIFF-PLAN-minus-AR-PLAN difference. Each resample draws source tasks with replacement and includes all their repair examples.

A.3.4 LONG-HORIZON GENERATION IN SCIENCEWORLD

The ScienceWorld checkpoints are trained separately on simulator-provided reference plans (Wang et al., 2022). These references average 27.1 leaf steps, compared with 5.8 for ALFWorld and 9.8 for TextCraft. We evaluate 825 held-out generations per planner and group them by reference-plan length.

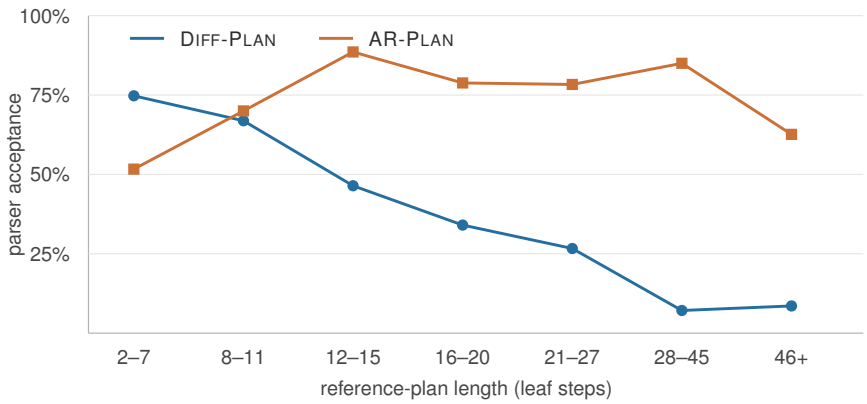


Figure A.6: **ScienceWorld parser acceptance by reference-plan length.** Rates are computed over 825 held-out generations per planner. Acceptance for DIFF-PLAN falls from 74.7% in the shortest bin to 7–9% at 28 or more reference steps, with pooled parser acceptance of 34.3% for DIFF-PLAN and 73.8% for AR-PLAN.

Table A.11: **ALFWorld and TextCraft training and decoding settings.** Block weights follow the listed block-size order.

Setting	Plan generation	Plan repair
Training endpoint	Convergence of development-set performance	
Learning rate	10^{-5}	10^{-5}
Batch × gradient accumulation	1×16	1×16
Learning-rate warmup	5% of optimization steps	
Maximum sequence length	4,096 tokens at training and inference	
DIFF-PLAN block sizes	$4, 8, 16, 32, 64, 128, 256, 512, 1024$	$32, 64, 128, 256, 2048, 4096$
DIFF-PLAN block weights	$(24, 20, 16, 12, 8, 6, 5, 4, 3)$	$(2, 3, 4, 6, 10, 24)$
DIFF-PLAN inference	Block-wise denoising, $b = 4$, $\eta = 0.95$	Full-canvas infilling, $\eta = 0.95$, $\eta_{\min} = \kappa = 0.7$, $r = 4$
AR-PLAN inference	Left-to-right with key–value cache	Left-to-right replacement generation using FIM serialization
Executor	Qwen3-8B, temperature 0.6, top-p 0.95, one environment action per call	

Of 758 rejected generations across both planners, 754 reach the generation limit without closing the plan hierarchy; the remaining four omit a subgoal body. Rejected outputs also contain repeated actions, with a median duplicated action–argument fraction of 93.3% for DIFF-PLAN and 40.0% for AR-PLAN. Among parser-rejected plans, the dominant failure is incomplete serialization, accompanied by extensive repetition, especially for diffusion.

For a nonempty plan with H leaf steps, let (u_ℓ, a_ℓ) denote the action–argument pair at step $\ell \in [H]$ after flattening the hierarchy. The duplicated fraction is

$$\frac{|\{\ell \in [H] : \exists k \in [H] \setminus \{\ell\},\ (u_k, a_k) = (u_\ell, a_\ell)\}|}{H}.$$

This counts every occurrence of a repeated pair, including its first occurrence.

A.4 IMPLEMENTATION DETAILS AND PROMPTS

A.4.1 TRAINING AND DECODING

On ALFWorld and TextCraft, we fully fine-tune the checkpoints specified in Section 5.1 with AdamW and bf16 weights. Each environment-specific checkpoint is trained jointly on generation and repair, with both objectives present in a single run. An epoch visits each generation and repair example once. We use one training seed (42); reported evaluation uncertainty does not include variation across training seeds. Activation checkpointing and Fully Sharded Data Parallel (FSDP) with full sharding and CPU parameter offloading support full fine-tuning. The planners share task splits, training examples, and evaluation interfaces (Table A.11). Native reasoning mode is disabled for both planners. The plan's `[THINK]` field remains part of the supervised structured output.

For diffusion decoding, η is the base confidence threshold. Repair uses DreamReasoner's Relaxed-Confidence procedure (Wu et al., 2026c), with threshold floor $\eta_{\min}$, neighbour-support cutoff κ, and neighbourhood radius r.

Diffusion repair allocates $2|R|$ positions in total across the selected spans. Each span has its own padding, and its replacement ends before the first `[delete]` token in its canvas segment. Across environment-specific settings, a diffusion repair forward pass commits 7.7–18.3 tokens; the AR decoder generates one token per forward pass using a key–value cache.

A.4.2 EXECUTOR BEHAVIOR

The primary executor is Qwen3-8B without task-specific fine-tuning. Each execution call receives the goal, current observation, complete plan, execution history, and action grammar, and generates one environment action. The executor strictly follows the planned actions and arguments in leaf-step order. Execution stops on success, environment failure, an invalid-action or

no-progress limit, 60 planned leaf steps, or 1,000 episode tokens. Parser rejection and execution ending without task success count as failure. The executor comparison uses Claude Sonnet 5 (`us.anthropic.claude-sonnet-5`) with the same inputs and budgets and default model settings.

A.4.3 PLAN REPRESENTATION AND PROMPTS

The following templates use the structured plan representation in Definition 1. Environment-specific system instructions define the available actions and required fields, delimiters, and nesting. The trained planners receive no demonstrations. Repair feedback and region selection follow Section 4.1.

Generation.

```
System: Return one complete hierarchical
plan for <environment> in the supplied
schema. Each leaf contains an action,
arguments, and expected observation.
User:
Goal: <task goal>
Initial observation: <observation>
Available actions: <action grammar>
Return only the structured plan.
```

Repair.

```
The plan below failed for the same goal.
Selected region: <identifier>; Reason:
<failure-detector explanation>
Expected: <plan prediction>; Observed:
<environment observation>
Plan: <prefix> <editable region> <suffix>
Return only corrected content for the
editable region; preserve all other text.
```

History-based failure detector. In this prompt, "execution trace" denotes execution history, and "Failure feedback" denotes environment feedback.

```
System: Given a goal, structured plan, and execution trace, select the smallest set of
exact step identifiers inconsistent with the observed feedback. Do not repair; abstain
if the trace identifies no defective step.

User: Goal: <task goal>; Current plan: <structured plan with step identifiers>

Execution history: <ordered actions, expected observations, and observed outcomes>;
Failure feedback: <latest environment observation>

Return exactly: REPLAN_NEEDED: <yes or no>; STEP_IDS: <comma-separated step
identifiers; empty if no>; REASON: <brief explanation grounded only in the execution
trace>.
```

Natural Plan judges. We query GPT-5 (`gpt-5`) with medium reasoning effort and Claude Sonnet 5 (`us.anthropic.claude-sonnet-5`) with default model settings, once per candidate, using the following prompt.

```
System: Check the candidate plan against every task requirement step by step. Return
accurate only if it is feasible and satisfies all constraints. Do not use or infer a
reference answer.
User: TASK: <task requirements>; CANDIDATE PLAN: <completed plan>
Return exactly: REASON: <brief explanation>; ACCURATE: <0 or 1>.
```