Temporal Backtracking Search for Test-time Generative Video Reasoning

Sejoon Jun Northeastern University
Zheng Ding Independent Researcher
Chloe H. Su Harvard & Kempner
Weirui Ye[†] MIT
Yilun Du[†] Harvard & Kempner

Abstract

While test-time scaling has revolutionized reasoning in large language models, generative video reasoning remains bottlenecked by a single-shot paradigm. We demonstrate that searching over denoising steps cannot rescue logically flawed rollouts because spatial trajectories commit early in the diffusion process. Root-level Best-of-N (BoN) sampling is similarly inefficient: reasoning errors cluster early in the temporal axis, and resampling blindly discards verified upstream progress. To unlock effective test-time scaling for video models, we introduce **Temporal Backtracking Search (TBS)**, which shifts the search space to the temporal axis. TBS transforms video generation into an iterative generate–verify–restart loop via three core mechanisms: (1) *variable-K conditioning* to resume generation from arbitrary clean prefixes; (2) *temporal process verification* to localize failures and extract valid restart anchors; and (3) *prefix-based search* to reallocate compute toward extending correct trajectories rather than root resampling. Across algorithmic, navigation, and robotics domains, TBS Pareto-dominates matched-budget BoN. In a strict out-of-distribution setting where one-shot generation collapses (0.7% for BoN), TBS achieves 22.7%, with every solved episode stemming from a restarted branch. Ultimately, TBS reveals that the local reasoning competence of video models far exceeds what single-shot rollouts indicate, providing a scalable test-time framework to unlock it.

1. Introduction

Generative video models are evolving beyond visual content creation to emerge as world simulators for sequential decision-making. By extracting action plans from generated rollouts, agents can search, plan, and solve complex tasks entirely in imagination (Du et al., 2023; Bruce et al., 2024; Brooks et al., 2024; Yang et al., 2023; Ha & Schmidhuber, 2018; LeCun et al., 2022). In this reasoning-oriented regime, the relevant axis for scaling test-time compute is no longer aesthetic quality but *trajectory correctness*—whether the generated video depicts a logically valid solution to the given problem.

While recent test-time scaling methods for video generation optimize over noise schedules or partially denoised latents to improve visual fidelity and prompt alignment (Oshima et al., 2025; He et al., 2025; Yang et al., 2025; Cong et al., 2025; Liu et al., 2025; Wu et al., 2026), they fall short for logical decision-making. Consequently, for temporal scaling, the community typically defaults to Best-of-N (BoN) sampling, treating an entire video rollout as an indivisible temporal block.

We identify two empirical bottlenecks that cause these paradigms to fail in long-horizon reasoning. First, decoding intermediate latents reveals that video models commit to a high-level spatial trajectory within the earliest denoising steps (Figure 1); allocating test-time compute to later denoising steps therefore cannot rescue a trajectory that is logically flawed from the start. Second, reasoning failures are not uniformly distributed along the trajectory; they cluster heavily in the early stages of generation. Root-level BoN thus wastes compute by repeatedly resampling the same early failure regions, discarding the valid partial prefixes generated upstream.

[†]Co-advisors.

Corresponding author: Sejoon Jun `jun.se@northeastern.edu`

Accepted to FoGen 2026: Foundations of Deep Generative Models: Understanding Memorization, Generalization, and Reasoning, an ICML 2026 workshop (non-archival).

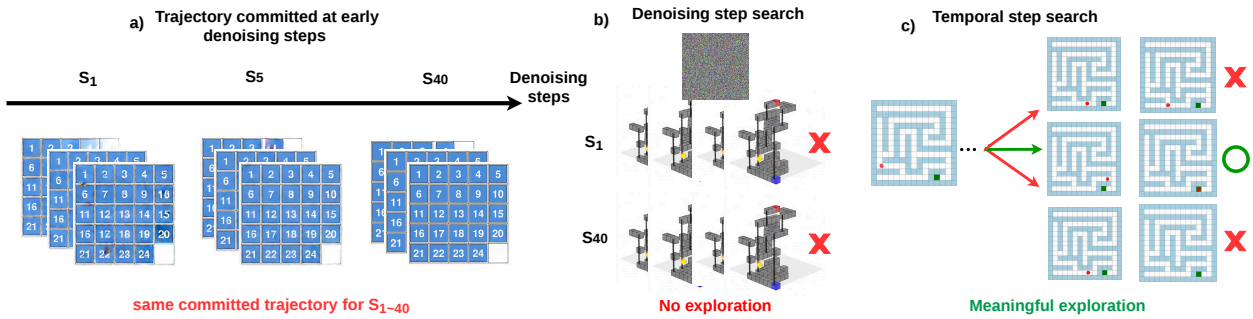


Figure 1. **Trajectories are determined by the earliest denoising steps**. Decoding the same noisy latent at successive denoising steps shows that the motion plan is fixed within the first few steps and persists through to the final rollout. **Takeaway**: Additional compute spent on later denoising steps cannot redirect a wrong trajectory, so test-time search for video reasoning must branch over temporal frames, not over denoising steps.

These observations motivate shifting the search space from the denoising axis to the *temporal* axis. Instead of discarding an entire rollout upon a single mistake, search algorithms should isolate the failure, salvage verified preceding frames, and reallocate compute to repair the invalid suffix. We introduce **Temporal Backtracking Search (TBS)**, transforming single-shot video generation into an iterative generate–verify–restart loop. TBS realizes prefix-based temporal search via three core components: (1) *variable-K conditioning*, a curriculum-trained capability that enables the diffusion model to resume from arbitrary clean prefixes while preserving prior motion context (Chen et al., 2024; Sun et al., 2025; Xie et al., 2025); (2) *temporal process verification*, which identifies the first failure frame to extract verified restart anchors (Lightman et al., 2023; Uesato et al., 2022; Setlur et al., 2024); and (3) *prefix-based search*, which reallocates inference compute to extend valid trajectories rather than resampling from the root (Zhang et al., 2025; Ma et al., 2025).

We evaluate TBS across three regimes spanning the realistic spectrum of process supervision: exact symbolic verification in algorithmic grid-worlds, learned noisy trajectory prediction in continuous navigation (Wang et al., 2025), and simulator-replay verification in robotic manipulation (Chen et al., 2025; Mu et al., 2024). Across all three domains, TBS Pareto-dominates matched-budget BoN. Notably, in a strict out-of-distribution (OOD) setting where one-shot generation collapses (0.7% exact match for BoN), TBS achieves 22.7%, with every solved episode stemming from a restarted child branch. Our primary contributions are:

- Identifying two critical bottlenecks in video test-time scaling: early trajectory commitment renders denoising-step search ineffective, and clustered early failures make root-level BoN grossly inefficient.
- Proposing **Temporal Backtracking Search (TBS)**, which transitions video test-time scaling from monolithic single-shot sampling to prefix-based temporal correction.
- Demonstrating empirically that temporal backtracking unlocks and stitches together the underlying local reasoning competence of video models under symbolic, learned, and physics-based verification.

2. Related Work

Test-time scaling and process verification. Test-time scaling via Best-of-N and tree search has transformed LLM reasoning (Snell et al., 2024; Brown et al., 2024; Feng et al., 2024), driven by the insight that *process* verifiers scoring intermediate steps strictly outperform *outcome* verifiers scoring only final answers (Lightman et al., 2023; Uesato et al., 2022; Setlur et al., 2024). TBS adapts this paradigm to generative video reasoning by shifting evaluation from monolithic rollouts to verified prefixes. Crucially, our process verifier localizes *where* a trajectory deviates, rather than merely judging *whether* the terminal sequence is flawed.

Long-horizon video generation. Diffusion Forcing (Chen et al., 2024) demonstrates causal continuation from clean prefixes as a native sequence diffusion capability, providing the modeling primitive for our variable-K conditioning. While generative stitching methods (Luo et al., 2025; Song et al., 2025) also compose long horizons from local segments, they heavily rely on bidirectional harmonization or global joint resampling. In contrast, TBS performs causal, verifier-localized repair, leaving verified pasts strictly untouched.

Test-time scaling for video generation. Recent methods scale inference compute for video diffusion by searching

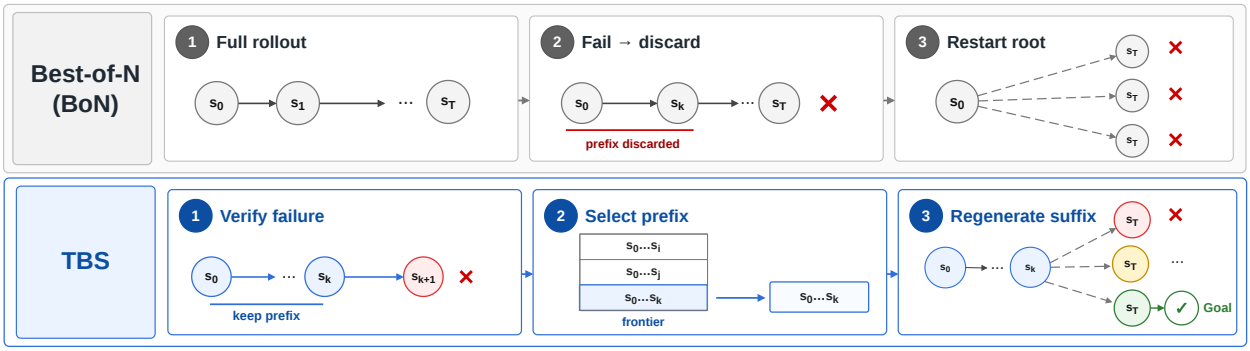


Figure 2. **Mechanism of Temporal Backtracking Search (TBS).** (a) **Single-shot generation:** The trajectory is sampled in one pass; a single error at frame F invalidates the entire remaining suffix. (b) **TBS:** Decomposes long-horizon reasoning into short-horizon sub-problems by iteratively extending verified prefixes. The algorithm executes a generate–verify–restart loop: 1) **Generate** a candidate rollout; 2) **Verify** the trajectory to localize the first failure and cache the valid prefix; and 3) **Restart** generation from the highest-priority anchor.

over denoising trajectories or partially denoised latents (Liu et al., 2025; Oshima et al., 2025; He et al., 2025; Yang et al., 2025; Cong et al., 2025; Wu et al., 2026). However, these approaches operate entirely on the *denoising* axis to optimize visual aesthetics and prompt alignment. TBS fundamentally differs by searching the *temporal* axis to optimize trajectory correctness, leveraging process supervision that ranges from symbolic rules to simulator replay. Furthermore, while concurrent studies (Wang et al., 2026; Wiedemer et al., 2025) diagnose the long-horizon reasoning limits of video models, they offer no constructive mitigation—a critical gap TBS bridges via prefix-based restartable generation.

3. Method

To break the single-shot generative bottleneck, we reformulate video reasoning as inference-time search over the temporal axis. This section formalizes the *conditional reuse gap* that drives our approach (Section 3.1), introduces the **Temporal Backtracking Search (TBS)** algorithm designed to exploit this gap (Section 3.2), and details the variable-K generative engine required to enable temporal branching (Section 3.3).

3.1. Problem Formulation and the Conditional Reuse Gap

Let $\mathbf{v} = (v_1, \ldots, v_T)$ be a video of T frames generated from an initial image v_1 and a text prompt p. For sequential reasoning tasks, we define the logical correctness of a trajectory via a transition validity function:

$$\phi(v_1, \ldots, v_t) = \mathbb{I}[\, v_i \to v_{i+1} \text{ is valid } \forall\, i \leq t{-}1\,]\,. \tag{1}$$

A video achieves an *exact match* (EM $= 1$) if and only if $\phi(\mathbf{v}) = 1$ and the trajectory reaches the goal state by frame T. For invalid rollouts, we identify the earliest logical failure as the *first failure frame*:

$$F(\mathbf{v}) = \min\{t : \phi(v_1, \ldots, v_t) = 0\}, \tag{2}$$

with $F(\mathbf{v}) = T{+}1$ if the video is fully correct. This failure index naturally partitions the rollout into a valid prefix $\mathbf{v}_{1:F-1}$ and an invalid suffix.

Root-level scaling paradigms, such as BoN, indiscriminately discard this verified prefix upon encountering a failure. To illustrate why this is suboptimal, consider a ground-truth trajectory $\mathbf{v}^*$ where a generated sample matches $\mathbf{v}^*$ up to frame ℓ. A root-level generator must synthesize the entire future trajectory from the initial frame, $P_{\text{root}} = p_\theta(\mathbf{v}^*_{2:T} \mid v^*_1, p)$, whereas prefix repair only needs to resolve the remaining suffix, $P_{\text{repair}}(\ell) = p_\theta(\mathbf{v}^*_{\ell+1:T} \mid \mathbf{v}^*_{1:\ell}, p)$.

We define the discrepancy in modeling difficulty between P_{root} and P_{repair} as the **conditional reuse gap**. Because reasoning errors accumulate multiplicatively over time, extending a clean partial sequence is inherently more tractable than generating a long-horizon rollout from scratch. TBS explicitly exploits this gap.

3.2. Temporal Backtracking Search (TBS)

Leveraging the conditional reuse gap, TBS performs state-space search directly over temporal prefixes. The search maintains a **priority frontier** $\mathcal{F}$, a pool of verified prefix anchors awaiting expansion. Each entry in $\mathcal{F}$ is a tuple

Algorithm 1 Temporal Backtracking Search (TBS)

Require: Initial frame v_1, prompt p, budgets (k_1, s, m), Verifier $\mathcal{V}$, Generator $\mathcal{G}$
Initialize priority frontier $\mathcal{F} \leftarrow \emptyset$ {Pool of verified prefix anchors $(K, \mathbf{v}_{1:R}, \pi)$}
Sample k_1 root rollouts from $\mathcal{G}$ conditioned on $(K{=}1, [v_1], p)$
for each generated video $\mathbf{v}$ **do**
  $(\text{EM}, F, R, K, \pi) \leftarrow \mathcal{V}(\mathbf{v})$ {F: first failure frame; R: aligned restart anchor; π: verifier-assigned priority}
  if EM=1 **then**
    return $\mathbf{v}$
  else if $R > 1$ **then**
    $\mathcal{F} \leftarrow \mathcal{F} \cup \{(K, \mathbf{v}_{1:R}, \pi)\}$ {Store verified prefix as a new anchor}
  end if
end for
for expansion step $j = 2, \ldots, m$ **do**
  $n^* \leftarrow \arg\max_{n \in \mathcal{F}} \pi_n$ {Select highest-priority anchor}
  $\mathcal{F} \leftarrow \mathcal{F} \setminus \{n^*\}$
  Sample s child suffixes from $\mathcal{G}$ conditioned on prefix n^*
  Evaluate children with $\mathcal{V}$; insert unsolved valid prefixes into $\mathcal{F}$
  if any child achieves EM=1 **then**
    return solved child video
  end if
end for
return best unsolved video by priority π

$(K, \mathbf{v}_{1:R}, \pi)$, where $\mathbf{v}_{1:R}$ is the verified prefix, $K = R/\tau$ is its corresponding latent length (Section 3.3), and $\pi \in \mathbb{R}$ is a deterministic priority score assigned by the verifier (higher π indicates a more promising anchor for continuation). Instead of repeatedly querying the computationally expensive P_{root}, TBS isolates localized failures and samples continuations strictly from anchors stored in $\mathcal{F}$.

Guided by a process-level verifier $\mathcal{V}$ (Algorithm 1), the search proceeds as follows. For each rollout, $\mathcal{V}$ evaluates the exact-match status and extracts the first failure frame F, an aligned restart anchor $R \leq F{-}1$, and the priority score π. If an invalid rollout contains reusable progress ($R > 1$), the corresponding tuple is inserted into $\mathcal{F}$. At each expansion step, TBS pops the highest-priority anchor from $\mathcal{F}$ and queries the generator for s candidate suffix continuations.

For latent diffusion models, the restart frame must strictly align with the VAE's temporal latent factor τ. The aligned restart anchor is computed as $R(\mathbf{v}) = \tau \cdot \lfloor (F(\mathbf{v}) - 1)/\tau \rfloor$. This guarantees the generator cleanly resumes from exactly $K = R/\tau$ latent frames, preventing temporal overlap or information leakage between the clean prefix and the resampled suffix.

3.3. Variable-K Conditioning

For TBS to function, the generator must support temporal branching: given a prefix verified up to frame R, the model must synthesize the suffix while preserving the exact prefix conditioning. Standard video diffusion models lack this capability, as they denoise all frames jointly from a single-frame initial condition ($K{=}1$).

To unlock temporal branching while keeping per-branch inference cost constant, we finetune the video diffusion model (based on Wan2.2-TI2V-5B(Team, 2025)) with **variable-K conditioning**. Let $\mathbf{z} = \mathcal{E}(\mathbf{v}) \in \mathbb{R}^{C \times T_z \times H_z \times W_z}$ be the VAE-encoded latent representation. During training, we apply a per-frame timestep mask: the first K latent frames remain clean ($t_i = 0$), while the remaining suffix frames are noised at a single sampled timestep t:

$$\tilde{\mathbf{z}}_i = \begin{cases} \mathbf{z}_i & i \leq K, \\ (1-t)\,\mathbf{z}_i + t\,\boldsymbol{\epsilon}_i, \quad \boldsymbol{\epsilon}_i \sim \mathcal{N}(0, \mathbf{I}) & i > K, \end{cases} \tag{3}$$

At inference, K is dictated by the verifier's restart anchor ($K = R/\tau$). We finetune the base video diffusion model with a progressive curriculum that shifts the training distribution from $K{=}1$ to a broad mixture up to $K{=}20$. Without this curriculum, the model collapses when conditioned on longer prefixes (Appendix B.2).

A naive branching alternative is extracting the last verified frame to use as a $K{=}1$ initial condition. However, a single static frame strips away the motion, dynamics, and task momentum established by the preceding trajectory. To isolate the necessity of motion context, we evaluated a controlled subset of 40 provably repairable long-horizon

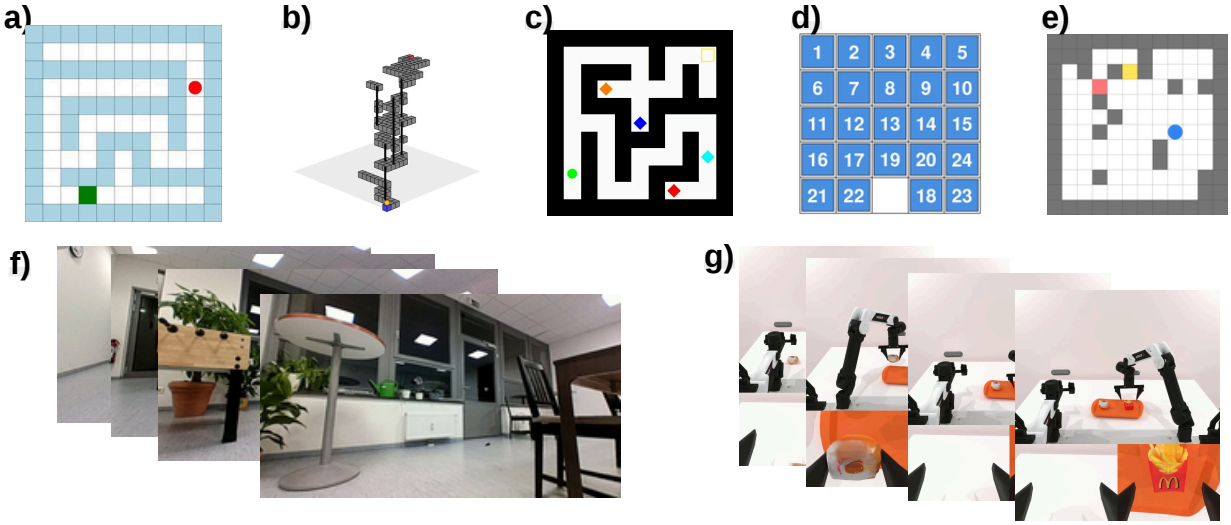


Figure 3. **Evaluation domains.** Rule-based grid-worlds: **2D Maze** (a), **3D Maze** (b), **Keys-and-Doors** (c), **Sliding Puzzle** (d), **Sokoban** (e); continuous navigation **Target-Bench** (f); robotic manipulation **RoboTwin** (g).

Maze trajectories. Replacing the full temporal prefix with a single-frame state reset collapsed the repair success rate from 100% to 2.5% (Appendix B.1). Full prefix conditioning is therefore the foundational engine enabling temporal backtracking.

4. Experiments

4.1. Experimental Setup

We evaluate TBS in three regimes spanning the spectrum of process supervision available in practice: exact symbolic verification in rule-based grid-worlds, learned trajectory prediction in continuous navigation, and simulator-replay verification in robotic manipulation.

Structured video reasoning. We use five rule-based grid-worlds (Figure 3), where deterministic game rules serve as verifiers and identify the first failure frame. The domains target distinct reasoning bottlenecks: *2D Maze* and *3D Maze* for spatial path adherence, *Sliding Puzzle* for semantic correctness, *Sokoban* for irreversible dead states, and *Keys-and-Doors* for long-horizon sub-goal ordering. Each domain has 600 evaluation samples split into easy/medium/hard; *Keys-and-Doors* additionally includes a 300-sample out-of-distribution (OOD) split with extended trajectories (Appendix B.3, B.5).

Continuous navigation. We use Target-Bench (Wang et al., 2025), a mapless path-planning benchmark whose official score (Eq. 4) aggregates five trajectory metrics against a ground-truth path:

$$S = 0.05\, e^{-\mathrm{ADE}} + 0.10\, e^{-\mathrm{FDE}} + 0.10\,(1-\mathrm{MR}/100) + 0.65\,\mathrm{SE}\cdot\mathrm{AC}, \tag{4}$$

where ADE and FDE are the average and final L_2 displacement errors against the GT, MR is the percentage of trajectory points farther than 1 m from the GT, SE and AC are the benchmark's soft-endpoint and approach-consistency scores, and TC is the trajectory-coverage term (full definitions in Appendix C). The verifier is a learned trajectory predictor that maps the current frame and prompt to a future metric-space path, and TBS branches when the decoded video trajectory leaves the predicted spatial corridor.

Robotic manipulation. We evaluate manipulation on RoboTwin (Mu et al., 2024), a bimanual robot benchmark backed by a physics simulator. Generated videos are decoded into 14-D action sequences via an inverse dynamics model and rolled out in the simulator, whose ground-truth task predicate serves as the verifier. RoboTwin trajectories have a natural two-phase structure—an approach phase followed by a final goal-attempt window—which TBS uses to choose restart anchors (Section 4.3).

Baseline and compute budget. Our baseline is Best-of-N (BoN), the standard test-time-scaling method for video generation (Wiedemer et al., 2025; Wang et al., 2026). To isolate the effect of prefix reuse, both methods share identical terminal evaluators and root candidate pools. A TBS configuration uses root width k_1, child samples s, and maximum

Table 1. **Structured video reasoning, exact-match results.** Reported as exact-match percentage with 95% Wilson confidence intervals; intervals are clipped at 0 and 100. Bold marks the better method; ties are not bolded. The TBS–BoN gap widens on *Hard* and *Strict-OOD* splits.

Domain	Split / Metric	Baselines		Budget 10		Budget 20	
		BoN-1	**BoN-5**	**BoN-10**	**TBS-10**	**BoN-20**	**TBS-20**
Maze	Time	1.00×	2.58×	4.01×	3.53×	6.25×	4.42×
	Overall	51.0±4.0	67.7±3.9	75.2±3.6	**84.0±3.1**	79.0±3.4	**90.3±2.6**
	Easy	98.0±2.0	$100.0_{-1.9}$	$100.0_{-1.9}$	$100.0_{-1.9}$	$100.0_{-1.9}$	$100.0_{-1.9}$
	Med.	76.5±6.3	93.0±4.4	96.0±3.7	**99.0±2.6**	97.0±3.4	**99.5±2.3**
	Hard	18.3±4.8	40.0±5.6	53.0±5.7	**68.7±5.5**	60.0±5.6	**81.0±4.8**
Maze3D	Time	1.00×	1.38×	1.59×	1.57×	1.95×	1.83×
	Overall	85.5±3.0	95.3±2.0	96.2±1.9	**97.2±1.7**	96.7±1.8	**97.7±1.6**
	Easy	$100.0_{-1.9}$	$100.0_{-1.9}$	$100.0_{-1.9}$	$100.0_{-1.9}$	$100.0_{-1.9}$	$100.0_{-1.9}$
	Med.	97.0±3.4	99.5±2.3	99.5±2.3	$\mathbf{100.0}_{-1.9}$	99.5±2.3	$\mathbf{100.0}_{-1.9}$
	Hard	73.0±5.3	91.0±3.8	92.7±3.6	**94.3±3.2**	93.7±3.4	**95.3±3.0**
Sokoban	Time	1.00×	2.78×	4.17×	4.19×	6.29×	6.20×
	Overall	41.5±4.0	67.5±3.8	75.5±3.6	**77.7±3.5**	82.0±3.3	**82.7±3.3**
	Easy	78.0±9.1	96.0±5.8	96.0±5.8	**98.0±5.0**	98.0±5.0	**99.0±4.4**
	Med.	30.0±6.7	59.5±6.9	70.0±6.7	**74.5±6.5**	**79.5±6.1**	79.0±6.2
	Hard	37.0±5.6	63.3±5.6	72.3±5.3	**73.0±5.3**	78.3±5.0	**79.7±4.9**
Sliding	Time	1.00×	2.81×	4.27×	4.11×	6.35×	5.67×
	Overall	39.5±4.0	67.5±3.8	74.7±3.7	**80.8±3.3**	82.2±3.3	**86.5±3.0**
	Easy	23.0±9.2	57.0±9.8	65.0±9.7	**69.0±9.6**	**77.0±9.2**	75.0±9.3
	Med.	45.0±6.9	75.5±6.4	80.5±6.0	**86.0±5.5**	86.0±5.5	**89.0±5.1**
	Hard	41.3±5.7	65.7±5.6	74.0±5.2	**81.3±4.8**	81.3±4.8	**88.7±4.1**
Keys and Doors	Time (ID/OOD)	1.00/1.00	2.10/4.95	2.51/9.89	2.59/9.31	2.86/19.66	2.60/18.13
	Overall	52.3±4.0	89.2±2.8	94.5±2.1	**96.5±1.8**	97.7±1.6	**99.2±1.1**
	Easy	72.0±9.5	94.0±6.5	96.0±5.8	**99.0±4.4**	99.0±4.4	$\mathbf{100.0}_{-3.7}$
	Med.	48.5±6.9	85.5±5.5	90.5±4.9	**94.5±4.1**	97.0±3.4	**99.0±2.6**
	Hard (ID)	48.3±5.7	90.0±3.9	96.7±2.7	**97.0±2.6**	97.7±2.4	**99.0±1.9**
	Strict OOD	$0.0_{+1.3}$	$0.0_{+1.3}$	$0.3^{+1.6}_{-0.3}$	**8.3±3.7**	$0.7^{+1.7}_{-0.6}$	**22.7±5.0**

expansions m, giving budget $B = k_1 + s\,(m{-}1)$; we compare against BoN at matched budget $N{=}B$. The *Time* columns in Table 1 report normalized wall-clock cost, which grows sub-linearly due to early termination after the first verified solve.

4.2. Main Results

Structured reasoning: the gap widens with horizon length. Across all five rule-based domains (Table 1), TBS matches or exceeds matched-budget BoN on every *Overall* score. The two domains with the longest path distributions show the largest improvements: comparing TBS-20 against BoN-20, Hard-split EM rises from 60.0% to 81.0% on 2D Maze and from 81.3% to 88.7% on Sliding Puzzle. On a few short-horizon splits (e.g., Sokoban Med., Sliding Easy), TBS-20 and BoN-20 fall within each other's confidence intervals, consistent with TBS targeting long-horizon failures rather than uniformly improving every problem. The clearest separation is on Keys-and-Doors *Strict-OOD*: BoN-20 reaches only 0.7% EM while TBS-20 reaches 22.7% (+22.0 absolute points), a gap that Section 4.3 traces to restarted child branches. Appendix Table 12 verifies the mechanism directly: on this split, $0/68$ TBS solves were found by depth-0 root rollouts, while all $68/68$ solved episodes came from depth-≥ 1 restarted branches.

Continuous navigation: TBS improves all trajectory metrics under noisy supervision. Target-Bench (Table 2) tests whether TBS still helps when the verifier is a learned and noisy trajectory predictor. TBS improves all five spatial metrics at both matched budgets, with the largest gains on trajectory precision: at budget 5, TBS reduces Miss Rate from 2.70% to 1.56% and raises Approach Consistency from 0.872 to 0.936. This suggests that even a noisy process signal can localize where the rollout drifts.

Robotic manipulation: prefix reuse helps in a physics simulator. On RoboTwin's clean scenes (Figure 4a), matched-budget TBS improves the simulator-evaluated success rate from 46% (BoN-3) to 50% (TBS-3), and from 49% (BoN-5) to 53% (TBS-5). The consistent +4% gain across both budgets indicates that the same prefix-reuse mechanism that

Table 2. **Continuous navigation on Target-Bench.** TBS improves all five trajectory metrics at both matched budgets, with the largest gains on miss rate (MR) and approach consistency (AC). *Overall* is the official weighted score.

Method	ADE↓	FDE↓	MR (%)↓	SE↑	AC↑	Overall↑
BoN-1	0.541	1.074	16.70	0.268	0.730	0.323
BoN-3	0.300	0.765	4.05	0.407	0.817	0.424
TBS-3	**0.276**	**0.640**	**1.73**	**0.496**	**0.936**	**0.508**
BoN-5	0.265	0.680	2.70	0.466	0.872	0.473
TBS-5	**0.263**	**0.622**	**1.56**	**0.508**	**0.936**	**0.511**

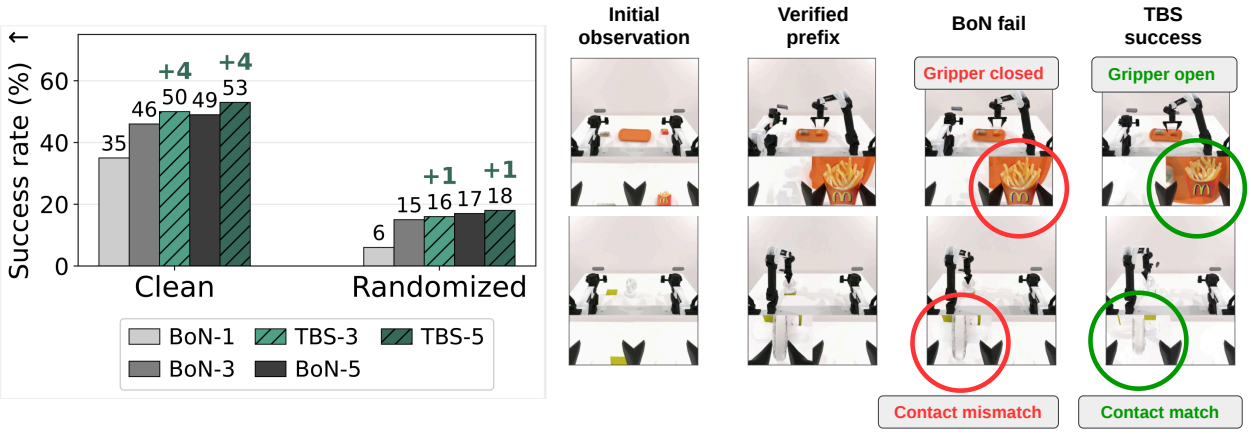


Figure 4. **RoboTwin Results.** (a) Matched-budget success rates: TBS adds $+4\%$ at every budget in clean scenes but only $+1\%$ under visual randomization. (b) Repair sequence: TBS reuses the verified pre-grasp prefix from a failed root attempt and regenerates only the final manipulation window.

helps on grid-worlds and continuous navigation also transfers to a closed-loop physics evaluator. We return to the visually randomized OOD setting in Section 4.5.

4.3. Mechanism: Prefix Reuse Targets Early Failure Bottlenecks

Across all three regimes, rollout failures concentrate in a few short windows along the trajectory; the rest of the trajectory is locally easy (Figure 5). Independent root sampling has to replay these windows for every new candidate; prefix reuse keeps the verified portion and spends the budget on the residual suffix.

Failures cluster early in structured domains. On 2D Maze, 72.7% of failed BoN-20 rollouts have their first failure in the first 0–20% of the trajectory; on Keys-and-Doors OOD this rises to 78.3% (Figure 5c). Each fresh root has to replay the same early region, so BoN spends most of its budget re-failing at the same bottleneck. Once paths exceed 40 steps, BoN-20 EM drops to 17.1% while TBS-20 stays at 60.0% (Figure 5a). The same pattern explains the strict-OOD result in Section 4.2: every TBS-20 solve came from a restarted child branch (Appendix B.9).

Restart anchors preserve verified progress in manipulation. RoboTwin shows the same idea via a different anchor. When the base video-to-action policy approaches the target correctly but fails the final grasp, TBS uses the gripper-opening boundary as the restart anchor (Figure 4b). This preserves the pre-grasp motion (often several seconds of video) and concentrates the additional compute on regenerating the failed manipulation window.

4.4. Ablation Studies

We ablate three design choices that drive the gains in Section 4.2: the variable-K prefix conditioning interface, the depth-versus-width allocation of the search budget, and the per-domain verifier signal. Prefix conditioning and verifier quality are each individually necessary, while the depth/width allocation must be tuned per domain. Appendix B.13 compares the TBS frontier against Greedy, Beam-2, and Random prefix-reuse policies under the same matched search shapes.

Variable-K prefix conditioning is required, not just nice-to-have. TBS resumes generation from a verified prefix by

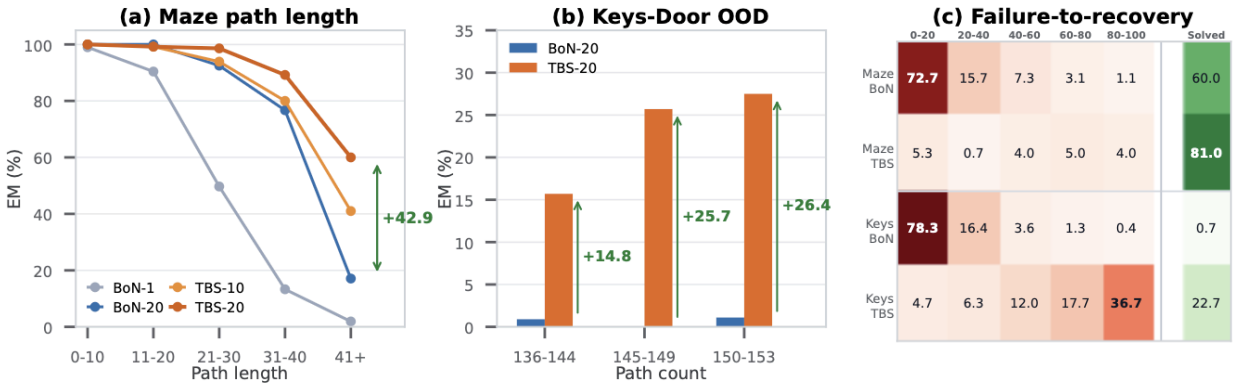


Figure 5. **BoN degrades on long horizons; failures cluster at the start of trajectories. (a) 2D Maze:** TBS sustains EM as path length grows, while BoN degrades sharply past 40 steps. **(b) Keys-and-Doors (OOD):** TBS extends to longer paths than seen during training; BoN remains near zero. **(c) Failure-to-recovery distribution:** BoN failures concentrate in the first 0–20% of trajectories, where independent root sampling repeatedly re-fails.

encoding it as clean prefix latents that the diffusion model conditions on. A natural alternative is to discard the prefix and restart from a single frame (K=1) at the verifier's restart anchor. We construct 40 repairable parent prefixes per domain, each of which led to a solved child continuation in a prior TBS run, and re-present the same prefix to both interfaces under matched generation budget. Repair collapses from 100% to 2.5% on Maze and from 100% to 5.0% on Sliding when the prefix is replaced by a single-frame state reset (Appendix B.1), showing that the motion content carried by the verified prefix is not redundant with the visual content of its last frame.

Search shape adapts to the failure mode. At a fixed budget of B=20, we sweep the search shape (k_1, s, m) and report the best EM per root width k_1 (Appendix B.11). 2D Maze is depth-friendly: narrow-deep search (k_1=2) reaches 90.3% and wider variants fall within 0.5 pp, because failures are late spatial drift on a structurally correct prefix that restart can repair. Sokoban is width-friendly: k_1=2 stalls at 76.5% while k_1=8 reaches 82.7% (+6.2 pp), because Sokoban prefixes can be visually plausible but already strategically dead, so deeper restart from a single noisy root inherits the corruption. The headline TBS-20 rows in Table 1 use the best shape per domain identified by this sweep, suggesting a simple rule of thumb: prefer depth when failures are localized at the suffix, and width when failures contaminate the prefix.

Verifier quality matters in the harder structured domains. On Sokoban and Sliding Puzzle, the verifiers expose more than just prefix length. The Sokoban verifier reports BFS dead-state detection and remaining-moves count, so the priority can rank a strategically dead prefix below a shorter but live one. The Sliding Puzzle verifier flags any move whose optimal continuation is to undo it, treating the move itself as the failure rather than waiting for a downstream syntactic error. At fixed search shape, the BFS-aware Sokoban verifier improves TBS by +4.2 to +10.2 pp over a prefix-only verifier; the undo-aware Sliding verifier improves TBS by +2.2 to +4.3 pp (full sweeps in Appendix B.12). Neither signal helps BoN at matched budget—BoN never restarts, so a better restart anchor has nothing to act on—and the asymmetry rules out a generic "better verifier" explanation: the gain comes from where TBS uses the verifier, not from the verifier alone.

4.5. Operating Envelope: When Prefix Reuse Helps and When It Stops

The mechanism in Section 4.3 depends on two conditions: (i) a verifier that approximately localizes *where* a trajectory deviates, and (ii) a base model that produces a repairable prefix at all. We test each condition on the regime that strains it most.

TBS tolerates noisy process supervision. The Target-Bench verifier is the only one in our setup whose underlying signal is itself noisy, so we use it to stress-test condition (i). Figure 6 shows the qualitative behavior: TBS interrupts a rollout the moment it leaves the predicted corridor, before the drift becomes irrecoverable. To quantify this, we replace the predictor's output with a ground-truth path corrupted by Gaussian noise of increasing magnitude $q \in \{1.0, 0.7, 0.5, 0.3\}$, yielding mean ADE perturbations of approximately 0, 0.3, 0.5, and 0.7 m. As shown in Figure 7, TBS matches or exceeds BoN at every noise level. Under the strongest perturbation, TBS and BoN converge at budget 3 ($\Delta = +0.001$ overall score), and at budget 5 TBS is still ahead by $\Delta = +0.011$. The mechanism does not need a flawless verifier; it falls back to standard root sampling as the signal weakens.

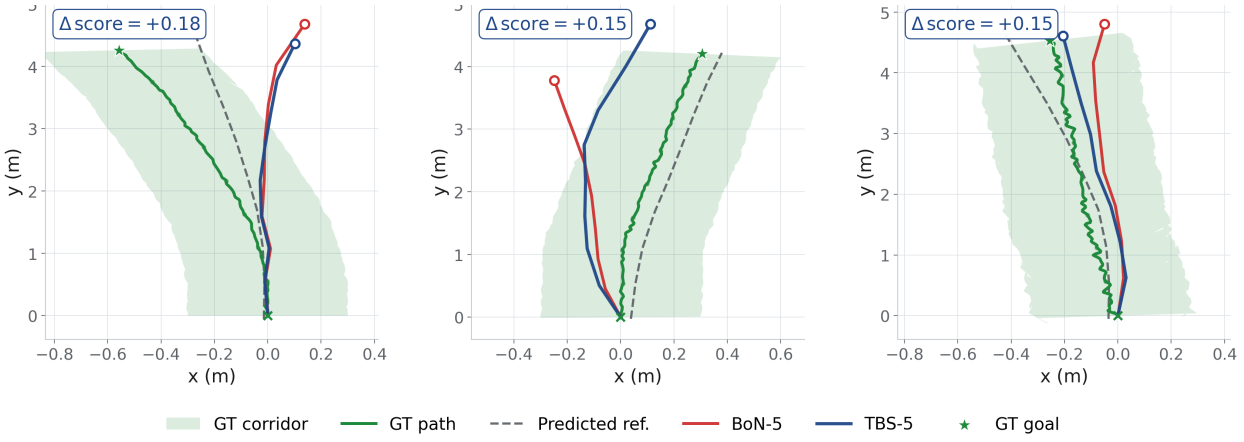


Figure 6. **Trajectory refinement via temporal branching.** BoN re-ranks completed roots only (red); TBS extracts a verified prefix and resumes generation before the rollout drifts from the predicted corridor (blue), staying closer to the GT path.

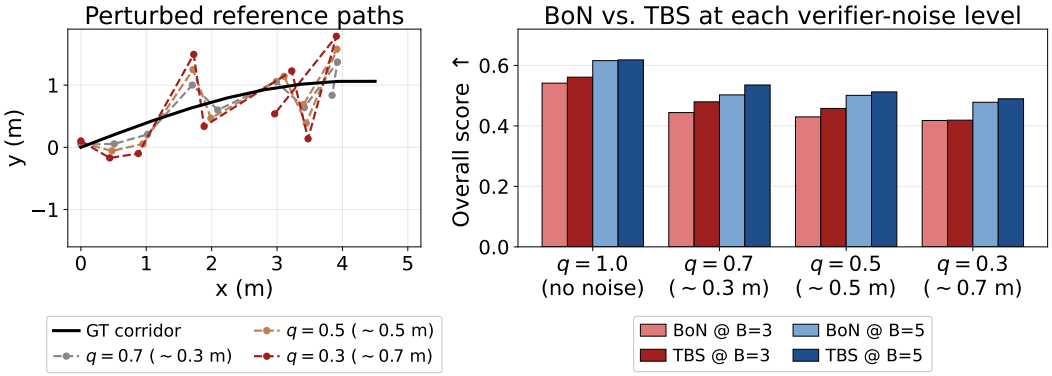


Figure 7. **Robustness to verifier noise.** Target-Bench overall scores under increasing Gaussian perturbation q to the reference path. TBS matches or exceeds BoN at every noise level, and converges to BoN under the strongest perturbation.

TBS cannot supply competence the base model lacks. Condition (ii) is tested on RoboTwin's visually randomized OOD setting. Under heavy visual randomization the base video-to-action policy rarely reaches the target, so TBS has little to repair: BoN-3 succeeds on 15% of episodes and TBS-3 on 16% (Figure 4a). When the base policy works, TBS adds $+4\%$ to BoN at every budget; when it does not, the gain shrinks to $+1\%$. Prefix reuse amplifies competence the base model already has; it does not supply competence the base model fails to produce.

In summary, across the three regimes, TBS's gains grow with horizon length (Section 4.3) and persist under verifier noise; but they inherit the limits of the underlying generator. Prefix reuse moves compute past the bottlenecks where the generator predictably fails; it does not change what the generator can locally produce.

5. Discussion

This work shows that single-shot video generation bottlenecks long-horizon reasoning, and proposes Temporal Backtracking Search (TBS) to shift test-time scaling to the temporal axis. By caching verified partial progress and reallocating compute to repair localized errors, TBS unlocks stronger reasoning across symbolic, learned, and physics-based environments.

Limitations and Future Work. While TBS significantly improves long-horizon performance, it still depends on the base video model's local competence: the model must generate at least a short valid prefix before search can repair later errors, which may fail under extreme OOD visual conditions. Our current implementation also relies on domain-specific external verifiers; future work could internalize this supervision by training implicit video process reward models (PRMs), enabling autonomous temporal search across broader domains without external simulators.

References

Brooks, T., Peebles, B., Holmes, C., DePue, W., Guo, Y., Jing, L., Schnurr, D., Taylor, J., Luhman, T., Luhman, E., et al. Video generation models as world simulators. *OpenAI Blog*, 1(8):1, 2024.

Brown, B. et al. Large language monkeys: Scaling inference compute with repeated sampling. 2024.

Bruce, J., Dennis, M. D., Edwards, A., Parker-Holder, J., Shi, Y., Hughes, E., Lai, M., Mavalankar, A., Steigerwald, R., Apps, C., et al. Genie: Generative interactive environments. In *Forty-first International Conference on Machine Learning*, 2024.

Chen, B., Martí Monsó, D., Du, Y., Simchowitz, M., Tedrake, R., and Sitzmann, V. Diffusion forcing: Next-token prediction meets full-sequence diffusion. *Advances in Neural Information Processing Systems*, 37:24081–24125, 2024.

Chen, T., Chen, Z., Chen, B., Cai, Z., Liu, Y., Li, Z., Liang, Q., Lin, X., Ge, Y., Gu, Z., et al. Robotwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. *arXiv preprint arXiv:2506.18088*, 2025.

Cong, W., Zhu, H., Wang, P., Liu, B., Xu, D., Wang, K., Pan, D. Z., Wang, Y., Fan, Z., and Wang, Z. Can test-time scaling improve world foundation model? *arXiv preprint arXiv:2503.24320*, 2025.

Du, Y., Yang, S., Dai, B., Dai, H., Nachum, O., Tenenbaum, J., Schuurmans, D., and Abbeel, P. Learning universal policies via text-guided video generation. *Advances in neural information processing systems*, 36:9156–9172, 2023.

Feng, X. et al. Alphazero-like tree-search can guide large language model decoding and training. 2024.

Feng, Y., Tan, H., Mao, X., Liu, G., Huang, S., Xiang, C., Su, H., and Zhu, J. Vidar: Embodied video diffusion model for generalist bimanual manipulation. *arXiv preprint arXiv:2507.12898*, 2025.

Ha, D. and Schmidhuber, J. World models. *arXiv preprint arXiv:1803.10122*, 2(3):440, 2018.

He, H., Liang, J., et al. Evosearch: Scaling image and video generation via test-time evolutionary search. *arXiv preprint arXiv:2505.17618*, 2025.

LeCun, Y. et al. A path towards autonomous machine intelligence version 0.9. 2, 2022-06-27. *Open Review*, 62(1): 1–62, 2022.

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let's verify step by step. In *The twelfth international conference on learning representations*, 2023.

Liu, F., Wang, H., Cai, Y., Zhang, K., Zhan, X., and Duan, Y. Video-t1: Test-time scaling for video generation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 18671–18681, 2025.

Luo, Y., Mishra, U. A., Du, Y., and Xu, D. Generative trajectory stitching through diffusion composition. *arXiv preprint arXiv:2503.05153*, 2025.

Ma, N., Tong, S., Jia, H., Hu, H., Su, Y.-C., Zhang, M., Yang, X., Li, Y., Jaakkola, T., Jia, X., et al. Inference-time scaling for diffusion models beyond scaling denoising steps. *arXiv preprint arXiv:2501.09732*, 2025.

Mu, Y., Chen, T., Peng, S., Chen, Z., Gao, Z., Zou, Y., Lin, L., Xie, Z., and Luo, P. Robotwin: Dual-arm robot benchmark with generative digital twins (early version). In *European Conference on Computer Vision*, pp. 264–273. Springer, 2024.

Oshima, Y., Suzuki, M., Matsuo, Y., and Furuta, H. Inference-time text-to-video alignment with diffusion latent beam search. *arXiv preprint arXiv:2501.19252*, 2025.

Setlur, A., Nagpal, C., Fisch, A., Geng, X., Eisenstein, J., Agarwal, R., Agarwal, A., Berant, J., and Kumar, A. Rewarding progress: Scaling automated process verifiers for llm reasoning. *arXiv preprint arXiv:2410.08146*, 2024.

Snell, C., Lee, J., Xu, K., and Kumar, A. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Song, C., Stary, M., Chen, B., Kopanas, G., and Sitzmann, V. Generative view stitching. *arXiv preprint arXiv:2510.24718*, 2025.

Sun, M., Wang, W., Li, G., Liu, J., Sun, J., Feng, W., Lao, S., Zhou, S., He, Q., and Liu, J. Ar-diffusion: Asynchronous video generation with auto-regressive diffusion. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 7364–7373, 2025.

Team, W. Wan: Open and advanced large-scale video generative models. 2025.

Uesato, J., Kushman, N., Kumar, R., Song, F., Siegel, N., Wang, L., Creswell, A., Irving, G., and Higgins, I. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022.

Wang, D., Ye, H., Liang, Z., Sun, Z., Lu, Z., Zhang, Y., Zhao, Y., Gao, Y., Seegert, M., Schäfer, F., et al. Target-bench: Can world models achieve mapless path planning with semantic targets? *arXiv preprint arXiv:2511.17792*, 2025.

Wang, R., Cai, Z., Pu, F., Xu, J., Yin, W., et al. Demystifying video reasoning. *arXiv preprint arXiv:2603.16870*, 2026.

Wiedemer, T., Li, Y., Vicol, P., Gu, S. S., Matarese, N., Swersky, K., Kim, B., Jaini, P., and Geirhos, R. Video models are zero-shot learners and reasoners. *arXiv preprint arXiv:2509.20328*, 2025.

Wu, M., Zhu, J., Feng, X., Chen, C., Zhu, C., Song, B., Mao, F., Wu, J., Chu, X., and Huang, K. Imagerysearch: Adaptive test-time search for video generation beyond semantic dependency constraints. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 10700–10708, 2026.

Xie, D., Xu, Z., Hong, Y., Tan, H., Liu, D., Liu, F., Kaufman, A., and Zhou, Y. Progressive autoregressive video diffusion models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 6322–6332, 2025.

Yang, H., Tang, F., Hu, M., Yin, Q., Li, Y., Liu, Y., Peng, Z., Gao, P., He, J., Ge, Z., et al. Scalingnoise: Scaling inference-time search for generating infinite videos. *arXiv preprint arXiv:2503.16400*, 2025.

Yang, S., Nachum, O., Du, Y., Wei, J., Abbeel, P., and Schuurmans, D. Foundation models for decision making: Problems, methods, and opportunities. *arXiv preprint arXiv:2303.04129*, 2023.

Zhang, X., Lin, H., Ye, H., Zou, J., Ma, J., Liang, Y., and Du, Y. Inference-time scaling of diffusion models through classical search. *arXiv preprint arXiv:2505.23614*, 2025.

Appendix Contents

A Terminology and Claim Boundaries **12**

B Structured Video Reasoning Details **13**

B.1 Prefix-Conditioning Ablation . . . 13

B.2 Training and Inference Details . . . 13

B.3 Structured Video Reasoning Domains . . . 14

B.4 Prompt Templates . . . 15

B.5 Verifier and Restart Details . . . 16

B.6 TBS Search Budgets and Priority . . . 17

B.7 Structured Video Reasoning Extended Results . . . 18

B.8 Failure Clustering Statistics . . . 19

B.9 Root Solves vs. Restarted-Branch Solves . . . 19

B.10 Root-Width Sensitivity at Matched Budget . . . 20

B.11 Depth–Width Search Shape . . . 21

B.12 Verifier-Quality Ablations: Sokoban BFS-Aware and Sliding Undo-Required . . . 21

B.13 Search-Policy Ablation: Greedy, Beam-2, and Random Frontiers . . . 23

C Target-Bench Navigation Details **24**

C.1 Official benchmark score and metric definitions . . . 24

C.2 Predictor details . . . 24

D RoboTwin Manipulation Details **25**

D.1 Pipeline . . . 25

D.2 First-goal-attempt window . . . 25

D.3 The 10-task progress-aware subset . . . 25

E Limitations and Failure Cases **26**

F Visual Examples **27**

A. Terminology and Claim Boundaries

Search policy. TBS is implemented as prefix-priority best-first search over a global frontier of verified prefixes. It is not UCT-style Monte Carlo tree search. The paper claims that prefix reuse with frontier search improves long-horizon generative video reasoning. It does not claim that the current search policy is a full symbolic planner.

Verifier scope. The structured-domain verifiers are domain-specific process verifiers. They can localize the first checked failure frame because the tasks are grid-world or grid-like rule-based problems. They do not provide the correct continuation. TBS receives a verified prefix, restart anchor, and priority score. The generator must still synthesize the suffix.

B. Structured Video Reasoning Details

This appendix collects the structured video-reasoning details: prefix-conditioning ablation, training settings, domain definitions, prompts, symbolic verifiers, search budgets, extended results, failure clustering, root-versus-child solves, and depth–width search behavior. It also includes verifier-quality ablations and search-policy ablations.

B.1. Prefix-Conditioning Ablation

TBS presents the verified prefix to the generator as clean prefix latents that the diffusion model conditions on while denoising the regenerated suffix. We compare two policies that branch from the same verified parent prefix under matched generation budget:

- `full_prefix` (TBS default): the verified prefix frames are encoded as prefix latents with their actual motion content.
- `last_frame_reset`: only the last verified frame is encoded as a single $K{=}1$ first-frame condition. Prefix-length conditioning and motion content are both discarded. This is the naive single-frame state reset.

Subset construction. The 40-sample evaluation subset per domain is constructed as follows. From a prior `full_prefix` TBS run, we select samples where a child branch at depth ≥ 1 achieved EM, then trace back to the failed parent record from which that child was branched. We retain only samples where the parent prefix is at least 5 pixel frames long at a multi-frame conditioning boundary, and take the first 40 such samples in dataset order. The same parent prefix is then re-presented to the generator under each conditioning policy at matched generation budget.

By construction, `full_prefix` extends the parent prefix to a solved trajectory in 100% of cases (the manifest is filtered to "parent prefixes that `full_prefix` provably repairs"). The ablation therefore does not measure population-level success rates; it measures how much of `full_prefix`'s repair capability the simpler conditioning policy can match given the identical parent prefix and identical budget.

Task	Policy	EM	SR (any goal)
Maze	`last_frame_reset`	$1/40 = 2.5\%$	$2/40 = 5.0\%$
Maze	`full_prefix` (TBS)	$40/40 = 100\%$	$40/40 = 100\%$
Sliding	`last_frame_reset`	$2/40 = 5.0\%$	$2/40 = 5.0\%$
Sliding	`full_prefix` (TBS)	$40/40 = 100\%$	$40/40 = 100\%$

Table 3. Prefix-conditioning ablation on known-repairable parent prefixes. `full_prefix` is 100% by construction. Replacing the full prefix with a single last-frame condition collapses recovery to 2.5% on Maze and 5.0% on Sliding.

B.2. Training and Inference Details

All TBS results use a single base video diffusion model (Wan2.2-TI2V-5B (Team, 2025)) fine-tuned with domain-specific LoRA adapters that support variable-K prefix conditioning. The base model weights are frozen throughout; only LoRA parameters are updated. A separate LoRA adapter is trained per domain. All training is performed on pre-encoded VAE latent caches.

B.2.1. LoRA HYPERPARAMETERS

Hyperparameter	Value
Base model	Wan2.2-TI2V-5B
LoRA rank / alpha	$32/32$
Optimizer	AdamW
Learning rate	2×10^{-4}
Weight decay	1×10^{-4}
Batch size / grad accumulation	$1/2$
Epochs	100
Dataset repeat	2
Resolution	512×512
Frame count	81 (161 for strict-OOD Keys-Doors and 24-grid 3D Maze)
Saved FPS	24
LR scheduler	constant (no warmup)
Inference denoising steps	50
CFG scale	5.0
Sigma shift	5.0
VAE compression τ	4 (each latent frame $\equiv$ 4 pixel frames)
Seed schedule	base seed 0; sample seed = base + sample_index

Table 4. LoRA fine-tuning and inference hyperparameters shared across all domains.

B.2.2. VARIABLE-K CONDITIONING CURRICULUM

The model is trained with a curriculum over prefix length K that progressively increases the maximum allowed prefix and shifts probability mass toward longer prefixes. Prefix lengths are sampled from four buckets: $K{\in}\{1\}$, $K{\in}\{2{-}4\}$, $K{\in}\{5{-}8\}$, $K{\in}\{9{+}\}$. Without this curriculum the model fails to learn long-prefix conditioning and collapses on $K{>}5$ at inference.

Training progress	Max K	Bucket weights $(K{=}1,\ 2{-}4,\ 5{-}8,\ 9{+})$
0–10%	1	$(1.00, 0.00, 0.00, 0.00)$
10–30%	5	$(0.50, 0.30, 0.20, 0.00)$
30–55%	10	$(0.40, 0.30, 0.20, 0.10)$
55–80%	15	$(0.35, 0.30, 0.20, 0.15)$
80–100%	20	$(0.35, 0.25, 0.20, 0.20)$

Table 5. Variable-K training curriculum.

B.2.3. PER-FRAME TIMESTEP MASKING

During training, the first K latent frames receive timestep $t{=}0$ (clean signal, $\sigma_{t_i}{=}0$), and the remaining suffix frames receive a single sampled timestep t. This per-frame timestep mask teaches the model to treat the first K frames as given context and regenerate only the continuation. Equation 3 in the main text gives the formal rectified-flow forward process.

B.3. Structured Video Reasoning Domains

We evaluate on five rule-based domains. Each is a 2D top-down or 3D isometric rendering of a sequential decision problem, governed by deterministic rules so that an exact symbolic verifier can return both terminal success (EM) and process-level failure information. Test sets contain 600 samples per domain split as 100 easy / 200 medium / 300 hard, plus a 300-sample strict-OOD split for Keys-and-Doors.

2D Maze. Grid sizes $7{\times}7$ (easy), $11{\times}11$ (medium), $15{\times}15$ (hard). A red agent must traverse the white-corridor maze to a green goal without entering the light-blue walls. Solution path lengths range from 2 to 78 steps.

3D Maze. First-person rendered isometric mazes with stacked paths and ladders. The agent (yellow ball) must climb to a red goal cube. Path-following structure mirrors 2D Maze, but visual tracking is harder.

Sliding Puzzle. Grid sizes 3×3 (easy), 4×4 (medium), 5×5 (hard). The agent must arrange numbered tiles in row-major order with the blank in the bottom-right. The goal arrangement is not visible in the conditioning image, so failures are semantic rather than visual wall-crossings. Tiles are rendered in large font for OCR readability; the easy split is paradoxically harder because 3×3 has smaller tiles.

Sokoban. Board sizes 5×5 / 8×8 / 12×12 with wall probabilities $0.08/0.12/0.16$. Single box, single target. The blue agent can push but not pull; a visually valid prefix can already be in a dead state where no solution exists.

Keys-and-Doors. The agent must collect colored keys in the correct sequence before reaching the door. Adds long-horizon sub-goal ordering on top of path-following. The strict-OOD split uses 161-frame videos with path lengths exceeding the training distribution (typically 136–153 path cells); root sampling on this split nearly collapses (BoN-20 EM 0.7%).

B.3.1. TRAIN / TEST SPLITS

Domain	Easy	Medium	Hard
2D Maze	500	500	500
3D Maze	500	500	500
Sokoban	500	500	500
Sliding Puzzle	500	500	500
Keys-and-Doors	500	500	500
Test (all domains)	100	200	300
Keys-OOD strict split	300 samples, 161-frame videos		

Table 6. Dataset sizes per domain. All domains use 1500 training samples balanced across difficulty. The Keys-and-Doors strict-OOD split uses path lengths that exceed the training distribution.

B.4. Prompt Templates

All rich prompts share a four-block structure: object identities and colors; movement constraints; scene invariance; camera constraints. The prompts are uniform across difficulty levels within each domain—difficulty is determined by the input image, not by the text.

B.4.1. 2D MAZE

Create a 2D animation based on the provided image of a maze. The red circle slides smoothly along the white square path, stopping perfectly on the green square. The red circle never slides or crosses into the light blue square areas of the maze. The camera is a static, top-down view showing the entire maze.

Maze: The maze paths are white square, the walls are light blue square. The red circle moves to the goal position, represented by green square. The red circle slides smoothly along the white square path. The red circle stops perfectly on the green square.

Scene: No change in scene composition. No change in the layout of the maze. The red circle travels along the white square path without speeding up or slowing down.

Camera: Static camera. No zoom. No pan. No glitches, noise, or artifacts.

B.4.2. 3D MAZE

Create a 3D animation based on the provided image of a cube maze. A yellow ball slides smoothly along the gray cube pathway, climbs up the vertical ladders step by step, and finally stops perfectly on the red cube at the top. The yellow ball never touches or passes through the blue cube or any non-gray areas of the maze. The camera remains static in an isometric, top-down angle showing the entire structure.

Maze: The maze consists of stacked transparent gray cubes forming a 3D pathway. The red cube represents the goal position. The blue cube marks the starting platform where the yellow ball begins. The yellow ball moves upward along the gray path, climbing vertically via the ladders. The ball slides smoothly without sudden changes in direction or speed. The ball stops exactly on top of the red cube at the end.

Scene: No structural or color changes during animation. The maze layout and cube arrangement remain unchanged. The yellow ball moves continuously at a constant speed along the 3D path.

Camera: Static camera. No zoom. No pan. No glitches, noise, or artifacts.

B.4.3. SOKOBAN

Create a 2D animation based on the provided image of a grid puzzle. The blue ball moves into position behind the yellow square and smoothly pushes it toward the red goal square. The yellow square only slides when pushed from behind by the blue ball and moves in a straight line along the white floor tiles. When the direction of the yellow square's movement needs to change, the blue ball must reposition itself to a new side of the yellow square. The yellow square never crosses or overlaps any gray walls. The camera is a static, top-down view showing the entire puzzle.

Maze: The floor area is white, and the walls are gray. The yellow square can only move when pushed by the blue ball from behind. The blue ball cannot pull the yellow square or move through walls. The yellow square slides smoothly in one direction until it reaches the red goal square. The animation stops perfectly when the yellow square aligns with the red goal square.

Scene: No change in scene composition. No change in the layout of the puzzle. The movement is smooth, with no speed variation.

Camera: Static camera. No zoom. No pan. No glitches, noise, or artifacts.

B.4.4. SLIDING PUZZLE

Complete this sliding puzzle by arranging the numbered tiles in sequential order. The goal state has tiles numbered left-to-right, top-to-bottom, with the blank space at the bottom-right corner.

Puzzle: The grid contains numbered tiles and one blank space (white cell). A tile can only move if it is directly adjacent (up, down, left, or right) to the blank space. Each move slides exactly one adjacent tile into the blank space; the tile and blank space swap positions. Only one tile moves per step; all other tiles remain stationary during each move. The number displayed on each tile must remain the same throughout the entire animation; tiles must not change, swap, or lose their numbers at any point. The sequence of moves must follow a shortest valid solution path from the initial state to the goal state; do not make unnecessary detours or extra moves.

Scene: No change in scene composition. No change in the grid layout, tile sizes, or tile colors. Tiles slide smoothly without speeding up or slowing down. The background and grid structure remain unchanged throughout.

Camera: Static camera. No zoom. No pan. No glitches, noise, or artifacts.

B.4.5. KEYS-AND-DOORS

Create a 2D animation based on the provided image of a maze. The green circle is the agent. There are multiple colored diamond-shaped keys and one hollow rectangle door in the maze. The green circle slides smoothly along the white paths, using the shortest valid paths to collect all visible keys in the order that minimizes the total travel distance, then moves to the door. Each key disappears when the green circle reaches it. The green circle never crosses or overlaps any black wall areas of the maze. The camera is a static, top-down view showing the entire maze.

Maze: The maze paths are white, and the walls are black. The green circle must collect all visible keys before reaching the door. The green circle slides smoothly along the white paths. The green circle never slides or crosses into the black wall areas. The green circle stops perfectly on the door after collecting all keys.

Scene: No change in scene composition. No change in the layout of the maze. The movement is smooth, with no speed variation.

Camera: Static camera. No zoom. No pan. No glitches, noise, or artifacts.

B.5. Verifier and Restart Details

For every structured video-reasoning domain, the verifier returns a process trace (`first_failure_frame`, `restart_frame`, `prefix_pixel_frames`). The verifier exposes a failure location and a restartable prefix; it does not provide the correct continuation.

B.5.1. 2D MAZE AND 3D MAZE

The 2D Maze verifier extracts the player sprite template from the first frame using its known starting bounding box, then locates the player at each subsequent frame via normalized cross-correlation. Pixel coordinates are mapped to grid cells using the known maze geometry, and each cell transition is checked against the maze graph, wall layout, and task-derived goal/progress constraints. The verifier does not receive or return the future solution trajectory. The first failure frame is the earliest of motion / tracking / appearance / duplicate-player / wall-crossing / length failures. The restart frame is backed up to the entry point of the last correctly visited cell, then aligned to the VAE's latent frame boundary. The 3D Maze verifier uses the same logic on first-person renderings and additional blob diagnostics for ladder identity swaps.

B.5.2. SOKOBAN

The Sokoban verifier tracks both the player and the box via template matching, maps centers to grid cells, and validates wall collisions and legal push transitions. It additionally runs a single-box BFS state check for remaining moves and dead-state detection: `bfs_initial_moves` (BFS shortest path from initial state), `bfs_remaining_moves` (BFS from current state), `bfs_dead_state` (true if the box is cornered and no solution exists), and `bfs_progress` $= 1 - \texttt{remaining}/\texttt{initial}$. Sokoban TBS uses progress-aware priority (Section B.6).

B.5.3. SLIDING PUZZLE

The Sliding Puzzle verifier reads tile values via OCR (CPU EasyOCR) at stable frames where the board is not mid-transition, simulates board transitions between consecutive stable states, and verifies that exactly one adjacent tile moved into the blank cell at each transition. It detects illegal multi-tile moves, duplicate tiles, wrong board states, and stalled segments.

In addition, an *undo-required* semantic policy checks at every stable state whether the only optimal next move is to undo the previous move. If yes, the previous move was a strategic mistake and the verifier triggers a rollback to the prior stable state. This cuts prefixes at semantic dead-ends rather than at superficial OCR failures, providing cleaner restart anchors for search.

B.5.4. KEYS-AND-DOORS

Extends the maze verifier with key-order checking. At each key-pickup position the verifier confirms the player collected the key, and after all keys are collected confirms the door is reached. OOD evaluation uses 161-frame videos with path lengths exceeding the training distribution.

B.6. TBS Search Budgets and Priority

Budget formula. A TBS configuration is parameterized by root width k_1, samples per expanded non-root node s, and maximum expansions m. Total generation budget is

$$B = k_1 + s\,(m - 1). \tag{5}$$

Matched-budget BoN samples $N{=}B$ independent root rollouts.

Domain-specific priority exceptions.

- **Sokoban (progress-aware)** The frontier priority ranks first by EM, then by BFS state quality (non-dead state, fewer remaining moves, higher progress), and only falls back to prefix length. The BFS state-quality signal therefore takes precedence over raw prefix length.
- **Sliding Puzzle.** The undo-required policy can roll the restart anchor back to the previous stable board state when the current move creates a semantic dead-end.

Per-row TBS configurations. Each row of Table 1 in the main paper corresponds to a specific (k_1, s, m) tuple:

Domain	Budget	k_1	s	m	Priority
Maze	TBS-10	2	2	5	default prefix priority
Maze	TBS-20	2	2	10	default prefix priority
Maze3D	TBS-10	8	2	2	default
Maze3D	TBS-20	8	2	7	default
Sokoban	TBS-10	8	2	2	progress-aware (BFS)
Sokoban	TBS-20	8	2	7	progress-aware (BFS)
Sliding Puzzle	TBS-10	6	2	3	default + undo-required policy
Sliding Puzzle	TBS-20	5	3	6	default + undo-required policy
Keys-Doors ID	TBS-10	6	2	3	default
Keys-Doors ID	TBS-20	4	2	9	default
Keys-Doors OOD	TBS-10	2	2	5	default
Keys-Doors OOD	TBS-20	4	2	9	default

Table 7. Structured-domain TBS search configurations. Domain-specific shapes were chosen via the depth-versus-width sweep in Section B.11. Total generation budget $B = k_1 + s(m-1)$.

B.7. Structured Video Reasoning Extended Results

B.7.1. Confidence interval methodology

We report 95% Wilson confidence half-widths because EM is a binomial proportion: a sample either solves a problem or does not. Wilson is preferred over Wald because Wald collapses at 0% and 100%. For a proportion $p = k/n$ with $z = 1.959964$:

$$c = \frac{p + z^2/(2n)}{1 + z^2/n}, \quad h = \frac{z\sqrt{p(1-p)/n + z^2/(4n^2)}}{1 + z^2/n},$$
$$\mathrm{CI}_{95} = [\,\max(0, c-h),\ \min(1, c+h)\,]. \tag{6}$$

The displayed $\pm$ is the larger of the upper and lower halves, so the interval contains the full Wilson CI even at saturation. Sample sizes are n=600 overall, n=100 easy, n=200 medium, and n=300 hard or strict-OOD.

B.7.2. Maze rich-prompt path-length breakdown

Method	0–10	11–20	21–30	31–40	41+	Total
BoN-1	99.0	90.4	49.7	13.3	1.9	51.0
BoN-5	100.0	99.2	78.9	47.5	5.7	67.7
BoN-10	100.0	100.0	89.8	65.0	12.4	75.2
BoN-20	100.0	100.0	92.5	76.7	17.1	79.0
TBS-10	100.0	99.2	93.9	80.0	41.0	84.0
TBS-20	100.0	99.2	98.6	89.2	60.0	90.3

Table 8. 2D Maze rich-prompt EM by shortest-path-length bin. The bins are used only for post-hoc analysis of horizon length, not as an online solution signal for TBS. The TBS gain grows monotonically with horizon: at $41+$ steps, BoN-20 collapses to 17.1% while TBS-20 reaches 60.0% ($+42.9$ pp). This is the numerical table backing Figure 5(a).

B.7.3. Keys-and-Doors strict-OOD path-cell breakdown

	Path cells (strict-OOD)			
Method	136–144 (n=108)	145–149 (n=101)	150–153 (n=91)	Total
BoN-20	0.9	0.0	1.1	0.7
TBS-20	15.7	25.7	27.5	22.7
Δ	$+14.8$	$+25.7$	$+26.4$	$+22.0$

Table 9. Keys-and-Doors strict-OOD EM by path-cell-count bin. BoN-20 stays below 1.1% across all bins; TBS-20 reaches 15.7–27.5%. Numerical version of Figure 5(b).

B.8. Failure Clustering Statistics

This section provides the numerical values behind the failure-to-recovery panel of Figure 5(c) and quantifies the clustering observation that motivates TBS in Section 1.

B.8.1. Failure-to-recovery distribution

Domain	Row	0–20%	20–40%	40–60%	60–80%	80–100%	Solved
2D Maze	BoN-20 failures	72.7	15.7	7.3	3.1	1.1	0.0
2D Maze	TBS-20 outcomes	5.3	0.7	4.0	5.0	4.0	81.0
3D Maze	BoN-20 failures	21.6	31.9	27.5	14.7	4.4	0.0
3D Maze	TBS-20 outcomes	2.0	0.7	1.0	1.0	0.0	95.3
Sokoban	BoN-20 failures	8.3	18.6	22.7	22.2	28.1	0.0
Sokoban	TBS-20 outcomes	0.3	2.3	2.0	8.7	7.0	79.7
Sliding Puzzle	BoN-20 failures	25.7	35.2	17.2	16.1	5.7	0.0
Sliding Puzzle	TBS-20 outcomes	2.0	0.7	2.3	3.7	2.7	88.7
Keys-and-Doors ID	BoN-20 failures	14.3	38.8	30.8	12.8	3.3	0.0
Keys-and-Doors ID	TBS-20 outcomes	0.0	0.0	0.3	0.3	0.3	99.0
Keys-and-Doors-OOD	BoN-20 failures	78.3	16.4	3.6	1.3	0.4	0.0
Keys-and-Doors-OOD	TBS-20 outcomes	4.7	6.3	12.0	17.7	36.7	22.7

Table 10. Failure-to-recovery distribution by trajectory-progress bin. "BoN-20 failures" rows give the distribution of failed BoN-20 root samples by their farthest verified-progress bin (sums to 100%). "TBS-20 outcomes" rows give the distribution of TBS-20 final episode outcomes; the right-most column is TBS hard-split EM. Maze and Keys-and-Doors-OOD show the strongest clustering: more than 70% of BoN failures concentrate in the earliest 0–20% bin, exactly the region where independent root sampling re-fails.

B.8.2. Densest local-window share

Domain	Failed records	Median densest share	$\geq 50\%$ in window	$\geq 70\%$ in window	Dominant window
2D Maze	4920	77.8%	92.9%	64.5%	frames 1–16
3D Maze	1522	93.9%	100.0%	86.8%	frames 19.5–34.5
Sokoban	3830	57.1%	71.9%	19.7%	frames 41–56
Sliding Puzzle	4306	66.7%	88.7%	44.9%	frames 14–29
Keys-and-Doors ID	2918	71.4%	93.3%	53.7%	frames 21–36
Keys-Doors strict-OOD	5997	85.0%	98.7%	81.3%	frames 1–32

Table 11. For every hard problem we compute the median fraction of failed BoN-20 seeds that fall within a single 16-frame local window (32 frames for the 161-frame strict-OOD videos, so the window covers the same fraction of the trajectory). Local clustering is strong everywhere except Sokoban.

B.9. Root Solves vs. Restarted-Branch Solves

This section directly probes the TBS mechanism: of the episodes solved by TBS, how many were already solved by a root rollout (depth 0) and how many required a restarted child branch (depth ≥ 1). If TBS gains came purely from extra root sampling, the depth-0 share would be high; if they came from prefix reuse, the depth-≥ 1 share dominates.

Domain	TBS-20 final	Root solves (d=0)	Restart solves (d$\geq$1)	Restart-added EM
2D Maze	$542/600 = 90.3\%$	357	185	$+30.8$ pp
3D Maze	$586/600 = 97.7\%$	574	12	$+2.0$ pp
Sokoban	$496/600 = 82.7\%$	448	48	$+8.0$ pp
Sliding Puzzle	$519/600 = 86.5\%$	405	114	$+19.0$ pp
Keys-and-Doors ID	$595/600 = 99.2\%$	510	85	$+14.2$ pp
Keys-and-Doors-OOD	$68/300 = 22.7\%$	0	68	$+22.7$ pp

Table 12. Depth at which each TBS-20 solve was first achieved. Root solves are solves found at depth 0 inside the TBS run; a child branch first solves restarted solves. Keys-and-Doors-OOD is the cleanest mechanism case: every solved episode comes from a restarted branch.

Domain	Depth 0	Depth 1	Depth 2	Depth 3+
2D Maze	$357/542 = 65.9\%$	$70/542 = 12.9\%$	$50/542 = 9.2\%$	$65/542 = 12.0\%$
3D Maze	$574/586 = 98.0\%$	$9/586 = 1.5\%$	$2/586 = 0.3\%$	$1/586 = 0.2\%$
Sokoban	$448/496 = 90.3\%$	$32/496 = 6.5\%$	$12/496 = 2.4\%$	$4/496 = 0.8\%$
Sliding	$405/519 = 78.0\%$	$76/519 = 14.6\%$	$24/519 = 4.6\%$	$14/519 = 2.7\%$
Keys ID	$510/595 = 85.7\%$	$63/595 = 10.6\%$	$18/595 = 3.0\%$	$4/595 = 0.7\%$
Keys-OOD	$0/68 = 0.0\%$	$11/68 = 16.2\%$	$16/68 = 23.5\%$	$41/68 = 60.3\%$

Table 13. First-solved depth distribution among solved TBS-20 episodes. Depth 0 denotes a root solve; depth ≥ 1 denotes at least one temporal restart.

Cumulative EM by search depth (2D Maze). On Maze, TBS continues to gain across multiple restart depths rather than saturating after the first restart.

Depth	Reached	New solves at depth	Solve rate when reached	Cumulative EM
0	600	357	59.5%	59.5%
1	243	70	28.8%	71.2%
2	173	50	28.9%	79.5%
3	120	33	27.5%	85.0%
4	83	15	18.1%	87.5%
5	55	14	25.5%	89.8%
6	27	3	11.1%	90.3%

Table 14. 2D Maze TBS depth-by-depth. "Reached" is the number of problems where TBS expanded a node at this depth; "new solves at depth" is the number of problems first solved at exactly this depth. The solve rate stays above 25% through depth 3, evidence that reused prefixes remain structurally useful after multiple continuations.

B.10. Root-Width Sensitivity at Matched Budget

This subsection reports the *best EM per domain* as k_1 varies at the headline budget $B{=}20$, holding the total generation count fixed. For each k_1 value, we take the maximum EM across all (s, m) combinations satisfying $k_1 + s(m{-}1) = 20$ in the per-domain sweep. The range across k_1 measures how sensitive each domain is to the choice of root width.

Domain	$k_1{=}2$	$k_1{=}4$	$k_1{=}5$	$k_1{=}8$	Range
2D Maze	90.3%	89.8%	89.8%	89.8%	0.5 pp
3D Maze	97.0%	97.3%	97.3%	97.7%	0.7 pp
Sokoban	76.5%	81.2%	82.0%	**82.7%**	**6.2 pp**
Sliding	85.0%	85.8%	86.5%	86.5%	1.5 pp
Keys-Doors ID	97.3%	**99.2%**	99.0%	99.0%	1.9 pp
Keys-Doors strict-OOD	21.3%	**22.7%**	21.7%	21.7%	1.4 pp

Table 15. Best EM per root width k_1 at fixed budget $B{=}20$. For each (domain, k_1) cell we take the maximum EM among all swept TBS configurations with that root width that fit the budget. Bold marks the best k_1 per domain. 2D Maze and Maze3D are essentially k_1-insensitive (range ≤ 1 pp); Sokoban requires wider roots ($+6.2$ pp from $k_1{=}2$ to $k_1{=}8$). Strict-OOD Keys-and-Doors prefers a balanced $k_1{=}4$ versus BoN-20 at 0.7%.

Reading. The root-width-sensitivity range across domains exposes the operating mode of TBS in each:

- **Maze and Maze3D** are k_1-insensitive (range ≤ 1 pp): TBS converges to nearly the same EM regardless of root width, and depth does the work.
- **Sokoban** requires wider roots: a single-root TBS run ($k_1{=}2$) yields only 76.5%, while $k_1{=}8$ reaches 82.7%. Sokoban prefixes can be visually plausible but already strategically dead, so additional independent roots are required to escape shallow-rollback traps.
- **Sliding Puzzle and Keys-Doors ID** prefer slightly wider roots ($k_1 \in \{4, 5, 8\}$ with ≤ 2 pp spread).

- **Strict-OOD Keys-and-Doors** prefers a balanced k_1=4, but every k_1 value yields $\geq 18.9\%$ EM versus BoN-20's 0.7%. The mechanism (prefix reuse), not the specific shape, is what matters.

B.11. Depth–Width Search Shape

The matched-budget table above isolates the effect of k_1 at the single budget B=20. Here we show the full root-width versus budget interaction. For each domain and each root width $k_1 \in \{2, 4, 8\}$, we take the best EM achievable across all swept (s, m) configurations satisfying $k_1 + s(m-1) = B$ at every available B. Lines only span the budgets where that k_1 was actually swept.

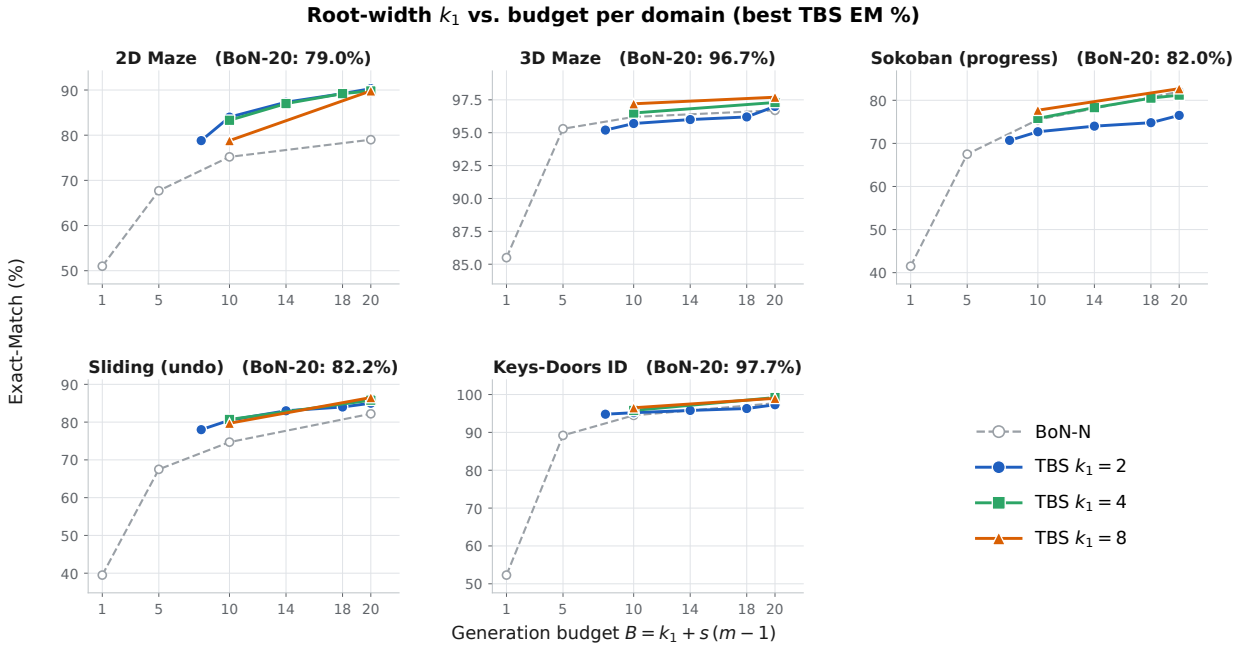


Figure 8. Per-domain root-width k_1 versus generation budget B. Each panel shows the best TBS EM (%) at every swept (k_1, B) combination for that domain; dashed gray is the BoN-N reference for $N \in \{1, 5, 10, 20\}$. **Sokoban** shows the strongest k_1 separation (wider roots dominate at every budget). **2D Maze** prefers k_1=2 (depth-friendly), with the narrow-and-deep curve sitting above wider variants at every budget. **3D Maze** and **Keys-Doors ID** are nearly k_1-insensitive at the saturation ceiling. Strict-OOD Keys-and-Doors is omitted from this plot because its absolute EM range (0–23%) cannot be meaningfully compared to in-distribution scales; per-budget numbers are reported in Table 15 and the strict-OOD path-cell breakdown in Table 9.

Cross-domain takeaway. The figure shows three regimes consistent with the per-domain analysis:

- **Depth-friendly (2D Maze):** k_1=2 dominates wider variants at every budget. Narrow-deep search converts each extra budget unit into prefix-reuse depth, which keeps producing new solves (cumulative EM by depth in Table 14 reaches 90.3% at depth 6).
- **Width-friendly (Sokoban):** k_1=8 dominates at every budget. Sokoban prefixes are strategically fragile, so wider roots are required to escape shallow-rollback traps before deeper search becomes useful.
- **Saturated (3D Maze, Keys-Doors ID):** all three TBS curves cluster within ≤ 2 pp of one another and well above the BoN reference. The mechanism helps but the operating ceiling limits visible separation across k_1.

B.12. Verifier-Quality Ablations: Sokoban BFS-Aware and Sliding Undo-Required

The Sokoban and Sliding Puzzle headline TBS rows in the main paper use domain-specific verifier upgrades. Sokoban uses `SOKOBAN_VERIFIER_MODE=smart` with `SCORE_MODE=progress`, which exposes a BFS dead-state and remaining-moves signal; without it, the verifier returns priority based purely on prefix length (the *prefix-only* verifier).

Sliding Puzzle uses `SLIDING_SEMANTIC_POLICY=undo_required`, which marks any move whose only optimal next move is to undo the previous move as a *semantic* failure and triggers rollback to the prior settled state; the alternative `SLIDING_SEMANTIC_POLICY=none` waits for a syntactic transition error. This appendix isolates how much each verifier upgrade contributes to TBS, holding generation, dataset, and search structure fixed.

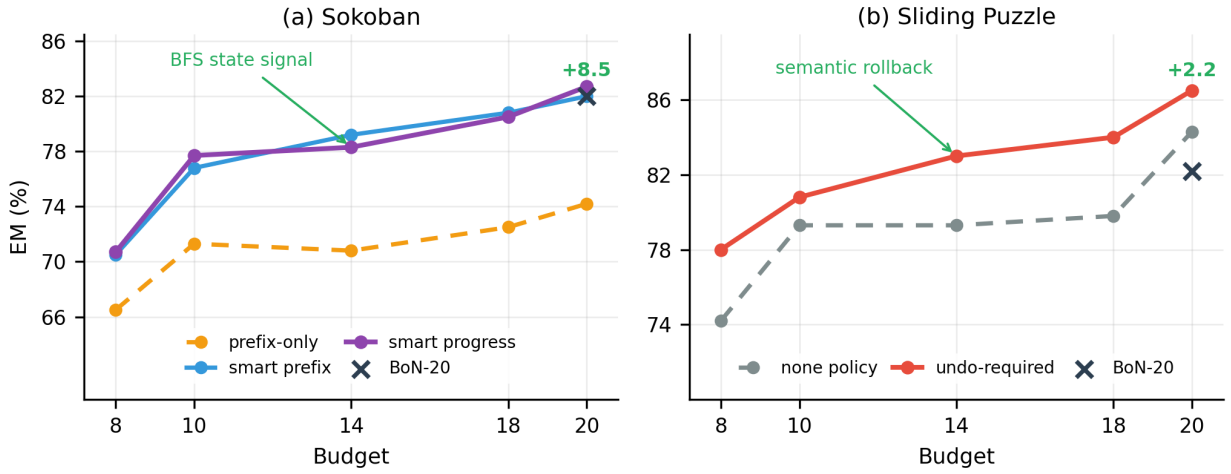


Figure 9. **Verifier-quality ablations for state-aware restart. (a)** On Sokoban, the prefix-only verifier underperforms because long visual prefixes can already be strategically dead; adding BFS-based state verification and progress priority recovers the gap. **(b)** On Sliding Puzzle, undo-required semantic rollback improves TBS across budgets while the corresponding BoN-20 reference does not improve, showing that the gain comes from cleaner restart anchors rather than a more permissive terminal evaluator.

Sokoban: state-aware verification is necessary. Sokoban is the clearest verifier-quality ablation in the structured suite because visual prefix length is not the same as useful progress. A rollout can be visually legal for many frames while pushing the box into a dead state from which the target is unreachable. Conditioning on the longest prefix then preserves the wrong strategic state, and TBS repeatedly re-attempts from a prefix that should have been rejected.

We compare three variants on the same Sokoban. The *prefix-only* variant ranks nodes by prefix length and does not use BFS dead-state information. The *BFS prefix-search* variant uses the BFS-aware Sokoban verifier to reject unsolvable box states, but still uses the standard prefix-based priority. The *progress-aware* variant is the headline setting: it ranks EM first, then BFS non-dead state, fewer BFS remaining moves, BFS progress, and finally the prefix fallback.

TBS config	Budget	Prefix-only verifier	Progress-aware verifier	Δ verifier
$k_1{=}2, s{=}3, m{=}3$	8	$399/600 = 66.5\%$	$424/600 = 70.7\%$	$+4.2$ pp
$k_1{=}2, s{=}2, m{=}5$	10	$401/600 = 66.8\%$	$436/600 = 72.7\%$	$+5.9$ pp
$k_1{=}2, s{=}2, m{=}7$	14	$409/600 = 68.2\%$	$444/600 = 74.0\%$	$+5.8$ pp
$k_1{=}2, s{=}2, m{=}9$	18	$414/600 = 69.0\%$	$449/600 = 74.8\%$	$+5.8$ pp
$k_1{=}4, s{=}2, m{=}9$	20	$438/600 = 73.0\%$	$487/600 = 81.2\%$	$+8.2$ pp
wider $k_1{=}2, s{=}2, m{=}10$	20	$415/600 = 69.2\%$	$453/600 = 75.5\%$	$+6.3$ pp
wider $k_1{=}2, s{=}3, m{=}7$	20	$418/600 = 69.7\%$	$459/600 = 76.5\%$	$+6.8$ pp
wider $k_1{=}5, s{=}3, m{=}6$	20	$431/600 = 71.8\%$	$492/600 = 82.0\%$	$+10.2$ pp
wider $k_1{=}8, s{=}2, m{=}7$	20	$445/600 = 74.2\%$	$496/600 = 82.7\%$	$+8.5$ pp

Table 16. Sokoban verifier-quality ablation under matched TBS configurations: prefix-only verification versus progress-aware verification (BFS dead-state, remaining-moves, and progress). The progress-aware verifier contributes $+4.2$ to $+10.2$ pp across all 9 matched configurations spanning budgets 8 to 20, with the gap widening at higher budgets and wider roots. Dataset, generation model, prompt, and budget are held fixed; only `SOKOBAN_VERIFIER_MODE` and `SCORE_MODE` change.

Reading. The prefix-only variant is far below BoN-20, so naive temporal branching is actively harmful on Sokoban. The BFS prefix-search variant recovers to BoN-20 parity, and progress-aware ranking adds the small headline gain. This matches the per-domain restart-quality analysis: Sokoban's deeper TBS expansions are dominated by tracking and

dead-state failures, so a state-aware priority function is required to escape shallow-rollback traps and rank non-dead BFS states above visually plausible but unsolvable prefixes.

The main lesson is not that Sokoban gives a large TBS gain—it does not. Rather, prefix reuse is only as good as the process signal defining the prefix. Maze prefixes are usually reusable because a valid route prefix remains useful. Sokoban prefixes can be *poisoned*: the box may be in a legal but strategically dead configuration. State-aware verification is therefore required before prefix conditioning becomes a helpful search primitive.

Sliding Puzzle: undo-required semantic rollback. Sliding Puzzle has a different verifier failure mode. A generated tile move can be locally legal under the blank-cell rule while still moving away from the shortest path to the goal board. Under `SLIDING_SEMANTIC_POLICY=none`, the verifier waits until a syntactic error appears (illegal transition, duplicate tile, wrong settled board), by which point the restart prefix can already be semantically poisoned. Under `SLIDING_SEMANTIC_POLICY=undo_required`, the verifier checks each settled board state using the puzzle rules and goal board: if the only optimal next move is to undo the previous move, the previous move is treated as the semantic failure and TBS rolls back to the prior settled state. This gives TBS a cleaner restart anchor without providing the correct future continuation.

TBS config	Budget	None policy	Undo-required	Δ verifier
k_1=2, s=3, m=3	8	$445/600 = 74.2\%$	$468/600 = 78.0\%$	$+3.8$ pp
k_1=2, s=2, m=5	10	$467/600 = 77.8\%$	$483/600 = 80.5\%$	$+2.7$ pp
k_1=2, s=2, m=7	14	$476/600 = 79.3\%$	$498/600 = 83.0\%$	$+3.7$ pp
k_1=2, s=2, m=9	18	$479/600 = 79.8\%$	$504/600 = 84.0\%$	$+4.2$ pp
k_1=4, s=2, m=9	20	$495/600 = 82.5\%$	$515/600 = 85.8\%$	$+3.3$ pp
wider k_1=2, s=2, m=10	20	$480/600 = 80.0\%$	$506/600 = 84.3\%$	$+4.3$ pp
wider k_1=2, s=3, m=7	20	$485/600 = 80.8\%$	$510/600 = 85.0\%$	$+4.2$ pp
wider k_1=5, s=3, m=6	20	$498/600 = 83.0\%$	$519/600 = 86.5\%$	$+3.5$ pp
wider k_1=8, s=2, m=7	20	$506/600 = 84.3\%$	$519/600 = 86.5\%$	$+2.2$ pp

Table 17. Sliding Puzzle verifier-quality ablation: `none` versus `undo_required` semantic policy across all budgets and search configurations. The undo-required policy slightly hurts BoN at every budget (-0.8 to -1.6 pp) because it changes how the verifier classifies borderline rollouts. It consistently helps TBS by $+2.2$ to $+4.3$ pp across all 9 matched configurations spanning budgets 8 to 20. The asymmetry isolates a TBS-specific contribution: cleaner restart anchors only matter once you are restarting from them. The gain trends slightly upward with search depth (e.g., $+2.7$ at m=5, $+4.2$ at m=9 at fixed k_1=2), consistent with deeper search benefiting more from cleaner anchors.

Reading. The undo-required policy consistently helps TBS by $+2.2$ to $+4.3$ pp. This is the cleanest evidence that the verifier upgrade interacts with restart specifically: the smarter verifier produces better restart anchors, which has no effect on a method that does not use restart anchors (BoN) but a measurable effect on a method that does (TBS). It also explains why the headline Sliding TBS-20 row in the main paper uses k_1=5, s=3, m=6 with undo-required: the wider-root shape works in concert with the cleaner-anchor verifier.

Cross-domain takeaway. Both ablations are consistent with the same conclusion. Where the basic prefix-length priority already aligns with structurally sound restart anchors (Maze), TBS gains immediately from prefix conditioning. Where basic prefix-length priority returns visually or semantically corrupted anchors (Sokoban, Sliding), a domain-specific verifier upgrade is the operational path to TBS-style gains: BFS-progress for Sokoban, undo-required semantics for Sliding.

B.13. Search-Policy Ablation: Greedy, Beam-2, and Random Frontiers

The main paper compares TBS against matched-budget root-only BoN. To isolate whether the gain comes merely from any prefix reuse or from the priority-frontier search policy, we run three additional prefix-reuse controls on the hard structured splits. All controls use the same variable-K generator, prompt, verifier, terminal evaluator, and per-domain TBS search shape (k_1, s, m) as the corresponding TBS row. Only the frontier policy changes:

- **Greedy:** priority pop with frontier cap 1. This is single-lineage prefix repair: after each expansion, only the best verified anchor is retained.

- **Beam-2:** priority pop with frontier cap 2. This retains the best two verified anchors after each expansion, testing whether one alternate prefix is enough.
- **Random:** uncapped frontier with random pop. This keeps frontier diversity but disables the verifier priority at the pop step.

Domain	BoN-20	Greedy	Beam-2	Random	TBS-20
2D Maze Hard	60.0	61.0	78.3	74.7	**81.0**
Maze3D Hard	93.7	95.0	**95.3**	95.0	**95.3**
Sokoban Hard, progress-aware	78.3	74.7	77.3	78.7	**79.7**
Sliding Hard, undo-required	81.3	83.7	87.3	84.0	**88.7**
Keys-Doors ID Hard	97.7	96.0	97.3	98.7	**99.0**
Keys-Doors strict-OOD	0.7	6.7	19.7	12.7	**22.7**

Table 18. Search-policy ablation at matched headline budget B=20 on hard / strict-OOD structured splits. Greedy, Beam-2, Random, and TBS use the same per-domain TBS-20 shape; only the frontier memory and pop rule change. All entries are exact-match percentages over n=300 samples. TBS is best or tied on every split.

Reading at B=20. Greedy prefix repair is not enough: it is far below TBS on 2D Maze (61.0 vs. 81.0), Sliding (83.7 vs. 88.7), Sokoban (74.7 vs. 79.7), Keys-ID (96.0 vs. 99.0), and strict-OOD Keys (6.7 vs. 22.7). Beam-2 recovers much of the gap because it preserves one alternate restart anchor, but full TBS remains best or tied on all splits: $+2.7$ pp over Beam-2 on Maze, $+2.4$ pp on Sokoban, $+1.4$ pp on Sliding, $+1.7$ pp on Keys-ID, and $+3.0$ pp on strict-OOD Keys. Random sometimes beats Greedy by avoiding single-anchor over-commitment, but it remains below TBS on every split with headroom, showing that the verifier priority is useful in addition to frontier diversity.

Conclusion. The policy ablation refines the claim. The dominant mechanism is verified temporal prefix reuse, but the full priority frontier is not redundant. Greedy single-lineage repair over-commits. Beam-2 is a strong compact approximation, especially at shallow budgets, but full TBS is best or tied at the headline budget across all six structured splits.

C. Target-Bench Navigation Details

C.1. Official benchmark score and metric definitions

Target-Bench reports five trajectory metrics combined into a single overall score:

$$S = 0.05\, e^{-\mathrm{ADE}} + 0.10\, e^{-\mathrm{FDE}} + 0.10\,(1{-}\mathrm{MR}/100) + 0.65\,\mathrm{SE}\cdot\mathrm{AC}, \tag{7}$$

where ADE / FDE are average and final L2 displacement errors against the GT trajectory (interpolated to GT length), MR is the percentage of interpolated points farther than 1 m from GT, SE is a soft endpoint score $\exp(-\text{endpoint_error}^2/2\sigma^2)$ with σ=0.5, and AC is an approach-consistency corridor score using GT reference points with $\sigma_{\min}$=0.5, $\sigma_{\max}$=1.5, β=0.25, γ=5.0.

Trajectories are decoded from generated videos via VGGT, anchored to the first frame pose, and converted to benchmark metric space using the calibration $\lambda = d_{\text{real}}/d_{\text{pred}}$. The reported numbers are the official benchmark overall score of the selected generated video.

C.2. Predictor details

The Target-Bench verifier is a learned trajectory predictor that maps the current frame and prompt to a metric-space future path. The predictor uses V-JEPA2 ViT-G/384 image tokens and text embeddings, followed by a fusion image-text transformer with 4 layers, 8 heads, hidden dimension 512, and dropout 0.1. It predicts 9 future trajectory points in benchmark metric space. Training uses 100 scenarios with a weighted point loss [1.0, 1.0, 1.0, 1.2, 1.5, 1.8, 2.2, 2.6, 3.0] and a late-x weight of 0.2 starting at point index 5.

At test time, a generated video is decoded with VGGT into a trajectory and compared against the predicted path at the 9 prefix points. We use a bell-shaped per-axis tolerance schedule with τ_{base}=0.15 m at the prefix endpoints

rising to $0.50\,\mathrm{m}$ near the middle, scaled by τ_x/τ_y. A prefix point is invalid when $|dx| > \tau_x$ or $|dy| > \tau_y$. The first invalid prefix point defines the restart anchor, with restart backoff 1. TBS-3 uses $k_1{=}2, s{=}1, m{=}2$ with TBS-5 uses $k_1{=}4, s{=}1, m{=}2$.

D. RoboTwin Manipulation Details

D.1. Pipeline

A generated video is decoded into a 14-D RoboTwin/Aloha action sequence by an inverse dynamics model and replayed in the RoboTwin simulator. Verifier inputs are the simulator's per-step state: gripper open/close events, object displacement, object–target distance, task-specific progress, and the official task predicate `check_success()`.

- Generator: released VIDAR / Wan-style TI2V (Feng et al., 2025).
- Action decoder: released VIDAR IDM (Feng et al., 2025).
- Action format: 14-D RoboTwin/Aloha sequences.
- Observation: stitched 3-view RGB image (head + left/right wrist cameras).
- Semantic FPS: 8.
- Segment length: 61 frames per generation (1 conditioning + 60 generated), with optional 61+61 continuation when the first segment does not yet contain a goal-attempt window.
- Search budget is counted at the rollout/branch level.
- TBS-3 uses $k_1{=}1, s{=}1, m{=}3$; TBS-5 uses $k_1{=}1, s{=}1, m{=}5$.

D.2. First-goal-attempt window

The verifier identifies a *first goal attempt*: a gripper close–open window co-occurring with object motion toward the task target and a release/settle event near the goal pose. Detection uses simulator state, IDM-decoded actions, gripper timing, and object–target distance. When a root rollout fails after a meaningful first attempt, TBS restarts from the preceding gripper-opening boundary—not from the initial observation—so the pre-grasp setup motion is preserved and compute is concentrated on repairing the failed manipulation.

This avoids two failure modes of naive restart: (i) crediting repeated retries after the first failed attempt as separate solves, and (ii) discarding successful pre-grasp motion that took several seconds to produce.

D.3. The 10-task progress-aware subset

Task	Type	Restart cue
`place_bread_skillet`	place	release near skillet
`stack_blocks_two`	stack	failed stack/release
`place_burger_fries`	place	release near target
`place_shoe`	place	object-target release
`stack_bowls_two`	stack	failed bowl stack/release
`place_a2b_right`	move/place	gripper-open near target
`place_a2b_left`	move/place	gripper-open near target
`place_fan`	place	release near fan target
`place_mouse_pad`	place	release near pad target
`place_object_scale`	place	release near scale

Table 19. RoboTwin 10-task progress-aware subset used in the paper.

Selection criteria. The subset is chosen to test the mechanism TBS is designed for: branching from verified partial progress. Tasks require progress-aware manipulation rather than one-shot contact triggering, a clear first-goal-attempt window detectable from simulator state, visible sub-goal structure (reach/grasp/move/place/stack/release).

Evaluation protocol. We run 10 evaluation episodes per task across the 10 tasks listed above. Each cell is evaluated under two scene-randomization regimes: *clean*, which preserves the canonical RoboTwin initial state, and *randomized*, an out-of-distribution split that perturbs the visual background and object placement relative to the conditions seen by the generator. Both regimes use the same task list, the same per-cell episode count, and the same matched-budget BoN versus TBS comparison.

Setting	Method	Budget	Successes	Success rate
Clean	BoN	3	$46/100$	46%
Clean	TBS	3	$50/100$	50%
Clean	BoN	5	$49/100$	49%
Clean	TBS	5	$53/100$	53%
Randomized	BoN	3	$15/100$	15%
Randomized	TBS	3	$16/100$	16%
Randomized	BoN	5	$17/100$	17%
Randomized	TBS	5	$18/100$	18%

Table 20. RoboTwin selected-subset results. In clean scenes, TBS gains $+4$ successful episodes over matched-budget BoN at both budgets. In randomized scenes, absolute performance is lower and the gain is smaller because fewer root rollouts produce repairable verified prefixes.

E. Limitations and Failure Cases

TBS requires a useful verified prefix. TBS amplifies a base model's local competence: it can stitch verified partial trajectories into longer correct ones. It cannot manufacture competence the base model does not have. When the base rollout fails before producing meaningful progress (RoboTwin randomized setting, Sliding hard with semantic prefix corruption, Maze3D 24-grid where the template tracker loses the agent on long horizons), TBS has little to reuse and the gain shrinks toward zero.

Verifier mislocalization. TBS depends on the verifier returning an approximately correct restart anchor. The Sokoban smart verifier exposes the limit case: deeper TBS expansions in Sokoban are dominated by tracking failures, meaning the verifier rolls back only a few frames before re-failing on visually unstable prefixes. The width-versus-depth ablation (Section B.11) shows that wider roots, not deeper search, mitigate this for Sokoban. Sliding undo-required has a similar structure: when the verifier mis-detects the wrong-board state, TBS inherits the corrupted prefix and re-fails.

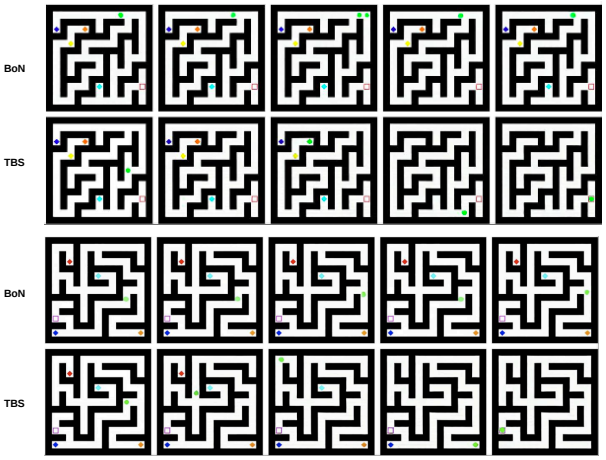
Learned process supervision is the bottleneck on Target-Bench. The Target-Bench gain is positive at both matched budgets, but the remaining headroom depends on the quality of the learned process signal. The synthetic-noise ablation (Section C) shows that TBS does not collapse under moderate process-hint noise but does converge with BoN at the strongest perturbation tested (q=0.3

, ~ 0.7 m mean ADE noise).

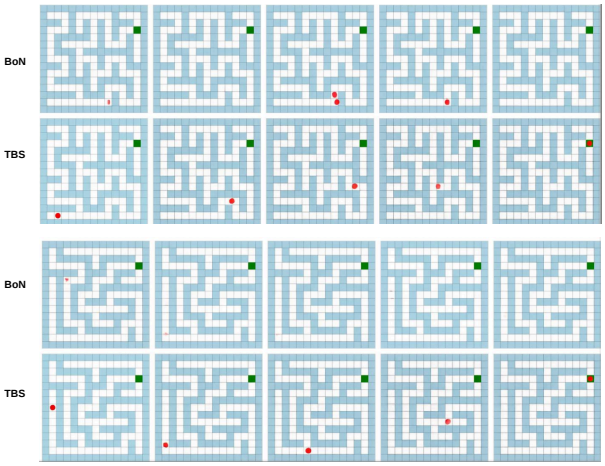
Domain saturation. On 3D Maze, in-distribution Keys-and-Doors, and most Sokoban splits, BoN-1 already solves a large fraction of problems. TBS matches or slightly exceeds BoN at matched budget on these but does not produce large gains. We treat them as null controls for downside risk rather than as evidence for TBS.

Compute overhead. At budget B, TBS requires the same nominal generation count as BoN-B, but the verifier must run on each rollout (root and child). For exact symbolic verifiers this is negligible; for the learned Target-Bench verifier and the simulator-replay RoboTwin verifier the overhead is measurable but small relative to the cost of video generation.

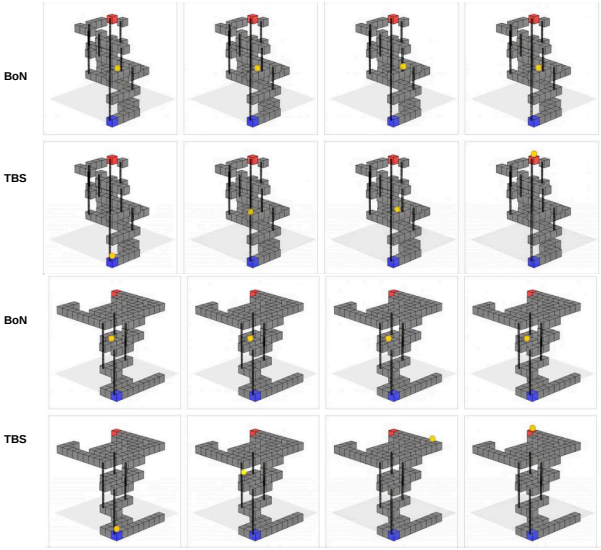
F. Visual Examples



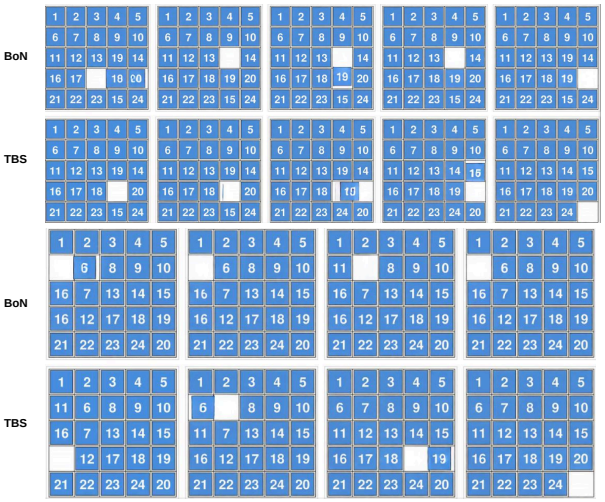
(a) Example 1



(b) Example 2

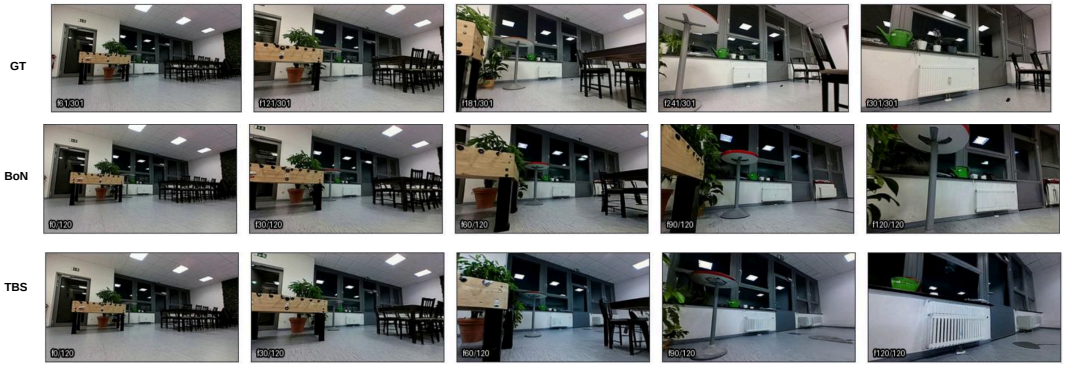


(c) Example 3

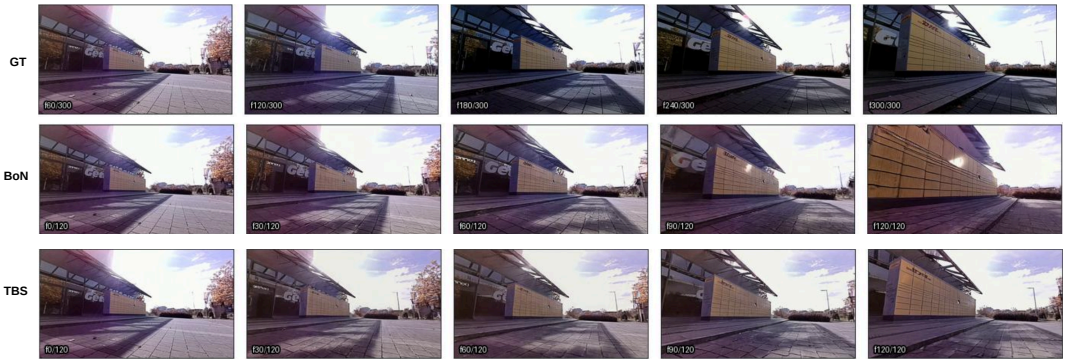


(d) Example 4

Figure 10. Visual examples from the experiments, part 1.



(a) Example 5



(b) Example 6

(c) Example 7

Figure 11. Visual examples from the experiments, part 2.