WORLD STATE GENERATOR

Anonymous authors
Paper under double-blind review

ABSTRACT

Language agents solve complex tasks through plans and actions. A single step the world refuses puts the goal out of reach, and what the agent does next decides the task. Prompted planners fail at exactly this point, rewriting the refused step in new words, meeting the same refusal, and burning the attempt budget without moving. They fail because the plan was never tied to the world, so a refusal has nothing in the plan to attach to. A world is where a task runs, and it has its own rules, its own admissible actions, and its own constraints. We build synthetic worlds across 7 domains and extract training data from them. A program enforces each world's rules and grades its goal, and every world is admitted only if its goal is reachable from its initial state. Agents run inside and leave verified failures paired with repairs that carried the run to a state the world certified, a record of about 226K trajectories. On this record we train the *World State Generator*, a model that writes a plan as checkable states of the world and keeps that plan aligned with the world it runs in. That alignment is what a plan written in language lacks, since the world it runs in has physical limits, logical dependencies, and required orders the language never states, and the plan encounters these rules only when a state fails. *WSG* takes that failure as the rule the world has stated and rewrites the remaining states to obey it, so the plan bends to the world as the run goes on. Across 7 public benchmarks, *WSG* raises end-to-end success for two open models near 30B parameters over prompting and brings to the level of proprietary model.

1 INTRODUCTION

A language agent working through a long task follows a plan, and long tasks give the world many chances to refuse a step of it. When that happens, the agent rewrites the refused step in different words, submits it again, and meets the same refusal, cycling until the attempt budget is spent (Huang et al., 2024; Yan et al., 2026). Reflecting on the failure before the rewrite does not break the cycle (Shinn et al., 2023; Madaan et al., 2023), which survives because the plan was never tied to the world. Since a prompted planner writes what it imagines the task should look like, nothing in that plan says which tools exist, which calls the interface accepts, or which conditions must already hold. The model therefore has nothing in its plan to attach the world's report to, and when that report comes, the only thing left to change is the wording. A refusal is therefore information, the world stating a rule the plan omitted, and repair means reading that rule and rewriting the rest of the plan to obey it.

Learning to repair this way requires data with three properties. Every failure must come with the repair that resolved it, and both sides must be decided against the world rather than by opinion. Agent corpora are filtered for success, so failures are rarely kept at all (Zeng et al., 2024; Chen et al., 2023), generated environments that collect trajectories at scale harvest successes in the same way (Hu et al., 2025; Song et al., 2026; Guo et al., 2025), and a failure that does survive stands alone. Even when a failure and a repair are recorded together neither is verified: no procedure checks whether a written state holds in its world, so nothing establishes that the failure was real or that the repair worked. The third property concerns where the pairs come from. A model trained on one world learns that world's rules, and rules are what differs between worlds, so the pairs must span many worlds for the model to learn how to read a refusal instead. Agents trained in one environment degrade sharply when the evaluation environment changes (Li et al., 2026).

No existing corpus holds pairs with these three properties, so we build them ourselves. We construct synthetic worlds, self-contained environments whose rules are enforced by program, and run agents

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.

inside them under a runtime that checks each state against the world. Two choices follow from what classical planning already knows. A world is written in a form a program can execute, so a solver can confirm that its goal is reachable from its initial state before any agent runs in it (Liu et al., 2023; Guan et al., 2023), and each world is composed from seed material rather than stamped from one template (Hu et al., 2025). The worlds are drawn from 7 domains, whose states record what must be known, what must be built, or how the world must stand, and no two worlds in a domain enforce the same rules. Agents running inside fail against the rules and repair their plans, and we keep a repair only when it carried the run to a state the world certified. Each kept pair holds a failure and the repair that followed it, both decided by the world rather than by us, and the pairs come from worlds that differ across all 7 domains.

On these pairs we train the World State Generator (*WSG*), a planner that keeps its plan aligned with the world it runs in. Alignment is what a prompted planner cannot hold, since the rules that decide a plan are physical limits, logical dependencies, and required orders that the goal never states, and the planner meets them one at a time as its steps fail. The runtime that collected these pairs is the State-Centric Decision Process (*SDP*) (Jeong et al., 2026), where each step of a plan is written as a state the world must reach, so the loop checks each state against the world and reports which condition did not hold and what it found instead, a refutation. *SDP* fills the planner in this loop by prompting, and *WSG* is a model trained to fill it, so the loop is unchanged and only the planner inside it learns. We train it on the repair call alone, so the first plan of a run is still written by the prompted backbone. *WSG* reads that refutation as a rule of the world and rewrites the remaining states to obey it, so the plan bends to the world as the run goes on. Given a goal, the states the world has already certified, and the refutation, *WSG* returns the states that carry the world from where it stands to the goal.

Our contribution is the pipeline that produces these pairs at scale, the model trained on them, and the finding that repair learned in synthetic worlds transfers to worlds outside the corpus. We run 226K episodes in this way and train *WSG* on the 239K failure–repair pairs they yield. The evaluation is zero-shot, since none of the benchmarks contributed a world, a task, or a trajectory to the corpus. Across 7 public agent benchmarks, the trained models gain 3.4 to 31.9 points of end-to-end success over prompting and, on all of them, land within the range of published proprietary results. The gains concentrate where the supervision aimed, in repairs after a failed state.

2 Related Work

Decomposition and its unit. Work on task decomposition sits between two poles that differ in what a part of the plan is. At one pole, least-to-most prompting and its successors elicit subproblems in free text, so a part is a sentence whose correctness nobody checks (Zhou et al., 2022; Wang et al., 2023). Plans built this way cannot be trusted without outside verification (Kambhampati et al., 2024). At the other pole, language models write PDDL for a symbolic planner, so a part is a predicate a solver decides (Liu et al., 2023; Guan et al., 2023). The price is a symbol set a person must formalize for every new domain. *SDP* keeps the part decidable and drops the formalization, since it writes each part as a state in natural language and decides it against observation at run time (Jeong et al., 2026). The model that writes these states is prompted in *SDP*. This paper trains it on the repair call.

Supervision for agents. Agent supervision has scaled by collecting more of what worked. Tuning corpora collect interaction traces from capable models, so what they teach is how a stronger model behaves (Zeng et al., 2024; Chen et al., 2023). Synthesis pipelines replay tutorials or explore applications to manufacture such trajectories at scale (Xu et al., 2025; Sun et al., 2025). The environments themselves are now generated by program, from PDDL domains (Hu et al., 2025) to tool backends (Song et al., 2026) and full web sites (Guo et al., 2025). A generated environment is checked for being well formed and executable before agents run in it. Verification of the trajectories has tightened alongside, with execution checks and rule-based validation filtering what enters training (Liu et al., 2024). What survives into training is still the path that succeeded. Rejection sampling states this bias openly, since it keeps the completions whose answer passed and discards the rest (Zelikman et al., 2022; Yuan et al., 2023). This line therefore does not supply the unit repair needs, which is a verified failure paired with the repair that resolved it. Our worlds are also admitted before use, each with a goal reachable from its initial state. What we keep from them is the opposite record, the refusals their rules issued and the repairs that carried the run to a state the world certified.

Learning to repair. Repair entered the literature as a prompting strategy. Reflexion turns a failure into a verbal lesson replayed in context, Self-Refine has the model critique its own output, and CRITIC routes interpreter and test results back into the prompt (Shinn et al., 2023; Madaan et al., 2023; Gou et al., 2024). Prompting has a documented limit, since without external feedback a model rarely locates its own fault and often revises an answer that was right (Huang et al., 2024). The response has been to train the ability, and training so far has meant reward. SCoRe reports that supervised fine-tuning on the model's own correction traces collapses onto the first attempt, and it therefore trains through multi-turn reinforcement learning (Kumar et al., 2025). Verifiable-reward and process-reward methods keep a verifier but reduce its verdict to a score, so the model learns that a step failed and never sees what the verifier found in its place (Lambert et al., 2024; Lightman et al., 2024). The collapse SCoRe reports was observed on corrections the model wrote for itself, judged only by the final answer. Our pairs differ on both counts. Each repair is conditioned on the refutation, which is evidence the first attempt never had, and a repair that restates the refuted chain is dropped at corpus construction. Regression onto the first attempt is therefore a pattern the training data never contains, and on such pairs repair is learnable by supervised fine-tuning alone.

3 PRELIMINARIES: THE STATE-CENTRIC RUNTIME

This paper builds on the State-Centric Decision Process (*SDP*) (Jeong et al., 2026), a runtime that makes every step of an agent's execution verifiable. An *SDP* agent does not select actions against raw observations but commits to states of the world. Each state is written as a predicate, which is a natural-language condition that can be checked true or false. The task's goal is itself a predicate, written g. Four operators drive the runtime. PROPOSE writes the plan, REALIZE acts toward it, VALIDATE judges each state against the world, and REPLAN rewrites the plan after a state fails. PROPOSE and REPLAN decide what the agent attempts, and this paper concerns those two. The other two stay as *SDP* defines them, and one fact about VALIDATE matters throughout. When a state fails, VALIDATE records the observation that stood in its place. That observation describes what held and carries no diagnosis or advice.

PROPOSE is where an open task takes form. Given a goal that says only how the world should end up, it writes the states the world must pass through on the way. In *SDP*, this is a prompted call to a large language model,

$$\hat{C} = \textsc{Propose}(g,\ c,\ \tau), \qquad \hat{C} = (\hat{s}_1, \ldots, \hat{s}_n), \qquad \hat{s}_n = g, \tag{1}$$

where g is the goal, c is the prefix of states the world has already certified, and τ is the refutation defined below, empty on the first call. The hat marks a state the world has not yet ruled on, and the runtime rules on one state at a time. REALIZE acts until it can report an observation for the first proposed state, and VALIDATE decides that state against the observation,

$$o \leftarrow \textsc{Realize}(c,\ \hat{s}_1), \qquad (c,\ \tau) \leftarrow \begin{cases} (c \oplus \hat{s}_1,\ \varnothing) & \text{if } \textsc{Validate}(\hat{s}_1,\ o) \text{ certifies } \hat{s}_1, \\ (c,\ (\hat{s}_1,\ o)) & \text{otherwise.} \end{cases} \tag{2}$$

Here $c \oplus \hat{s}_1$ appends $\hat{s}_1$ to the end of c, so a certified state joins the prefix and the chain advances to the next. When VALIDATE refutes a state, that state and the observation recorded in its place become the refutation, written $\tau = (s_f,\ o_f)$. After a refutation Eq. 1 is called again from the current prefix, so execution moves forward only, the prefix is never reopened, and the loop ends when g is certified. On the first call τ is empty and the call decomposes, and with τ present it repairs, the call *SDP* names REPLAN. The two are one operator under the single interface of Eq. 1, distinguished only by whether τ is present, and we call that operator the *proposer*. Throughout, the lowercase nouns proposer, realizer, and validator name the models that fill PROPOSE, REALIZE, and VALIDATE.

The model that fills this interface was never trained for it. What the runtime needs from the call is the chain whose execution most likely carries the world from where it stands to the goal,

$$\arg\max_{\hat{C}}\ P\big(g \text{ certified} \mid c,\ \hat{C}\big), \tag{3}$$

where the probability is over the runtime's stochastic execution of the chain through Eq. 2, yet no prompted model was trained toward Eq. 3. Its chains are whatever the prompt elicits, and they change when the wording does. The gap is widest on the calls that follow a refutation, for repairing under τ settles two decisions at once, what the rewritten remainder must pass through and how finely to cut it.

4 WORLD STATE GENERATOR

We build synthetic worlds whose rules a program enforces and whose goals are reachable from their initial states. Inside these worlds we run the *SDP* loop and collect pairs of an input $(g,\ c,\ \tau)$ and the repair written after it, keeping each repair that carried the run to a state the world certified. On these pairs we train the proposer for the repair call and call the trained model the World State Generator. Such pairs come only from worlds like these, since a repair can be certified only where the rules are enforced, and a refutation is worth repairing only where the goal was reachable.

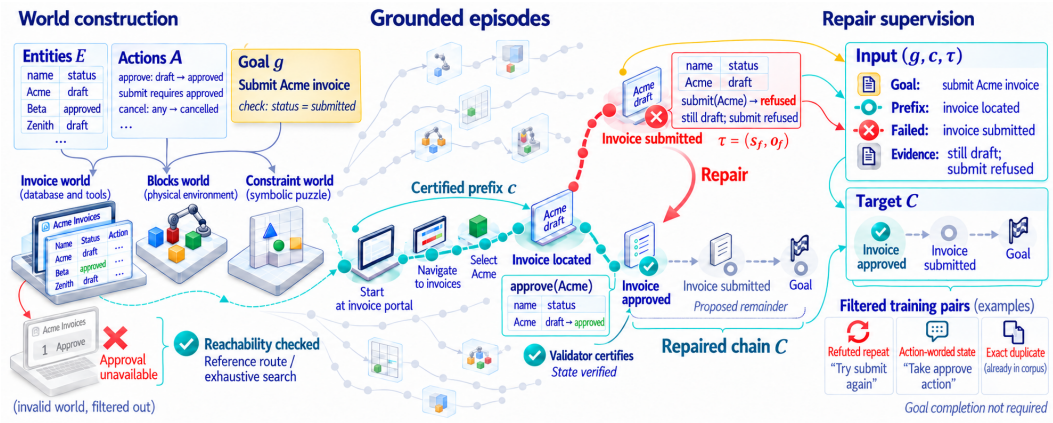

Figure 1: Left: worlds are built from entities, actions, and a goal, and a world is kept only if its goal is reachable from its initial state. Middle: agents run the *SDP* loop inside these worlds. When the world refutes a state, the certified prefix stays and the rest of the chain is rewritten. Right: each refutation becomes a training pair, the input (g, c, τ) and the rewritten chain. A pair is kept when the rewritten chain reaches a state the world certifies, whether or not the run later reaches the goal.

4.1 THE WORLD AS A STATE CHAIN

A world is a self-contained environment with three parts, **entities** with attributes, **actions** with rules, and a **goal** given as a sentence,

$$W = (E,\ A,\ g), \qquad g = (\ell_g,\ \chi_g). \tag{4}$$

The attributes of E are what an action in A can read or change, and the rules of A decide what each action does and when it is refused. The goal carries two forms, the sentence ℓ_g that the agent reads and the check χ_g that the program runs on the entities to decide whether the goal holds. In a tool world the entities are database rows such as invoices and tickets, the actions are tools that read and change them, a rule refuses a call on an invoice that is still in draft or on an id the table does not hold, and ℓ_g reads "Put the invoice Acme Consulting through for payment" while χ_g inspects the status of that row in the final table. Table 1 gives the three parts for each of the 7 domains.

A world is composed by drawing its three parts and joining them, and the parts supply the two sides of the problem, ℓ_g is the sentence the plan is written from, and E and A are the world that holds the limits, dependencies, and orders the sentence never states. Joining the parts is not enough on its own, because the three are composed separately and a goal can demand a condition that E and A together never allow. A plan in such a world fails for no reason it could repair, so admission removes the world before any agent enters. Writing e_0 for the initial configuration of the entities, admission proves

$$\exists\, a_1, \ldots, a_k \in A \quad \text{such that} \quad \chi_g\big(a_k \circ \cdots \circ a_1(e_0)\big) = 1, \tag{5}$$

by executing a reference route where one can be written and by exhaustive search where it cannot. What remains is a world whose goal was reachable when the run began. The per-domain procedure is given in Appendix A.2.

Inside an admitted world the agent runs the *SDP* loop, and one refutation from the invoice world shows what the loop records. The proposer writes a chain such as "the invoice Acme Consulting

Table 1: Entities, actions, enforced rules, goal checks, and admission criteria for the 7 domains.

Domain	Component	Description
Physical	Entities	Objects with locations and properties.
	Actions	Primitive actions such as open, move, and combine.
	Enforced rule	An action whose precondition does not hold changes nothing.
	Goal check	The goal predicate on the final configuration.
	Admission	Exhaustive search over configurations.
Software	Entities	An index of Wikipedia documents.
	Actions	Search, read, and note.
	Enforced rule	A read of a title the index does not hold returns an error with similar titles, and a search with no terms is refused.
	Goal check	The submitted answer against the gold answer.
	Admission	Every gold document is found by a search on its title.
Web search	Entities	Pages of a crawled snapshot.
	Actions	Search, navigate, read, and note.
	Enforced rule	A page opens only from a search result or a link on an open page.
	Goal check	The submitted answer against the snapshot.
	Admission	The reference route reaches the page, and exactly one page fits.
Tool use	Entities	Database rows such as invoices and tickets.
	Actions	22 to 36 tools that search, list, read, create, update, and advance rows.
	Enforced rule	A call on the wrong status or an unknown id is refused, and lists return ten rows per page.
	Goal check	Row status and events in the final table.
	Admission	The reference route is accepted, and a wrong run is rejected.
Constraint	Entities	Candidates per slot with prices.
	Actions	Inspect and select candidates.
	Enforced rule	A selection that breaks a hidden policy is refused.
	Goal check	Every constraint on the submitted selection.
	Admission	Exhaustive search over combinations finds exactly one solution, and a decoy is rejected.
Reasoning	Entities	A subject text and hidden facts.
	Actions	Record a finding on an aspect, and conclude a verdict.
	Enforced rule	A finding is checked against the hidden facts by program.
	Goal check	The verdict against the hidden facts.
	Admission	The verdict from the subject alone differs from the verdict with the facts.
Math	Entities	A problem and the values determined so far.
	Actions	Derive a value, and submit an answer.
	Enforced rule	Only the submitted answer is checked, by program against the reference answer.
	Goal check	The answer under tolerance.
	Admission	The reference answer is machine checkable.

has been located", "the invoice has been submitted for payment", and the second state fails when the submit tool refuses an invoice that is still in draft. The validator reads the program's record of the call and the table and writes that the invoice is still in draft and that the submission was refused. That sentence is o_f, and with the refuted state it is the τ of Eq. 2. From the table as the refused call left it, the proposer places "the invoice has been approved" before the submission and keeps the rest.

The invoice run divides its decisions in a way that holds in every domain. What a call returns, what an action changes, and whether χ_g holds are decided by the program. Whether an intermediate state holds is decided by the validator, a language model that reads the program's record and answers only that question, except in reasoning worlds, where a program checks each finding against the hidden facts. An episode ends when χ_g passes or when the attempt budget is spent, and a chain that completes without χ_g passing is handled by domain. In physical, software, and web worlds the check returns only that the goal does not hold, so there is nothing to repair from and the episode fails, while in tool, constraint, reasoning, and math worlds the check can name the condition it found

Table 2: 7 domains, what the program grades in each, and what each extraction model produced inside them. Episodes counts runs of the *SDP* loop, Refuted the share of episodes in which the world refuted at least one state, and Reached the share that certified the goal. Repair calls counts the repair calls the proposer made, and Kept the share of those that survived the corpus filters. Intermediate states are judged by the validator, by a program in reasoning worlds, and the goal by the program in every domain.

Domain	What the program decides	Model	Episodes	Refuted (%)	Reached (%)	Repair calls	Kept (%)
Physical	the goal check on the engine state	Qwen3-32B	11,064	90.6	24.9	85,921	33.3
		Qwen3.6-27B	10,614	91.5	19.1	87,231	39.2
		Gemma4-31B	10,097	87.9	19.0	80,134	15.8
Software	the submitted answer against the gold answer	Qwen3-32B	11,040	86.2	43.4	37,870	27.0
		Qwen3.6-27B	10,239	69.4	60.6	27,357	30.7
		Gemma4-31B	10,380	67.3	53.0	28,200	33.3
Web search	the answer against the snapshot	Qwen3-32B	13,338	51.3	87.6	19,671	37.7
		Qwen3.6-27B	10,306	39.9	90.5	14,858	29.3
		Gemma4-31B	11,279	52.6	72.7	25,375	21.8
Tool use	the database after the run	Qwen3-32B	12,614	73.9	78.5	29,272	16.0
		Qwen3.6-27B	10,141	78.3	88.8	22,359	20.1
		Gemma4-31B	11,074	67.7	94.6	17,463	17.5
Constraint	every constraint on the submitted selection	Qwen3-32B	11,345	97.9	39.3	46,873	17.3
		Qwen3.6-27B	10,495	96.0	54.9	35,979	20.1
		Gemma4-31B	10,817	97.6	62.5	26,208	34.6
Reasoning	the verdict against the hidden facts	Qwen3-32B	10,508	96.8	53.2	55,481	43.2
		Qwen3.6-27B	10,444	87.3	78.6	34,922	58.9
		Gemma4-31B	10,003	75.2	79.6	27,922	52.5
Math	the final answer under tolerance	Qwen3-32B	10,291	58.3	61.2	22,950	31.0
		Qwen3.6-27B	10,203	71.6	55.1	30,621	31.9
		Gemma4-31B	10,024	46.4	66.4	18,513	27.4
Total			226,316	75.3	61.5	775,180	30.8

unmet or the answer it rejected, and the proposer receives that as one more refutation and repairs from it.

A run leaves the world written as a chain of states, and the chains written after a refutation are what the corpus keeps,

$$\mathcal{D} = \Big\{ (x,\, C) \;:\; x = (g,\, c,\, \tau),\;\; C = (s_1, \ldots, s_n),\;\; s_n = g,\;\; \exists\, i \le n \text{ such that the run certified } s_i \Big\}, \tag{6}$$

one repair for every τ after which the run certified at least one state of the rewritten chain. From the invoice episode Eq. 6 yields one example, with the goal, the prefix up to "located", and the τ of the refused submission as x, and the chain from "approved" to the goal as C. The target is the whole remainder the proposer wrote, and what is certified about it is the prefix the run went on to reach, at least the decision of what to establish first where the previous chain broke, while the rest of the chain is decomposition that the run may or may not have carried out. 48.9% of the kept repairs come from episodes that went on to certify the goal, and the rest from episodes that certified part of the repair but ended short of the goal. A repair is dropped if it restates, as a set of states, a chain that was already refuted from the same prefix, if any of its states names an action rather than a condition, or if it duplicates a kept example exactly.

Table 2 gives the yield of Eq. 6. Three open models near 30B parameters, Qwen3-32B, Qwen3.6-27B, and Gemma4-31B, ran the loop over the same worlds for 226,316 episodes, and the world refuted at least one state in 75.3% of them. Those refutations produced 775,180 repair calls, and 238,563 of the repairs, 30.8%, passed the filters. The domains contribute unequally, from 75,455 repairs in physical worlds to 12,243 in tool worlds. The difference comes first from how often each world refutes, 253,286 repair calls in physical worlds against 69,094 in tool worlds, and only second from the share that passes, which ranges from 17.7% in tool worlds to 50.0% in reasoning worlds. A kept repair holds 5.5 states on average, from 1.2 in constraint worlds to 11.2 in physical worlds, which is shorter than the 5.9 states of a first plan because the certified prefix is not rewritten. An episode attempts 8.7 states on average, from 4.8 in web worlds to 18.5 in physical worlds, and Appendix A gives the lengths per domain.

4.2 Training the World State Generator

A training example is one element of $\mathcal{D}$ in Eq. 6, serialized with the prompt the proposer received at extraction, so the training input is the call itself. *WSG* is the backbone with LoRA (Hu et al., 2021) adapters θ on every attention and feed-forward projection of its language model, and θ is trained by supervised fine-tuning on these pairs alone,

$$\mathcal{L}(\theta) = -\sum_{(x,\, C)\in\mathcal{D}} \sum_{t=1}^{|C|} \log p_\theta\big(C_t \mid x,\, C_{<t}\big), \tag{7}$$

where C_t ranges over the tokens of the chain and the tokens of x carry no loss. No reward and no comparison between chains enters the objective. Every chain in $\mathcal{D}$ holds a state the world certified after the previous chain broke, so Eq. 7 raises the likelihood of remainders the world certified at least in part, and a remainder the world refuted at every state never enters the objective.

The 7 domains contribute unequally, so examples are drawn per domain with probability

$$p_d \;\propto\; n_d^{1/2}, \tag{8}$$

where n_d is the number of examples the domain holds. Eq. 8 keeps the largest domains from crowding out the smallest without discarding their examples, and both backbones we train use the same corpus, objective, and mixture.

Two properties of this supervision let it transfer beyond the worlds it came from. Each target is conditioned on τ, the evidence the first attempt did not have, so the model learns to read a refutation and write the remainder from it rather than to write chains from the goal alone. The states it learns to write are free descriptions of conditions, not symbols from a fixed vocabulary, so what the model learns is a map from refutation to remainder over descriptions, and nothing in that map names the tools, pages, or objects of a particular world.

5 Experiments

We train *WSG* on the corpus of Section 4.1 with Eq. 7, on two open backbones, Gemma4-31B (Team et al., 2026) and Qwen3.6-27B (Qwen Team, 2026), since smaller backbones had difficulty writing a full chain from a long context. We evaluate zero-shot on 7 public agent benchmarks, τ^2-bench (Barres et al., 2025), ScienceWorld (Wang et al., 2022), ToolSandbox (Lu et al., 2025), PlanBench-XL (Liu et al., 2026), REAL (Garg et al., 2026), DeepPlanning (Zhang et al., 2026), and AgentBoard's PDDL (Ma et al., 2024). Every run uses the *SDP* runtime, and in every run the same backbone fills the realizer and the validator. In the *SDP* condition the backbone is prompted for every proposer call, and in the *WSG* condition the same backbone writes the first plan while *WSG* takes every repair call, with budgets and decoding held identical. Each benchmark is scored by its own metric, which Table 3 lists. An aborted episode counts as a failure, and every number is the mean over the benchmark's task set. Implementation details are in Appendix B.

5.1 Training

Both backbones train on the same corpus with the same settings. The corpus holds 238,563 repair examples, and 2% of the worlds are held out with every example they produced. The settings are LoRA rank 32 and alpha 64, learning rate 10^{-4} with cosine decay, sequences packed to 4,096 tokens, an effective batch of 96, and 10 epochs in bf16. Each run uses two H200 GPUs and takes 12 to 16 hours per epoch. The loss on held-out worlds falls steadily on both backbones and flattens within the first 1200 steps, after which training loss on Qwen3.6-27B continues to fall slightly while held-out loss stays flat (Appendix C.1). Every chain the model generates from held-out repair prompts parses as a valid state list from the first evaluation onward.

When the training set included first plans, written without τ, the model overfit faster than on repairs alone and lost the planning it had under prompting. A first plan is conditioned on the goal alone, so it teaches the shape of a chain rather than a response to evidence, and a model trained on shapes overwrites its habit of planning instead of adding a conditional one. A repair does not do this, because it is tied to the τ that produced it. The same collapse appeared with a larger adapter or a

higher learning rate, and it was sharpest when one domain dominated the mixture. The adapter is small, the learning rate is low, and Eq. 8 balances the domains for this reason.

The corpus keeps the repairs each backbone extracted for itself alongside those of the other two models, because runs trained without a backbone's own repairs recovered less often on held-out worlds. The corpus is large and spans 7 domains because a repair is specific to the rule that was broken and the rules differ by world, so a model learns to read a refutation rather than memorize one world's fixes only after seeing refutations of many kinds.

Table 3: Zero-shot performance on 7 agent benchmarks. SDP prompts the backbone for every proposer call, and WSG replaces the calls under τ with the trained model on the same backbone and runtime, with the gain over SDP in parentheses. **Bold** marks the best proprietary result and the best open result per row.

						Gemma4-31B		Qwen3.6-27B	
Benchmark	**Metric**	**Published results**				SDP	WSG	SDP	WSG
τ^2-bench	Accuracy	46.3 Gemini-2.5-Pro	53.7 Sonnet-4	54.3 GPT-4.1	**79.9** GPT-5-Think	42.5	56.3 (+13.8)	68.3	**76.6** (+8.3)
ScienceWorld	Average score	36.4 GPT-4 ReAct	39.2 GPT-4 CoT	45.3 GPT-4 Reflexion	**47.9** GPT-4 Plan-and-Act	35.8	44.4 (+8.6)	36.4	**46.8** (+10.4)
ToolSandbox	Average score	64.3 GPT-4	65.6 GPT-3.5	69.2 Opus-3	**73.0** GPT-4o	64.3	**72.5** (+8.2)	64.7	69.6 (+4.9)
PlanBench-XL	Accuracy	51.9 GPT-5.4	52.2 Gemini-3.5-Flash	63.1 DeepSeek-V4-Flash	**77.1** Gemini-3.1-Pro	38.5	**53.9** (+15.4)	16.1	48.0 (+31.9)
REAL	Accuracy	25.0 o3-mini	35.0 Gemini-2.5-Pro	38.0 o3-pro	**41.0** Sonnet-3.7	27.7	37.3 (+9.6)	22.6	**40.2** (+17.6)
DeepPlanning (Shop)	Match score	50.1 Grok-4.1-Fast	58.6 GPT-5.2	69.1 Gemini-2.5-Pro	**75.8** Sonnet-4.5	48.5	**59.0** (+10.5)	29.1	51.2 (+22.1)
AgentBoard (PDDL)	Success rate	13.3 Haiku-3	40.0 Claude-2	61.7 GPT-4o	**61.7** GPT-4	53.3	58.3 (+5.0)	58.3	**61.7** (+3.4)

5.2 Zero-shot Performance on Agent Benchmarks

WSG raises performance across all 7 benchmarks (Table 3). The gain is largest where the prompted proposer is weakest, 32 points for Qwen3.6-27B on PlanBench-XL, and smallest on AgentBoard, where both backbones already succeed on more than half the tasks before training. On every benchmark *WSG* lands within the range of published proprietary results, and on five of them within a few points of the best, while on PlanBench-XL and DeepPlanning the strongest published systems remain well ahead. Much of this gap comes from the realizer and the validator, which we discuss in Section 6.

Every point of this gain comes from what the agent does after the world has refuted a state, since the two conditions differ only in the calls made under τ, and the logs show what changed there. When the prompted proposer receives a refutation, it often rewrites the refuted state in different words, in 38% of repairs on REAL and 28% on ScienceWorld for Gemma4-31B, and the rest of its chain changes with it. *WSG* keeps the refuted state and changes what surrounds it, rewording it in 19 and 7% of repairs on the same benchmarks, and its rewritten chain overlaps the chain it replaced far more, 0.49 against 0.29 in state overlap. What it inserts is the condition the refutation named. On ToolSandbox, for example, the observation reports a tool call rejected for a parameter below its valid range, the prompted proposer concludes that no reminders exist and closes the task, and *WSG* drops the filter, retrieves the full list, and reaches the goal. The difference is where each proposer looks for the fix. The prompted proposer searches the language of the plan, and *WSG* reads the rule the world has just stated, a parameter range, a button that must be enabled, a room the agent must first be in, and bends the plan to it.

5.3 Recovery after Refutation and Its Cost

WSG adds the most recovery where the prompted proposer recovered least, and Table 4 shows this on the refuted episodes behind the gain in Table 3. Recovery is the share of refuted episodes that still reached the goal. It can exceed the success rate in Table 3 because an episode the world never refutes can still end on a wrong answer that only the benchmark's final check rejects. On DeepPlanning and ScienceWorld the prompted proposer recovered fewer than one refuted episode in seven, and *WSG* raises that share by 66% to 187%. On τ^2-bench and ToolSandbox the prompted proposer already

Table 4: Recovery after refutation and its cost, reported as SDP / WSG. Recovery is the share of episodes that reached the goal after at least one refutation. Steps counts messages per episode, Replans counts proposer calls under τ per episode, and Time is wall-clock seconds per episode.

	Gemma4-31B				Qwen3.6-27B			
Benchmark	Recovery (%)	Steps	Replans	Time (s)	Recovery (%)	Steps	Replans	Time (s)
τ^2-bench	51.0 / 64.0	42.2 / 41.8	5.91 / 5.57	15.6 / 13.9	74.0 / 81.0	38.8 / 42.6	3.57 / 2.47	16.3 / 15.6
ScienceWorld	10.6 / 27.5	73.6 / 43.8	4.19 / 3.07	348 / 223	13.8 / 22.9	73.1 / 40.2	3.89 / 2.23	404 / 215
ToolSandbox	71.3 / 85.1	10.9 / 10.1	2.14 / 1.05	27 / 31	66.5 / 72.7	11.9 / 12.1	2.37 / 1.46	22 / 20
PlanBench-XL	40.2 / 63.1	32.0 / 30.7	1.90 / 4.55	307 / 335	20.8 / 48.4	32.8 / 36.2	1.30 / 4.26	259 / 353
REAL	17.2 / 25.0	33.6 / 27.6	3.94 / 2.43	832 / 376	18.9 / 23.0	41.9 / 28.6	3.92 / 2.01	1077 / 478
DeepPlanning (Shop)	8.7 / 17.2	55.6 / 67.3	6.74 / 4.26	2116 / 1036	6.8 / 19.5	30.8 / 50.0	2.92 / 2.73	1223 / 1511
AgentBoard (PDDL)	20.0 / 26.7	57.2 / 58.1	4.26 / 3.65	455 / 345	24.1 / 25.9	53.2 / 51.5	3.13 / 2.10	551 / 335

recovered more than half of its refuted episodes, and the rise there stays between 9% and 25%. Training on refutations adds the most where prompting could not repair a refusal.

What that recovery costs depends on where the prompted proposer was failing. On 6 of the 7 benchmarks replans per episode fall as recovery rises, because the calls the prompted proposer spent rewording a refuted state are no longer made. Wall-clock time falls where those calls were most expensive, by 55% on REAL with Gemma4-31B and by 47% on ScienceWorld with Qwen3.6-27B. PlanBench-XL is the exception, and there replans more than double on Gemma4-31B and more than triple on Qwen3.6-27B, because the prompted proposer stops after fewer calls where *WSG* keeps rewriting, and those added calls are what buy the largest point rise in recovery in the table. Recovery is cheaper where the prompted proposer wasted calls and dearer where it made too few.

6 Discussion

Most remaining failures come from the realizer and the validator, which *WSG* leaves untrained. Training changes only the repair call, and the realizer and the validator stay prompted. In a hand audit of the failed episodes of DeepPlanning and ScienceWorld, the realizer fails to reach a correctly written state in 45% of them. In another 20% the validator refutes a state that already holds and triggers a repair the run did not need. Recovery on these two benchmarks stays below 30% after training because these failures lie outside the repair call. The same two roles stay prompted on PlanBench-XL and DeepPlanning, where the gap to the strongest published systems is largest. Extending training to the realizer and the validator is therefore the next step.

The worlds that produced the repair corpus also hold supervision for the other two roles. Every call the realizer made is logged with the program's record of whether it was refused and what it changed. These records can train the realizer to reach a written state. The validator needs a label for whether a state holds, and reasoning worlds already supply one by checking each finding against the hidden facts. Writing such a check for the states of the other domains would give the validator the same supervision. The worlds also support reinforcement learning, since the program's goal check defines a reward and the observation in τ gives feedback at the failed state. Whether learning from both signals raises recovery beyond supervised repair remains open.

7 Conclusion

We introduced World State Generator, a planner trained to keep its plan aligned with the world it runs in. When a failure reveals a constraint the plan overlooked, *WSG* uses that evidence to revise the remaining plan while preserving the conditions already achieved. To train it, we built a pipeline that pairs failures with the repairs that followed them in 7 domains of synthetic worlds and collected about 226K trajectories. Each world is checked for goal reachability before agents run, and a repair is kept only when it leads to a state the world certified. Trained only on these worlds, two open backbones near 30B parameters improve end-to-end success over prompting on 7 public benchmarks and reach the range of published proprietary results on all of them. The gains concentrate in recovery after failed states, where the supervision aimed. These results show that verified failure and repair pairs from diverse synthetic worlds can teach a planner to use failure evidence in unseen environments.

AI USE STATEMENT

Generative AI is part of the method of this work. Language models author the worlds of three domains, serve as the validator that judges intermediate states, and run the extraction loop that produces every repair in the corpus, so the training data of this work is synthetic data generated by language models, as detailed in Section 4.1 and Appendix A. In writing the paper, we used generative AI tools to draft sections, to edit the text for readability, and to search for related work. We have reviewed all AI-assisted work. Every sentence was reviewed by the authors, every cited work was read by the authors, and every number reported was computed from our own logs by scripts the authors wrote and verified. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work involves no human subjects and no personal data. The training corpus is synthetic, produced by language models running inside program-enforced environments, and the material that seeds three of its 7 domains comes from public research corpora and a snapshot of English Wikipedia, used under their respective licenses and only as sources of problems and reference answers. No benchmark used for evaluation contributed a world, a task, or a trajectory to the corpus.

REPRODUCIBILITY STATEMENT

The world construction pipeline, admission procedures, extraction budgets, prompts, and corpus filters are specified in Appendix A, with one kept repair from every domain. Training hyperparameters and the held-out loss curves are given in Section 4.2 and Appendix C, and the runtime configuration for every benchmark is given in Appendix B. Full per-benchmark results with the published comparison rows and per-domain profiles of every run are in Appendix C. The code for world generation, extraction, training, and evaluation, the 7-domain corpus, and the trained adapters will be released and provided as supplementary material.

REFERENCES

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. tau2-bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. Fireact: Toward language agent fine-tuning. *arXiv preprint arXiv:2310.05915*, 2023.

Runnan Fang, Shihao Cai, Baixuan Li, Jialong Wu, Guangyu Li, Wenbiao Yin, Xinyu Wang, Xiaobin Wang, Liangcai Su, Zhen Zhang, et al. Towards general agentic intelligence via environment scaling. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 17610–17621, 2026.

Div Garg, Diego Caples, Andis Draguns, Nikil Ravi, Pranav Putta, Naman Garg, Prannay Hebbar, Youngchul Joo, Jindong Gu, Charles London, et al. Real: Benchmarking autonomous agents on deterministic simulations of real websites. *Advances in Neural Information Processing Systems*, 38, 2026.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Nan Duan, Weizhu Chen, et al. Critic: Large language models can self-correct with tool-interactive critiquing. In *International Conference on Learning Representations*, volume 2024, pp. 57734–57811, 2024.

Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. Leveraging pretrained large language models to construct and utilize world models for model-based task planning. *Advances in Neural Information Processing Systems*, 36:79081–79094, 2023.

Jiacheng Guo, Ling Yang, Peter Chen, Qixin Xiao, Yinjie Wang, Xinzhe Juan, Jiahao Qiu, Ke Shen, and Mengdi Wang. Genenv: Difficulty-aligned co-evolution between llm agents and environment simulators. *arXiv preprint arXiv:2512.19682*, 2025.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Mengkang Hu, Pu Zhao, Can Xu, Qingfeng Sun, Jian-Guang Lou, Qingwei Lin, Ping Luo, and Saravan Rajmohan. Agentgen: Enhancing planning abilities for large language model based agent via environment and task generation. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*, pp. 496–507, 2025.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *International conference on learning representations*, volume 2024, pp. 32808–32824, 2024.

Sungheon Jeong, Ryozo Masukawa, Sanggeon Yun, Mahdi Imani, and Mohsen Imani. State-centric decision process. *arXiv preprint arXiv:2605.12755*, 2026.

Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Saldyt, and Anil Murthy. Llms can't plan, but can help planning in llm-modulo frameworks. *arXiv preprint arXiv:2402.01817*, 2024.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, et al. Training language models to self-correct via reinforcement learning. In *International Conference on Learning Representations*, volume 2025, pp. 54523–54549, 2025.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, et al. Tulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 2024.

Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Huang, Kashif Rasul, Longhui Yu, Albert Q Jiang, Ziju Shen, et al. Numinamath: The largest public dataset in ai4maths with 860k pairs of competition math problems and solutions. *Hugging Face repository*, 13(9):9, 2024.

Xiaochuan Li, Ryan Ming, Pranav Setlur, Abhijay Paladugu, Andy Tang, Hao Kang, Shuai Shao, Rong Jin, and Chenyan Xiong. Benchmark test-time scaling of general llm agents. *arXiv preprint arXiv:2602.18998*, 2026.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

Bill Yuchen Lin, Yicheng Fu, Karina Yang, Faeze Brahman, Shiyu Huang, Chandra Bhagavatula, Prithviraj Ammanabrolu, Yejin Choi, and Xiang Ren. Swiftsage: A generative agent with fast and slow thinking for complex interactive tasks. *Advances in Neural Information Processing Systems*, 36:23813–23825, 2023.

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. Llm+ p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*, 2023.

Jiayu Liu, Qihan Lin, Cheng Qian, Rui Wang, Emre Can Acikgoz, Xiaocheng Yang, Jiateng Liu, Zhenhailong Wang, Xiusi Chen, Heng Ji, et al. Planbench-xl: Evaluating long-horizon planning of llm tool-use agents in large-scale tool ecosystems. *arXiv preprint arXiv:2606.22388*, 2026.

Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, et al. Apigen: Automated pipeline for generating verifiable and diverse function-calling datasets. *Advances in Neural Information Processing Systems*, 37:54463–54482, 2024.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, et al. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 1160–1183, 2025.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. Agentboard: An analytical evaluation board of multi-turn llm agents. *Advances in neural information processing systems*, 37:74325–74362, 2024.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Norman G Peterson, Michael D Mumford, Walter C Borman, P Richard Jeanneret, Edwin A Fleishman, Kerry Y Levin, Michael A Campion, Melinda S Mayfield, Frederick P Morgeson, Kenneth Pearlman, et al. Understanding work using the occupational information network (o* net): Implications for practice and research. *Personnel psychology*, 54(2):451–492, 2001.

Qwen Team. Qwen3.6-27B: Flagship-level coding in a 27b dense model, April 2026. URL `https://qwen.ai/blog?id=qwen3.6-27b`.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Xiaoshuai Song, Haofei Chang, Guanting Dong, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. Envscaler: Scaling tool-interactive environments for llm agent via programmatic synthesis. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 8326–8357, 2026.

Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, et al. Os-genesis: Automating gui agent trajectory construction via reverse task synthesis. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5555–5579, 2025.

Gemma Team, Sherif El Abd, Vaibhav Aggarwal, Robin Algayres, Alek Andreev, Olivier Bachem, Ian Ballantyne, Cormac Brick, Victor Cărbune, Michelle Casbon, et al. Gemma 4 technical report. *arXiv preprint arXiv:2607.02770*, 2026.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. musique: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022.

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 (volume 1: Long papers)*, pp. 2609–2634, 2023.

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*, pp. 11279–11298, 2022.

Yiheng Xu, Dunjie Lu, Zhennan Shen, Junli Wang, Zekun Wang, Yuchen Mao, Caiming Xiong, and Tao Yu. Agenttrek: Agent trajectory synthesis via guiding replay with web tutorials. In *International Conference on Learning Representations*, volume 2025, pp. 79822–79843, 2025.

Hang Yan, Xinyu Che, Fangzhi Xu, Qiushi Sun, Zichen Ding, Kanzhi Cheng, Jian Zhang, Tao Qin, Jun Liu, and Qika Lin. Tide: Trajectory-based diagnostic evaluation of test-time improvement in llm agents. *arXiv preprint arXiv:2602.02196*, 2026.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2369–2380, 2018.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*, 2023.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. Agenttuning: Enabling generalized agent abilities for llms. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 3053–3077, 2024.

Yinger Zhang, Shutong Jiang, Renhao Li, Jianhong Tu, Yang Su, Lianghao Deng, Xudong Guo, Chenxu Lv, and Junyang Lin. Deepplanning: Benchmarking long-horizon agentic planning with verifiable constraints. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7377–7407, 2026.

Weikang Zhao, Xili Wang, Chengdi Ma, Lingbin Kong, Zhaohua Yang, Mingxiang Tuo, Xiaowei Shi, Yitao Zhai, and Xunliang Cai. Mua-rl: Multi-turn user-interacting agent reinforcement learning for agentic tool use. *arXiv preprint arXiv:2508.18669*, 2025.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, et al. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022.

A CORPUS CONSTRUCTION

A.1 WORLDS AND THEIR MATERIAL

The corpus is built from 47,946 worlds in 7 domains, produced in three ways (Table 5). Tool and constraint worlds are sampled by a program with no language model and no outside data. Physical, web, and reasoning worlds are written by a language model and then verified by program. Physical worlds are written from occupation and task descriptions in O*NET (Peterson et al., 2001), and web worlds run over 12 crawled snapshots of English Wikipedia with 5,000 pages each. Software and math worlds adapt public corpora and take from them only a problem and its gold answer. Software worlds use the questions of HotpotQA (Yang et al., 2018) and MuSiQue (Trivedi et al., 2022) over an index of 5.2M Wikipedia documents, and math worlds use the problems of NuminaMath-CoT (Li et al., 2024). Every chain and every repair in the corpus is produced by the loop, and no trajectory is copied from an existing dataset.

Table 5: How the worlds of each domain are produced, what they are made from, and their size. Entities and actions are means per world with the range in parentheses.

Domain	Produced by	Material	Worlds	Episodes	Entities per world	Actions per world
Physical	language model, verified by search	143 occupation and task seeds from O*NET	4,456	31,775	10.6 objects (3 to 16)	11.7 (1 to 25)
Software	adapter over public corpora	HotpotQA and MuSiQue over full Wikipedia	10,381	31,659	an index of 5.2M documents	3
Web search	language model over a crawl	English Wikipedia snapshots of 12 topics	11,074	34,923	a snapshot of 5,000 pages	4
Tool use	program sampler	16 entity types, synonym pools for tool names	1,000	33,829	50.8 rows (17 to 78)	30.6 tools (22 to 36)
Constraint	program sampler	candidates, prices, constraints, hidden policies	1,000	32,657	25.0 candidates (16 to 36)	8.0 tools (7 to 9)
Reasoning	language model	a subject text and hidden facts written together	9,222	30,955	5.8 hidden facts (5 to 10)	2
Math	adapter over a public corpus	NuminaMath-CoT	10,813	30,518	one problem	2
Total			47,946	226,316		

The domains differ in what a world asks for. A physical world combines 2 to 7 of 8 rule types, which are precondition, capacity, exclusive use, irreversible action, session gate, chained transition, hidden observation, and stochastic outcome. Its goal check holds 4.5 conditions on average, and its reference route takes 4 to 36 actions with 14.6 on average. Worlds with long routes are not written directly, and 2,917 of the 4,456 are two verified worlds composed in series, where the second stage becomes reachable only when the goal of the first holds. A tool world spreads its rows over 3 or 4 of the 16 entity types, its goal takes one of 8 shapes such as a chain, a parallel task, or a refused operation, and its reference route takes 1 to 14 calls. A constraint world holds 2 or 3 slots with about 10 candidates each, and it takes one of 4 shapes, explicit, hidden, coupled, or infeasible. A reasoning world is one of 4 task kinds over 17 kinds of subject, its subject text runs to about 140 words, and its verdict is decided on 5 aspects drawn from 20. A web goal names its target page by 2 to 4 conditions, and a software goal needs 2 to 4 retrieval hops.

The program samples tool and constraint worlds in small numbers, and the loop runs each about 33 times, while every other world is run about 3 times. Three design choices keep a world from being solved by reading the goal. In tool worlds, the agent starts with system tools only and discovers the rest at run time, tool names are drawn from synonym pools so the goal never names the tool that solves it, and every list returns ten rows per page. In hidden, coupled, and infeasible constraint worlds the listing alone does not reveal the rule that decides the answer. In web worlds some target pages are reachable only by following links.

A.2 ADMISSION BY DOMAIN

Admission proves Eq. 5 for every world before any agent runs in it, and the procedure is set by what the domain allows.

- **Physical.** Breadth-first search runs from the initial configuration over every sequence of primitive actions. The world is admitted if the goal is reached, the shortest route is not trivial, and every required state lies on all routes. A world composed of two stages is searched again as a whole.
- **Tool use.** A reference route is executed against the database and must pass the goal check, and a deliberately wrong run must fail it. Every parameter the route needs must be producible by a tool in the same world.

- **Constraint.** Every combination of one candidate per slot is enumerated. The world is admitted if exactly one combination satisfies every rule including the hidden policies. For hidden and coupled shapes the cheapest combination that satisfies only the explicit constraints must violate a hidden policy, so the world cannot be solved from the listing alone. An infeasible world is admitted only if no combination satisfies every rule.
- **Web search.** The gold page must be reachable by search or by following links from a search result, the answer must appear on it, and exactly one page in the snapshot must satisfy every condition of the goal.
- **Software.** Every gold document must be in the index and found by a search on its title, and the fact each hop needs and the gold answer must appear in those documents.
- **Reasoning.** A verdict written from the subject alone must differ from the verdict written with the hidden facts.
- **Math.** The reference answer must be one a program can check for equivalence, and problems that ask for a proof, refer to a figure, or list several sub-questions are discarded.

In reasoning and math worlds reachability is not in question, so admission checks instead that the goal is not already decided and that the check can run.

Admission removes a large share of what is generated. Of 30,622 physical worlds written by the language model, 1,652 were admitted, which is 5.4%, and of 3,722 two-stage compositions 2,919 were admitted. Of 35,195 web worlds 22,496 were admitted, and most of the rest named a target that did not satisfy its own conditions or did not satisfy them uniquely. Of 110,385 software candidates 88,609 were admitted, and most of the rest held a gold document the search tool could not find by its title. The episodes of Table 5 run on admitted worlds only, and in the software, web, and math domains on a sample of them.

A.3 The Extraction Loop

Three models, Qwen3-32B, Qwen3.6-27B, and Gemma4-31B, ran the *SDP* loop inside the admitted worlds. In every episode one model fills all three roles, the proposer, the realizer, and the validator. Table 6 gives the budgets. b is the number of tries the realizer has for one state. A try continues from wherever the previous one left the world, a success within the budget costs no repair, and the loop never rolls the world back. The proposer and the realizer sample at temperature 0.7 with top-p 0.95 and top-k 20 in every domain except web search, where they decode greedily, and the validator decodes greedily in every domain.

Table 6: Extraction budgets by domain.

Domain	Tries per state b	Repairs per episode
Physical	3	10
Software	2	5
Web search	3	5
Tool use	1	6
Constraint	1	6
Reasoning	1	6
Math	1	6

A repair call receives the goal, the certified prefix, the refuted state, the validator's description o_f, and the program's record of the world. Two items are added when they exist. If the same state has already failed from the same prefix, the earlier repairs, at most three, are shown with a note that they did not work. If the run has returned to the same state more often than b allows, a note says that an equivalent repair will keep failing. The system prompt of the repair call in the physical domain is reproduced below, and the other domains use the same contract with domain-specific wording. Its instruction to insert a missing prerequisite as a separate state is part of what the extraction models were told, so it is part of what the corpus teaches.

```
ROLE: PROPOSER (REPLAN) in a State-centric Decision Process.
Your earlier decomposition failed at one step. You are given: the GOAL,
```

```
the predicates already ACHIEVED, the predicate that FAILED, and a FACTUAL
DESCRIPTION of the current situation from the validator. Reason about
what that description implies, then produce a REVISED continuation: the
remaining chain of states from the current situation to the goal.
- Do not restate achieved predicates; output only the continuation.
- Each predicate is a checkable world state, not an action.
- CRITICAL GRANULARITY (same as the initial plan): one predicate = ONE
  primitive action. If the failed predicate needs the agent to BE
  somewhere or HOLD something first, the continuation MUST begin with
  that as its OWN predicate, THEN the change.
- Use the EXACT entity ids shown in CURRENT WORLD STATE.
- If the same predicate keeps failing, the missing piece is usually a
  PREREQUISITE state; insert those as separate earlier predicates
  instead of rephrasing the failed one.
- Output ONLY JSON: {"chain": ["next state", ...]}
```

When a chain completes but the program's goal check fails, tool, constraint, reasoning, and math worlds issue one more refutation, written by the program. In tool and constraint worlds it names the condition found unmet, in math worlds it reports the rejected answer, and in reasoning worlds it reports that no verdict or a wrong verdict was reached. In tool and constraint worlds the proposer may answer such a call with a sentinel, IMPOSSIBLE or INFEASIBLE, which the program grades like any other answer. An episode that has performed an irreversible act leaving the goal unreachable is stopped at that point.

A.4 Corpus Filters and Lengths

A repair enters the corpus only if it passes every test below that applies to its domain.

- **A state was certified.** The run certified at least one state of the repair. This test applies in every domain. In the physical, software, web, tool, and constraint domains the certified state must be the first state of the repair.
- **Goal refutations.** A repair written from the program's goal refutation is kept only if the episode went on to certify the goal.
- **No restated repair.** The repair is not equal, as a set of normalized state strings, to an earlier repair that failed from the same prefix. This test applies in the physical, software, web, tool, and constraint domains.
- **States, not actions.** No state opens with an action verb such as search, click, or open. The sentinels are exempt.
- **Not aborted.** The call did not abort on a runner-side condition such as an unparseable response.
- **Sentinel graded.** A sentinel answer is kept only if the program graded it correct.
- **Not a duplicate.** The triple of system prompt, user prompt, and response is not an exact duplicate of one already kept.

The tool and web domains pass through one more deterministic filter before these tests, and its removals are counted in the Kept column of Table 2. In the tool domain an episode is removed if the same normalized state recurs three or more times, if the proposer concluded IMPOSSIBLE in a world that was solvable, if a single-target goal was passed by submitting three or more distinct items, or if a call aborted on a parse error. In the web domain a state is removed if it does not follow the state grammar, if it leaks the answer as an example, or if it introduces a proper noun that appears in neither the goal nor any earlier observation.

Table 7 gives the lengths of plans, repairs, and episodes per domain. A kept repair is shorter than a first plan in every domain, because a repair writes only the states that remain after the certified prefix. Physical worlds hold the longest chains and the longest episodes, and constraint worlds the shortest repairs, where a repair is most often the single state of a submitted selection.

Table 7: Lengths in states per domain. First plan is the chain the proposer wrote on the first call of an episode, over all 226,316 episodes. Kept repair is the target chain C of a training example, over all 207,660 examples. Episode is the number of states the run attempted.

	First plan		Kept repair		Episode	
Domain	Mean	Max	Mean	Max	Mean	Max
Physical	13.9	51	10.5	53	18.5	121
Software	4.2	18	3.4	33	6.8	24
Web search	3.5	12	2.2	17	4.8	37
Tool use	5.7	12	2.3	23	8.5	76
Constraint	4.7	12	1.2	5	6.0	18
Reasoning	5.1	8	3.4	7	8.2	27
Math	4.7	29	4.0	27	8.3	80
All	5.9	51	4.9	53	8.7	121

A.5 One Repair from Each Domain

Each entry gives the goal, the refuted state, the observation o_f, and the repair, taken from a kept episode that went on to certify the goal.

Physical. *Goal.* Log into the system, open the notebook, and record the experiment. *Refuted.* The notebook_1 is open. o_f. The notebook_1 exists, but its open attribute is false. *Repair.* (1) The agent is at notebook_1. (2) The notebook_1 is open. (3) The experiment_1 is recorded.

Software. *Goal.* Who was the ninth governor-general in the country where Mondrabet is located? *Refuted.* The record contains a source about the country where Mondrabet is located. o_f. The record contains a search result about Mondrabet, but no source about the country where Mondrabet is located (Papua New Guinea). *Repair.* (1) The record contains a source about Papua New Guinea. (2) The ninth governor-general in Papua New Guinea is known.

Web search. *Goal.* Who is the ancient Greek philosopher born in Stagira, who taught Alexander the Great and lived from 384 BC? *Refuted.* A page satisfying the conditions of an ancient Greek philosopher born in Stagira is open. o_f. The open page does not satisfy the conditions of the goal, it only lists names and does not confirm that the page contains information about an ancient Greek philosopher born in Stagira. *Repair.* (1) A page satisfying the conditions of an ancient Greek philosopher born in Stagira and who taught Alexander the Great is open. (2) The information that the philosopher lived from 384 BC is confirmed from the record.

Tool use. *Goal.* A new project named 'Kraken' needs to exist in the system, with budget 41816.55. *Refuted.* The project resource type is accessible in the system. o_f. The project resource type is not accessible in the system as no relevant tools or records were found to indicate its presence. *Repair.* (1) The 'Kraken' project with budget 41816.55 is created in the system.

Constraint. *Goal.* 13 people, one Friday, budget $540 all-in, put together the activity and transport. *Refuted.* The full records for candidates that satisfy the group size, day, and estimated budget constraints have been read. o_f. The record contains listings for activity and transport with fields for capacity, days, and price, but the full records including policy fields are only available through a separate read. *Repair.* (1) A plan naming activity_29769, transport_26023 has been submitted.

Reasoning. *Goal.* Below is a migration plan for a legacy system. Will it work as planned? *Refuted.* A position on the technical compatibility between the legacy HRIS-1.0 data schema and the new modular API architecture has been reached and argued. o_f, *written by the program.* This does not decide the answer, the material here is a side point, not something the verdict can turn on. *Repair.* (1) A position on the adequacy of the testing schedule relative to the complexity of the tax and compliance integrations in Phase 3 has been reached and argued. (2) The overall verdict has been reached and argued.

Math. *Goal.* Answer the question with the exact value asked for. *Refuted.* The subtraction $3.75 - 0.48$ has been performed. o_f. The subtraction has not been performed yet, as the problem has not been solved step by step. *Repair.* (1) The value of $3.75 - 0.48$ has been submitted as the answer.

B Implementation Details

Every reported result comes from the *SDP* runtime, and the two conditions are defined by which model answers a proposer call under τ. In the *SDP* condition the prompted backbone answers every proposer call. In the *WSG* condition the same backbone writes the first plan, and the backbone with the *WSG* adapter answers every call under τ. In both conditions the same backbone fills the realizer and the validator with the same prompts and decoding settings. An episode succeeds when the benchmark's evaluator returns full reward, and an aborted episode counts as a failure.

B.1 Runtime

Each role in the runtime sees only what its decision needs. On the first call the proposer receives the goal, the current observation, and the tool list. On a call under τ it receives the goal, the certified states, the refuted state, o_f, the recent tool results, and the earlier repairs that failed at the same state. A chain is accepted only if every state is a condition, so a chain with a state that begins with an action verb or has the form of a tool call is sent back, and the episode is aborted when no valid chain is returned after three tries. The realizer never sees the goal. It receives one target state, the observation, the tool list, the record of recent calls, and the calls it already tried for this state, and it returns one tool call. A call repeated under an unchanged observation is rejected before it reaches the environment. After every executed call the validator receives the target state, the results of the current attempt, and the observation. It returns whether the state holds and a factual description of what it found, and its prompt forbids suggesting a fix. A state is refuted when it is not certified within its step allowance, and the validator's last description becomes o_f.

An episode ends when the chain is exhausted, when the number of repair calls reaches its cap, or when the environment ends it. The benchmark's evaluator then scores the final environment outside the loop. Proposer and validator calls decode greedily, and realizer calls sample at temperature 0.7.

B.2 Benchmarks

Table 8: Task sets, scores, and budgets for the 7 benchmarks. Steps per state is how many tool calls the realizer may make for one state before that state is refuted. Max replans is how many times the proposer may be called under τ in one episode. Max steps is how many tool calls one episode may make in total.

Benchmark	Tasks	Task set	Score	Steps per state	Max replans	Max steps
τ^2-bench	278	airline 50, retail 114, telecom 114	task reward, one trial per task	10	10	200
ScienceWorld	270	test split, 30 task types, up to 10 variations	environment score	10	10	200
ToolSandbox	1032	all scenarios	milestone similarity, zero on a minefield	10	10	30
PlanBench-XL	327	retail	answer accuracy	10	10	100
REAL	112	action tasks	state assertions	10	10	50
DeepPlanning (Shop)	150	levels 1 to 3	cart match score	10	10	80
AgentBoard (PDDL)	60	gripper, blockworld, barman, tyreworld	success rate	10	10	100

Every benchmark is scored by its own evaluator, and Table 8 gives the task set and the budgets of each. ScienceWorld follows the protocol of SwiftSage (Lin et al., 2023), which evaluates the 30 task types on the test split with a bounded number of variations per task. τ^2-bench and ToolSandbox include a simulated user, and in our runs the backbone under evaluation plays that user. τ^2-bench also judges some tasks with language assertions, and the same backbone evaluates them. PlanBench-XL leaves unanswered queries out of its denominator, and we report accuracy over all 327 queries instead. REAL is evaluated on its action tasks, whose outcome is decided by assertions on the final application state. AgentBoard is evaluated on its four PDDL domains.

B.3 Prompts and Serving

Each role is a separate call with its own system prompt, and every reply is parsed as JSON. The proposer prompt asks for an ordered list of states and forbids tool names and actions as list elements.

The realizer prompt asks for one tool call toward the target state. The validator prompt asks whether the target state holds and for a factual description of what it found, and it forbids suggesting a fix. A call under τ uses the input layout below, which keeps the fields of the training input, so *WSG* is called at evaluation in the form it was trained on.

```
GOAL: <goal>
ACHIEVED (do not repeat):  [<certified states>]
FAILED PREDICATE: <refuted state>
OBSERVATION: <o_f>
PREVIOUS FAILED REVISIONS: <earlier repairs that failed at this
state, if any>
Recent tool results:  <latest calls and their results>
Revise.
Reply:  ["<state>", ..., "<goal condition>"]
```

Both backbones run locally in bf16 and are served with vLLM. In the *WSG* condition the LoRA adapter from the last training checkpoint is loaded on the same backbone, and calls under τ are routed to it while every other call goes to the backbone without the adapter. Proposer and validator calls decode greedily. Thinking mode is disabled for every call.

C Experiment Details

C.1 Training

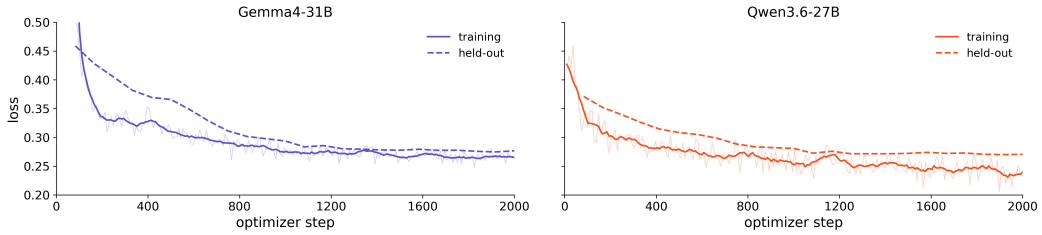

Figure 2: Training (solid) and held-out (dashed) loss over the first 2,000 optimizer steps for Gemma4-31B and Qwen3.6-27B. Faint lines are per-step training loss and solid lines are smoothed. One step is one update at an effective batch of 96.

Both backbones converge to nearly the same held-out loss early in training. Figure 2 shows the loss on the held-out worlds, which are the 2% of worlds removed from the corpus together with every example they produced. Gemma4-31B starts higher than Qwen3.6-27B, at about 0.46 against 0.37, and the two held-out curves meet at about 0.27 by step 1,100 and stay there through step 2,000. Training loss on Gemma4-31B stays within about 0.01 of its held-out loss, while on Qwen3.6-27B it keeps falling to about 0.24 after the held-out loss has flattened. Chains generated from held-out repair prompts parse as valid state lists from the first evaluation onward on both backbones.

C.2 Experiment Results and Comparison

C.2.1 τ^2-bench

τ^2-bench evaluates a customer-service agent that must both call tools and hold a conversation with a simulated user, across three domains with 114 retail, 50 airline, and 114 telecom tasks. Table 9 lists every published result we found alongside our four runs, and the four proprietary entries of Table 3 are drawn from it. On the task-weighted average *WSG* on Qwen3.6-27B is second only to GPT-5-think among all listed systems, and on telecom it is the only system other than GPT-5-think above 60. The gain from training is concentrated in one domain per backbone, telecom for Qwen3.6-27B, from 64.0 to 85.1, and airline for Gemma4-31B, from 60.0 to 64.0 with the larger move in retail from 43.0 to 71.6, while retail on Qwen3.6-27B falls slightly, from 76.3 to 74.6. The published rows

Table 9: Full results on τ^2-bench. Published rows are taken from (Fang et al., 2026; Zhao et al., 2025) under their function-calling, non-thinking settings. Average is the task-weighted mean over the three domains, with 114 retail, 50 airline, and 114 telecom tasks.

Agent	Retail	Airline	Telecom	Average
Published, function calling, non-thinking				
Qwen3-8B Non-thinking	41.0	12.5	19.1	26.89
Qwen3-8B Cold-start	31.4	16.0	9.0	19.44
MUA-RL-8B	49.8	19.0	21.8	32.78
Qwen3-14B Non-thinking	43.1	14.8	29.9	32.60
Qwen3-14B Cold-start	53.7	24.0	23.5	35.97
MUA-RL-14B	66.0	38.0	33.4	47.60
Qwen3-32B Non-thinking	50.2	23.5	24.8	34.98
Qwen3-32B Cold-start	58.2	31.1	19.3	37.37
MUA-RL-32B	67.3	45.4	28.3	47.37
GPT-4o-2024-11-20	67.3	46.9	24.1	45.92
GPT-4.1	70.2	53.0	38.9	54.27
DeepSeek-V3-0324	64.7	37.0	32.9	46.68
Qwen3-235B-A22B Non-thinking	64.9	36.0	24.6	43.18
Qwen3-30B-A3B Non-thinking	31.6	18.0	18.4	23.74
Qwen3-4B Non-thinking	28.1	12.0	17.5	20.86
Gemini-2.5-Pro	56.0	67.5	27.2	46.26
Claude-Sonnet-4	54.0	67.5	47.4	53.72
GPT-o3	64.8	80.2	58.2	64.86
GPT-o4-mini	56.0	70.2	46.5	54.66
GPT-5-think	62.6	81.1	96.7	79.91
Ours, SDP runtime				
SDP Gemma4-31B	43.0	60.0	34.2	42.45
WSG Gemma4-31B	71.6	64.0	37.5	56.25
SDP Qwen3.6-27B	76.3	60.0	64.0	68.32
WSG Qwen3.6-27B	74.6	62.0	85.1	76.64

were produced under their own scaffolds and user simulators, so the table places our runs among reported results rather than under one shared protocol.

Table 10: Per-domain profile of our four runs on τ^2-bench. Succ. is the success rate in percent, Replans and Msgs are means per episode, and Exh. is the share of episodes in percent that spent the full replan budget of 10.

	Retail				Airline				Telecom			
Run	Succ.	Replans	Msgs	Exh.	Succ.	Replans	Msgs	Exh.	Succ.	Replans	Msgs	Exh.
SDP Gemma4-31B	43.0	5.16	17.2	14	60.0	5.52	26.4	22	34.2	7.16	59.8	51
WSG Gemma4-31B	71.6	3.69	27.3	4	64.0	5.56	27.2	18	37.5	6.80	58.0	41
SDP Qwen3.6-27B	76.3	3.40	32.8	31	60.0	5.40	30.0	20	64.0	2.90	48.6	8
WSG Qwen3.6-27B	74.6	2.00	29.4	13	62.0	4.60	28.9	12	85.1	2.10	61.9	24

Table 10 shows that the gain arrives in two forms, and each domain shows one of them. In retail on Qwen3.6-27B success is unchanged while replans per episode fall from 3.4 to 2.0 and the share of episodes that exhaust the replan budget falls from 31 to 13%, so *WSG* reaches the same outcomes with fewer repairs. In telecom on Qwen3.6-27B and in retail on Gemma4-31B the outcome itself moves, from 64.0 to 85.1 and from 43.0 to 71.6, and in both cases episodes grow longer, from 48.6 to 61.9 and from 17.2 to 27.3 messages, because the prompted proposer ended many episodes early and *WSG* carries them on to the goal. Telecom on Gemma4-31B changes least and remains the hardest cell for that backbone, with over 40% of episodes spending the full budget under both proposers, and airline moves by a few points on both backbones. Recovery is cheaper where the prompted proposer wasted repairs and dearer where it made too few.

Table 11: Full results on ScienceWorld, average score by task length.

Agent	Short	Medium	Long	Overall
Published, GPT-4 backbone				
SayCan	43.83	36.58	23.65	33.82
ReAct	48.79	44.01	21.07	36.43
CoT	49.54	47.87	23.09	39.23
Reflexion	71.47	35.43	30.17	45.34
EvoAgent	48.67	36.17	11.38	30.42
Plan-and-Act	60.52	46.43	34.77	47.86
Ours, SDP runtime				
SDP Gemma4-31B	53.4	44.2	15.7	35.8
WSG Gemma4-31B	61.9	52.2	24.9	44.4
SDP Qwen3.6-27B	41.5	49.5	20.2	36.4
WSG Qwen3.6-27B	52.7	59.8	30.1	46.8

Table 12: Per-length profile of our four runs on ScienceWorld. Score is the average task score, Succ. is the share of tasks in percent that reached full score, and Replans and Sec are means per episode.

	Short				Medium				Long			
Run	Score	Succ.	Replans	Sec	Score	Succ.	Replans	Sec	Score	Succ.	Replans	Sec
SDP Gemma4-31B	53.4	30.0	2.76	277	44.2	10.1	4.70	384	15.7	2.9	4.68	361
WSG Gemma4-31B	61.9	31.4	2.29	200	52.2	29.3	3.19	219	24.9	9.8	3.48	242
SDP Qwen3.6-27B	41.5	14.3	2.70	310	49.5	23.2	3.99	411	20.2	4.9	4.61	461
WSG Qwen3.6-27B	52.7	31.4	1.49	175	59.8	39.4	1.56	154	30.1	10.8	3.38	301

C.2.2 SCIENCEWORLD

WSG raises the score at every task length on both backbones, and on Qwen3.6-27B it reaches the level of the strongest published GPT-4 agent. The largest relative gain is on long tasks, where success rises from 2.9 to 9.8% on Gemma4-31B and from 4.9 to 10.8 on Qwen3.6-27B. The gain costs nothing in repairs. Replans per episode fall in every length and backbone pair, most on Qwen3.6-27B medium tasks from 3.99 to 1.56, and wall-clock time falls with them.

C.2.3 TOOLSANDBOX

WSG on Gemma4-31B reaches 72.5 on the average score, within half a point of GPT-4o, the strongest published system, and *WSG* on Qwen3.6-27B passes Claude-3-Opus. The gain is spread across the scenario categories and the tool augmentations rather than concentrated in one, and on Gemma4-31B it exceeds GPT-4o under every scrambled augmentation, where the tool names, descriptions, or argument types no longer say what the tool does and the proposer must read what a call returns. Two cells run against the trend. Multiple user turn remains the weakest category for both backbones, and on Qwen3.6-27B argument type scrambling falls from 66.2 to 50.6 while every other column rises.

Table 14 shows that the gain on ToolSandbox is bought with fewer repairs, not more. Replans per scenario fall by half on both backbones, from 2.14 to 1.05 on Gemma4-31B and from 2.37 to 1.46 on Qwen3.6-27B, and fall in every scenario category and every augmentation except one. The largest moves are on single tool call, where replans drop below 0.5 and the share of perfect scenarios rises from 5 to over 20% on both backbones, and on state dependency, where a call must wait for the state an earlier call creates and replans drop from over 3 to under 2. The one exception is argument type scrambling on Qwen3.6-27B, where replans rise to 3.66 and time doubles while the score falls to 50.6, so the drop in that cell is a loop the trained proposer enters and the prompted one does not.

C.2.4 PLANBENCH-XL

WSG on Gemma4-31B reaches 53.9% accuracy, above GPT-5.4 and Gemini-3.5-Flash and behind only DeepSeek-V4-Flash and Gemini-3.1-Pro, and on Qwen3.6-27B accuracy triples from 16.1 to

Table 13: Full results on ToolSandbox. Avg is the average score over all scenarios. Scenario categories are single tool call (STC), multiple tool call (MTC), single user turn (SUT), multiple user turn (MUT), state dependency (SD), canonicalization (C), and insufficient information (II). Tool augmentations are 0, 3, and 10 distraction tools (DT), all tools (AT), tool name scrambled (TNS), tool description scrambled (TDS), argument description scrambled (ADS), and argument type scrambled (ATS).

		Scenario categories							**Tool augmentations**							
Agent	**Avg**	STC	MTC	SUT	MUT	SD	C	II	0 DT	3 DT	10 DT	AT	TNS	TDS	ADS	ATS
Published																
GPT-4o-2024-05-13	73.0	87.8	80.1	84.2	74.7	84.0	76.6	42.0	75.1	75.0	74.6	72.6	72.4	69.3	73.0	71.9
Claude-3-Opus-20240229	69.2	83.5	70.0	74.5	67.2	74.5	71.1	57.3	68.3	68.6	70.0	67.5	70.8	71.5	65.8	71.1
GPT-3.5-Turbo-0125	65.6	93.4	73.9	81.8	66.6	82.6	70.4	22.3	67.3	63.2	67.0	65.4	63.9	64.3	66.7	66.9
GPT-4-0125-Preview	64.3	89.1	69.0	74.4	68.6	69.2	65.2	33.6	66.8	62.5	64.0	65.1	69.7	64.4	58.1	63.5
Claude-3-Sonnet-20240229	63.8	82.1	66.2	69.1	69.7	84.5	65.5	44.2	67.2	64.5	63.2	58.8	63.7	61.9	62.5	68.7
Gemini-1.5-Pro-001	60.4	82.6	49.8	63.1	37.3	70.5	51.6	76.2	63.3	63.1	60.8	59.8	62.2	60.5	58.7	54.4
Claude-3-Haiku-20240307	54.9	80.9	54.2	64.3	46.0	69.5	54.4	39.4	56.0	56.9	54.1	52.2	56.6	54.1	54.5	55.1
Gemini-1.0-Pro	38.1	68.7	21.6	36.5	14.6	39.3	18.2	65.5	38.2	39.5	41.9	37.7	40.1	35.3	36.7	34.9
Hermes-2-Pro-Mistral-7B	31.4	63.3	18.3	29.9	18.6	27.1	19.9	48.3	33.1	31.9	30.6	28.3	31.8	31.0	32.6	32.2
Mistral-7B-Instruct-v0.3	29.8	48.1	9.5	20.1	7.9	19.5	6.1	76.8	30.5	30.2	24.7	27.1	32.0	30.7	32.8	30.1
C4AI-Command-R-v01	26.2	52.6	12.7	23.0	12.7	3.1	18.0	47.8	24.8	27.9	25.6	23.3	25.0	25.6	28.7	28.3
Gorilla-Openfunctions-v2	25.6	36.2	8.2	15.1	9.3	0.0	8.9	69.2	25.5	27.5	26.1	18.6	24.5	27.1	26.8	28.6
C4AI-Command-R+	24.7	57.2	13.6	24.3	15.2	4.0	19.4	35.3	23.4	27.2	24.9	23.5	21.7	27.6	24.8	24.8
Ours, SDP runtime																
SDP Gemma4-31B	64.3	76.4	64.9	75.6	44.7	67.3	63.5	55.1	64.6	64.5	63.5	63.5	63.8	66.1	64.1	64.7
WSG Gemma4-31B	72.5	92.5	73.5	86.1	53.5	80.9	73.9	56.2	74.9	72.2	71.1	73.5	71.5	71.5	71.3	73.2
SDP Qwen3.6-27B	64.7	76.1	66.2	73.0	55.0	68.6	61.7	53.5	68.5	65.1	62.6	60.9	64.0	65.3	67.4	66.2
WSG Qwen3.6-27B	69.6	87.5	69.9	79.2	57.3	73.0	69.1	57.5	77.1	68.7	70.8	66.6	69.9	71.7	75.9	50.6

Table 14: Per-scenario profile of our four runs on ToolSandbox. Score is the average score, Perf. is the share of scenarios in percent that scored full marks, and Replans and Sec are means per scenario.

	SDP Gemma4-31B				**WSG Gemma4-31B**				**SDP Qwen3.6-27B**				**WSG Qwen3.6-27B**			
Scenario	Score	Perf.	Replans	Sec	Score	Perf.	Replans	Sec	Score	Perf.	Replans	Sec	Score	Perf.	Replans	Sec
Single tool call	76.4	5.1	1.24	16	92.5	22.4	0.26	16	76.1	5.1	1.06	9	87.5	21.1	0.44	8
Multiple tool call	64.9	15.5	2.19	30	73.5	14.8	1.02	33	66.2	9.0	2.33	24	69.9	9.6	1.48	24
Single user turn	75.6	17.3	2.14	28	86.1	20.5	0.88	29	73.0	10.2	2.02	20	79.2	14.2	1.18	18
Multiple user turn	44.7	4.9	1.74	27	53.5	4.9	0.88	33	55.0	3.6	2.33	26	57.3	5.4	1.57	27
State dependency	67.3	4.7	3.19	43	80.9	8.9	1.68	38	68.6	1.6	3.10	35	73.0	3.2	1.97	28
Canonicalization	63.5	17.3	2.31	33	73.9	17.2	0.92	35	61.7	10.5	2.26	26	69.1	13.9	1.32	25
Insufficient information	55.1	50.9	2.55	23	56.2	51.8	1.68	33	53.5	49.1	3.28	21	57.5	53.1	2.10	20
0 distraction tools	64.6	21.3	2.00	24	74.9	24.8	0.95	28	68.5	21.3	2.18	20	77.1	24.8	0.97	16
3 distraction tools	64.5	22.7	2.15	26	72.2	23.6	1.08	31	65.1	17.8	2.39	21	68.7	21.5	1.68	22
10 distraction tools	63.5	21.3	2.17	28	71.1	24.8	1.02	32	62.7	15.0	2.39	22	70.8	19.4	1.14	18
All tools	63.5	19.7	2.21	31	73.5	24.0	1.02	31	60.9	13.4	2.44	24	66.6	14.7	1.19	18
Tool name scrambled	63.8	22.0	2.06	25	71.5	23.3	1.12	32	64.0	17.3	2.62	21	69.9	20.2	1.19	18
Tool description scrambled	66.1	22.8	2.27	27	71.5	23.3	0.94	28	65.3	18.9	2.38	23	71.7	19.4	1.16	18
Argument description scrambled	64.1	21.3	2.20	26	71.3	24.0	1.11	31	67.4	18.9	2.31	21	75.9	21.9	1.16	17
Argument type scrambled	64.7	23.6	2.09	26	73.2	23.3	1.09	31	66.2	18.1	2.36	21	50.6	22.5	3.66	42
Overall	64.3	21.9	2.14	27	72.5	23.9	1.05	31	64.7	17.3	2.37	22	69.6	20.7	1.46	20

48.0. Precision on the executed ground-truth datatypes rises with it on both backbones, from 76.1 to 82.7 and from 66.8 to 76.4, so the trained proposer reaches more of the datatypes the task needs rather than more in total, and the search-to-call ratio falls on both, most on Qwen3.6-27B from 15.2 to 4.9.

C.2.5 REAL

WSG lifts both backbones into the range of the strongest published systems, Qwen3.6-27B from 22.6 to 40.2, within a point of Claude-3.7-Sonnet, and Gemma4-31B from 27.7 to 37.3, at the level of o3-pro. Under prompting both backbones sat between DeepSeek-V3 and o3-mini, so training moves each past every published system but the top two.

Table 15: Full results on PlanBench-XL. EGT Prec. is the share of unique datatypes produced by executed tool calls that belong to the ground-truth datatype set. Mean EDT is the average number of new datatypes found through tool retrieval beyond the initial input datatypes of each query. S/C ratio is the ratio of tool-retrieval turns to tool-call turns. Metric definitions follow Liu et al. (2026).

	Task completion		**Exploration behavior**		
Agent	Accuracy ↑	EGT Prec. ↑	Avg. turns	Mean EDT	S/C ratio
Published					
Qwen3-8B	0.00	35.31	25.65	7.64	0.20
Qwen3-14B	0.92	47.77	35.74	12.01	0.09
Qwen3-32B	2.75	62.36	12.03	18.54	1.59
Llama-3.1-8B-Instruct	0.00	41.33	21.62	9.89	1.49
Llama-3.3-70B-Instruct	18.96	59.67	19.13	19.20	2.22
GPT-5.4-Mini	3.07	71.25	10.81	9.22	1.97
GPT-5.4	51.90	72.92	22.92	20.65	2.70
Gemini-3.5-Flash	52.19	85.29	57.87	25.16	10.44
DeepSeek-V4-Flash	63.08	65.57	31.41	25.34	2.80
Gemini-3.1-Pro	77.06	91.47	19.55	27.41	1.59
Ours, SDP runtime					
SDP Gemma4-31B	38.53	76.12	32.04	20.44	10.28
WSG Gemma4-31B	53.88	82.72	30.70	22.10	7.36
SDP Qwen3.6-27B	16.05	66.84	32.85	19.52	15.16
WSG Qwen3.6-27B	48.01	76.39	36.21	23.71	4.90

Table 16: Full results on REAL, task accuracy in percent.

Agent	**Accuracy**
Published	
Qwen-2.5-VL-32B	3.0
OpenAI CUA	7.0
Gemma-3-27B	10.0
Llama-3.3-70B	11.0
Llama-4-Maverick	12.0
Grok-3	12.0
GPT-4o	14.0
o1	16.0
DeepSeek-V3	20.0
o3-mini	25.0
Gemini-2.5-Pro	35.0
o3-pro	38.0
Claude-3.7-Sonnet	41.0
Ours, SDP runtime	
SDP Gemma4-31B	27.7
WSG Gemma4-31B	37.3
SDP Qwen3.6-27B	22.6
WSG Qwen3.6-27B	40.2

C.2.6 DEEPPLANNING, SHOPPING

WSG raises the match score by 10.5 points on Gemma4-31B and by 22.1 on Qwen3.6-27B, and case accuracy rises with it, from 11.7 to 16.7 and from 5.0 to 13.3%. On Gemma4-31B the trained backbone passes GPT-5.2 and DeepSeek-V3.2 in their non-reasoning settings and sits below Qwen-Plus, and every reasoning model in the table remains ahead. The gap to the top is largest here of the seven benchmarks, and it is a gap to systems that think before every action, which the runtime does not do.

Table 18 shows the match score rising at every difficulty level on both backbones, while full case matches remain at zero on L3 for every run. The two backbones pay for the gain differently. On Gemma4-31B replans per episode fall from 6.74 to 4.26 and wall-clock time halves, so the prompted

Table 17: Full results on the shopping split of DeepPlanning.

Agent	Type	Match score	Case acc.
Published			
Grok-4.1-Fast (non-reasoning)	Non-reasoning	50.1	5.9
DeepSeek-V3.2 (w/o thinking)	Non-reasoning	58.3	10.6
GPT-5.2-none	Non-reasoning	58.6	8.6
GLM-4.7 (w/o thinking)	Non-reasoning	61.2	14.2
Qwen-Plus (w/o thinking)	Non-reasoning	63.9	15.0
Seed-1.8-minimal	Non-reasoning	68.1	22.5
Qwen3-Max (w/o thinking)	Non-reasoning	70.2	24.7
Claude-4.5-Sonnet (w/o thinking)	Non-reasoning	75.8	33.3
Claude-4.5-Opus (w/o thinking)	Non-reasoning	82.2	45.8
Kimi-K2-Thinking	Reasoning	65.8	24.2
o4-mini	Reasoning	69.1	21.7
Gemini-2.5-Pro	Reasoning	69.1	30.8
GLM-4.7 (w/ thinking)	Reasoning	72.5	27.5
Qwen-Plus (w/ thinking)	Reasoning	73.3	34.1
Grok-4.1-Fast (reasoning)	Reasoning	74.0	31.7
o3	Reasoning	76.9	38.5
Seed-1.8-high	Reasoning	77.5	40.8
Gemini-3-Pro-Preview	Reasoning	78.0	45.8
DeepSeek-V3.2 (w/ thinking)	Reasoning	78.8	42.5
Claude-4.5-Opus (w/ thinking)	Reasoning	80.0	45.0
Claude-4.5-Sonnet (w/ thinking)	Reasoning	80.0	43.3
GPT-5-high	Reasoning	80.4	44.2
Gemini-3-Flash-Preview	Reasoning	80.6	51.7
Qwen3-Max (w/ thinking)	Reasoning	82.6	43.5
GPT-5.2-high	Reasoning	84.8	54.2
Ours, SDP runtime			
SDP Gemma4-31B	Non-reasoning	48.5	11.7
WSG Gemma4-31B	Non-reasoning	59.0	16.7
SDP Qwen3.6-27B	Non-reasoning	29.1	5.0
WSG Qwen3.6-27B	Non-reasoning	51.2	13.3

Table 18: Per-level profile of our four runs on the shopping split of DeepPlanning, over 50 L1, 50 L2, and 20 L3 cases. Match is the match score and Case the case accuracy, both in percent, Replans and Sec are means per episode, and Exh. is the share of episodes in percent that spent the full step budget.

	SDP Gemma4-31B					WSG Gemma4-31B					SDP Qwen3.6-27B					WSG Qwen3.6-27B				
Level	Match	Case	Replans	Sec	Exh.	Match	Case	Replans	Sec	Exh.	Match	Case	Replans	Sec	Exh.	Match	Case	Replans	Sec	Exh.
L1	58.6	20.0	6.42	2033	38	72.2	32.0	2.98	818	40	35.4	12.0	2.66	1143	12	59.4	24.0	2.20	1264	34
L2	46.4	8.0	7.32	2278	52	51.1	8.0	5.70	1287	62	21.9	0.0	3.26	1289	24	47.5	8.0	3.18	1767	56
L3	28.4	0.0	6.10	1917	15	45.6	0.0	3.85	950	40	31.3	0.0	2.70	1256	10	39.9	0.0	2.90	1488	45
Overall	48.5	11.7	6.74	2116	40	59.0	16.7	4.26	1036	49	29.1	5.0	2.92	1223	17	51.2	13.3	2.73	1511	45

proposer was spending its repairs on rewording and the trained one is not. On Qwen3.6-27B replans barely change and time and budget exhaustion rise, from 17 to 45% of episodes, because the prompted proposer abandoned many cases at its first call and the trained one carries them through the full budget instead.

C.2.7 AGENTBOARD (PDDL)

WSG on Qwen3.6-27B matches the success rate of GPT-4 and GPT-4o at 61.7%, the highest published, and passes GPT-4o on progress rate. Both backbones already stood above every published open model under prompting, so the gain here is the smallest of the seven benchmarks, 5.7 and 3.4 points of success, and it comes with progress rate rising by 10.9 on Gemma4-31B and 6.1 on Qwen3.6-27B.

Table 19: Full results on the PDDL split of AgentBoard, in percent.

Agent	Progress rate	Success rate
Published		
Llama2-13B	4.1	0.0
Mistral-7B	4.7	0.0
Vicuna-13B-16K	7.2	1.7
Llama2-70B	8.1	1.7
CodeLlama-13B	9.3	1.7
Lemur-70B	9.7	3.3
AgentLM-70B	13.0	3.3
CodeLlama-34B	18.5	3.3
Llama3-8B	20.1	3.3
DeepSeek-67B	22.0	6.7
GPT-3.5-Turbo-16K	22.6	3.3
Gemini-1.5-Flash	23.5	6.7
GPT-3.5-Turbo	25.0	5.0
Text-Davinci-003	31.7	11.7
Claude-3-Haiku	31.9	13.3
Llama3-70B	32.2	20.0
xLAM-70B	38.4	16.7
Claude-2	61.4	40.0
GPT-4o	69.8	61.7
GPT-4	81.2	61.7
Ours, SDP runtime		
SDP Gemma4-31B	57.0	52.6
WSG Gemma4-31B	67.9	58.3
SDP Qwen3.6-27B	65.3	58.3
WSG Qwen3.6-27B	71.3	61.7

Table 20: Per-domain profile of our four runs on the PDDL split of AgentBoard, over 20 gripper, 10 blocksworld, 20 barman, and 10 tyreworld tasks. Prog. is the progress rate and Succ. the success rate, both in percent, Replans and Sec are means per episode, and Exh. is the share of episodes in percent that spent the full step budget.

	SDP Gemma4-31B					WSG Gemma4-31B					SDP Qwen3.6-27B					WSG Qwen3.6-27B				
Domain	Prog.	Succ.	Replans	Sec	Exh.	Prog.	Succ.	Replans	Sec	Exh.	Prog.	Succ.	Replans	Sec	Exh.	Prog.	Succ.	Replans	Sec	Exh.
Gripper	88.2	88.2	0.20	286	0	100.0	100.0	0.20	196	0	95.0	95.0	0.00	349	0	99.6	95.0	0.00	232	0
Blocksworld	100.0	100.0	0.00	116	0	100.0	100.0	0.00	111	0	90.0	70.0	0.50	158	0	91.7	80.0	0.50	146	0
Barman	17.5	10.0	8.80	657	40	28.3	15.0	7.00	535	65	15.9	10.0	7.70	956	85	35.0	25.0	4.30	523	70
Tyreworld	39.9	30.0	6.30	657	50	51.0	20.0	7.50	497	50	80.0	70.0	2.90	540	20	67.2	50.0	3.40	353	30
Overall	57.0	52.6	4.26	452	23	67.9	58.3	3.60	345	30	65.3	58.3	3.13	551	32	71.3	61.7	2.10	335	28

Table 20 shows that gripper and blocksworld are solved under prompting on both backbones and stay solved, so the whole gain sits in barman and tyreworld. Barman moves on both backbones, progress from 17.5 to 28.3 on Gemma4-31B and from 15.9 to 35.0 on Qwen3.6-27B, with replans falling from 7.7 to 4.3 on the latter. Tyreworld splits the two, progress rising on Gemma4-31B from 39.9 to 51.0 while success falls on both backbones, from 30.0 to 20.0 and from 70.0 to 50.0, so on this domain the trained proposer gets further into each task and completes fewer of them.