STATE-CENTRIC DECISION PROCESS

Anonymous authors
Paper under double-blind review

ABSTRACT

Language environments such as web browsers, code terminals, and interactive simulations emit raw text rather than states, and provide none of the runtime structure that MDP analysis requires. No explicit state space, no observation-to-state mapping, no certified transitions, and no termination criterion. We introduce the State-Centric Decision Process (SDP), a runtime framework that constructs these missing inputs by having the agent build them, predicate by predicate, as it acts. At each step the agent commits to a natural-language predicate describing how the world should look, takes an action to make it true, and checks the observation against it. Predicates that pass become certified states, and the resulting trajectory carries the four objects language environments do not provide, namely a task-induced state space, an observation-to-state mapping, certified transitions, and a termination criterion. We evaluate SDP on five benchmarks spanning planning, scientific exploration, web reasoning, and multi-hop question answering. SDP achieves the best training-free results on all five, with the advantage widening as the horizon grows. The certified trajectories additionally support analyses unavailable to reactive agents, including per-predicate credit assignment, failure localization, partial-progress measurement, and modular operator replacement.

1 INTRODUCTION

Language agents operate in environments that were never designed for autonomous decision-making, from web browsers and code terminals to interactive simulations and multi-step tool-use pipelines (Yoran et al., 2024; Zhou et al., 2023b; Jimenez et al., 2023; Wang et al., 2022; Xie et al., 2024). Large language models have made this possible, absorbing enough world knowledge from vast training corpora to select reasonable actions from raw observations without environment-specific engineering (Brown et al., 2020; Wei et al., 2022; Achiam et al., 2023; Huang et al., 2022; Xi et al., 2025). Yet success at action selection does not give the trajectory any formal structure: no explicit states, no verified transitions, nothing on which downstream methods can assign credit. Language environments do not expose this structure through their interface. The same situation admits unboundedly many valid natural-language descriptions, and which counts as a useful state is determined by the goal, a choice the environment has no access to.

Existing language agents address parts of this gap but none closes it. Reactive agents (Yao et al., 2022; Schick et al., 2023) operate directly on raw observations. Reflective agents accumulate verbal lessons or causal memories across episodes (Shinn et al., 2023; Majumder et al., 2023; Zhao et al., 2024), action planners deliberate over candidate action sequences (Wang et al., 2023b; Yao et al., 2023; Zhou et al., 2023a), and world-model approaches build internal descriptions of the environment (Wang et al., 2023c; Hao et al., 2023; Liu et al., 2023; Sun et al., 2023a). The latter three do maintain intermediate structure, and the gap is not its absence but its standing: no step asks the environment to confirm it before the agent moves on. The representation is therefore asserted rather than established, and the trajectory records what the agent believed rather than what held. On short tasks this is tolerable; on long-horizon problems, where errors compound and intermediate progress must be tracked, agents have no formal signal that progress is being made.

We propose to close this gap with an execution protocol that makes each intermediate claim answerable to the environment before the agent proceeds. We call the resulting framework the *State-Centric Decision Process* (SDP). Instead of choosing actions and treating state as a byproduct, the agent commits to a natural-language predicate describing how the world should look after the next action, takes an action intended to make it true, and checks the observation against it. Predicates

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.

that pass become certified states. Intermediate representations are not themselves new, and several agent frameworks already maintain them. What the protocol adds is the requirement that each one be discharged against the environment before it counts, and the trajectory that results carries the four objects MDP analysis requires but language environments do not expose: states, transitions, actions, and credit. SDP is therefore not a competitor to MDP-based formulations of agency, but the interface layer they presuppose.

We formalize a specification problem unstated in the literature, namely that MDP-based analysis requires four objects no language environment exposes and that the gap is one of specification rather than sample complexity. We introduce SDP, a runtime protocol that decomposes agency into the operators PROPOSE, REALIZE, VALIDATE, and REPLAN over natural-language predicates, and admits a step into the certified state only once the environment has confirmed it. SDP achieves the best training-free results on all five benchmarks, with the gap widening as the horizon grows, and under matched backbone and budget the same predicates declared without enforcement score at or below a reactive baseline, which isolates the protocol rather than the representation as the source of the gain. The certified trajectories additionally support analyses unavailable to prior agents, including failure localization, partial-progress tracking, cascade recording, and validator auditing.

2 PRELIMINARIES: THE MDP GAP IN LANGUAGE ENVIRONMENTS

MDP analysis requires a state space, an observation-to-state mapping, a transition kernel, and a termination criterion (Puterman, 2014; Sutton et al., 1998). Language environments expose none of these, emitting unstructured text such as web pages, terminal outputs, and API responses rather than states Xi et al. (2025); Wang et al. (2024); Sumers et al. (2023). Agents may maintain their own representations, and many do, but an environment that does not commit to a state space also does not confirm one, so what an agent maintains internally is asserted rather than established.

Useful state abstractions are goal-dependent. Let $\mathcal{O}$ denote the space of raw environment outputs and $\mathcal{H} = \bigcup_{t \geq 0} (\mathcal{O} \times A)^t \times \mathcal{O}$ the space of interaction histories. An MDP state is the image of a history under an abstraction $\phi : \mathcal{H} \to S$ that preserves the Markov property (Li et al., 2006; Givan et al., 2003). A single ϕ suffices only if the distinctions it draws are adequate for every goal the agent might pursue, and in language environments they are not, since two histories a summarization policy can safely identify must be separated by a checkout policy, and vice versa. Any ϕ fine enough to be Markov across all goals collapses toward the identity on $\mathcal{H}$, while any coarser ϕ is goal-specific. The useful abstraction is therefore not a single ϕ but a family $\{\phi_g\}_{g \in \mathcal{G}}$ indexed by goals (Abel et al., 2018; Andrychowicz et al., 2017), which an environment cannot fix in advance because it does not know the goal, and which the MDP formalism provides no mechanism to select at runtime.

Without a fixed state space, the rest follows. A goal-dependent state space leaves the transition kernel $T : S \times A \to \Delta(S)$ undefined, since its domain and codomain are not fixed sets, and every construct built on T, including value functions, Bellman backups, and policy gradients (Sutton et al., 1998; Williams, 1992), inherits the gap, as does termination, since nothing in the raw output stream signals completion. The POMDP relaxation fixes a maximally fine ϕ (Kaelbling et al., 1998; Young et al., 2013; Murphy et al., 2000), but its filtering equations presuppose the S and T in question. Language agents instead let a network operate on raw history (Yao et al., 2022; Wang et al., 2023a), and what they accumulate, whether skill libraries, causal memories, or state descriptions, is intermediate structure that no step asks the environment to confirm, so transitions over it cannot be distinguished from temporal adjacency.

The four missing inputs. The preceding argument exposes four inputs that MDP analysis requires but that no language environment supplies:

1. **State space.** A set S over which policies, values, and transitions are defined. The environment fixes no S, since the useful abstraction is goal-indexed and the goal arrives with the task.
2. **Observation-to-state mapping.** A function that turns each new observation into a state update. No single ϕ serves every goal, so the mapping must be established per task.
3. **Certified transitions.** Tuples (s, a, s') whose validity has been checked, not merely assumed from temporal adjacency. Checking requires a step at which the environment is consulted about s', which no interface provides on its own.

4. **Termination criterion.** A predicate over the state space that signals task completion. Language environments emit no such signal; the goal is supplied with the task, not by the environment.

3 STATE-CENTRIC DECISION PROCESS

We introduce the *State-Centric Decision Process* (SDP), a runtime protocol that supplies the four missing inputs by requiring each one to be discharged against the environment, predicate by predicate, as the agent acts. The constructed states are not observations, not summaries of observations, and not latent vectors, but natural-language predicates the agent commits to before acting, each describing how the world should look at a future step. A predicate becomes certified when the validator judges that the observation satisfies it, a recorded and enforced verdict rather than a proof, and one the next step can refute. The resulting trajectory carries what language environments do not expose, namely a state space populated by the task, a mapping from observations to state updates, transitions whose validity has been checked, and a termination test grounded in the goal (Figure 1).

3.1 THE STATE-CENTRIC DECISION PROCESS

The framework rests on four operators over a state space Σ, formalized below.

Definition 1 (State-Centric Decision Process)**.** An SDP is a tuple

$$(\Sigma, A, \mathcal{O}, \mathcal{T}, g, \textsc{Propose}, \textsc{Realize}, \textsc{Validate}, \textsc{Replan}),$$

where Σ is a space of natural-language predicates over $\mathcal{O}$, A is the action space, $\mathcal{O}$ is the space of raw environment outputs, $\mathcal{T}$ is the space of certified trajectories accumulated during execution, and $g \in \Sigma$ is a goal predicate given with the task. The four functions are:

$$\textsc{Propose} : \Sigma \times \Sigma \to \Sigma, \qquad (s_t, g) \mapsto \hat{s}_{t+1}, \tag{1}$$

$$\textsc{Realize} : \Sigma \times \Sigma \to A, \qquad (s_t, \hat{s}_{t+1}) \mapsto a_t, \tag{2}$$

$$\textsc{Validate} : \Sigma^* \times \mathcal{O} \to \mathbb{N}, \qquad (\hat{s}_{t+1}, \ldots, \hat{s}_n;\, o) \mapsto k, \tag{3}$$

$$\textsc{Replan} : \Sigma \times \Sigma \times \mathcal{T} \to \Sigma^*, \qquad (s_t, g, \tau_t) \mapsto (\hat{s}_{t+1}, \ldots, \hat{s}_n). \tag{4}$$

PROPOSE sets the next target from the current state and the goal, REALIZE selects an action to make it hold, and VALIDATE takes the resulting observation and returns the number $k \geq 0$ of consecutive targets from $\hat{s}_{t+1}$ that it satisfies, with $k = 0$ meaning the target is unmet. REPLAN produces a new sequence from the current state to g when the plan is unrecoverable, and n is the plan length.

By construction, only VALIDATE consumes raw observations; PROPOSE and REALIZE operate entirely on Σ, so targets reflect the agent's intent rather than reactions to whatever the environment presents. The three operators in the normal loop each read only local inputs, never the preceding history. REPLAN is the sole exception, receiving τ_t as the recovery mechanism invoked when the normal loop cannot advance. This locality yields the Markov property (Proposition 1).

Optimizing over states, not actions. Definition 1 reorganizes the decision problem. A reactive agent solves $a_t^* = \arg\max_{a \in A} P(\text{success} \mid h_t, a)$ at each step, with A as its decision space. SDP takes the *state plan* as decision variable, decomposing the problem into two coupled stages:

$$(\hat{s}_1, \ldots, \hat{s}_n)^* = \arg \max_{(\hat{s}_1, \ldots, \hat{s}_n) \in \Sigma^n} P\big(\text{plan reaches } g \,\big|\, s_0\big), \tag{5}$$

$$a_t^* = \arg\max_{a \in A} P\big(\textsc{Validate}(\hat{s}_{t+1}, \textsc{Env}(a)) \geq 1 \,\big|\, s_t, \hat{s}_{t+1}\big). \tag{6}$$

The outer objective searches over Σ^n for a predicate chain from s_0 to g, and the inner searches over A for an action whose environment response satisfies the next predicate. PROPOSE commits to the outer choice before execution begins, REALIZE solves the inner problem at each step, and VALIDATE supplies the per-step signal that closes the loop. The loop approximates both objectives rather than solving them in closed form, since PROPOSE and REALIZE each emit one candidate per call and VALIDATE scores that candidate against the environment's actual response, which is precisely the event Eq. 6 maximizes. A rejected candidate is excluded from the next proposal, so the search over A proceeds by elimination against environment feedback rather than by sampling alone, and REPLAN runs the same cycle one level up when the environment refutes a committed chain. Making this approximation explicit, by training a model for either stage instead of prompting one, is the subject of the research program SDP is intended to enable (Appendix D).

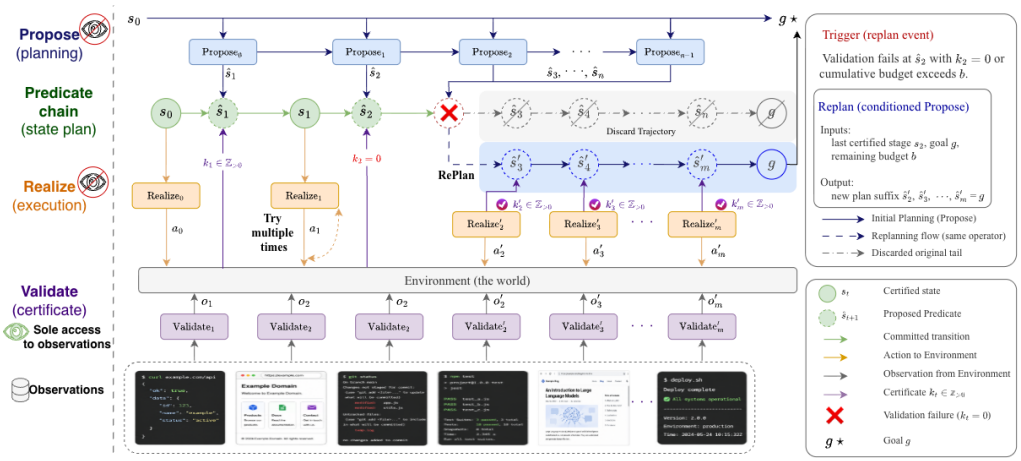


Figure 1: **The SDP execution structure.** PROPOSE builds the predicate chain from s_0 to g. REALIZE acts toward the next predicate. VALIDATE checks the observation and is the sole interface for $\mathcal{O}$. When failures at a target exceed b, REPLAN discards and builds a new state.

3.2 THE EXECUTION LOOP

The SDP execution loop separates planning from stepping. The agent first applies PROPOSE from s_0 repeatedly until it reaches g, producing a state plan $(\hat{s}_1, \ldots, \hat{s}_n = g)$, and Algorithm 1 gives the complete loop. At each step the agent selects an action via REALIZE and sends the response to VALIDATE (L. 4–6). If validation certifies one or more predicates, the cursor advances and the certified states enter the trajectory (L. 7–9). Otherwise the agent retries the same target, and after b consecutive failures REPLAN replaces the remaining plan suffix (L. 10–14).

Algorithm 1 The SDP execution loop.

Require: initial state s_0, goal g, attempt budget b
1: $(\hat{s}_1, \ldots, \hat{s}_n) \leftarrow$ BUILDPLAN(s_0, g) ▷ Propose until $\hat{s}_n = g$
2: $\tau \leftarrow (s_0),\ t \leftarrow 0,\ r \leftarrow 0$
3: **while** $t < n$ **do**
4: $\quad a_t \leftarrow$ REALIZE$(s_t, \hat{s}_{t+1})$
5: $\quad o_{t+1} \leftarrow$ ENV(a_t)
6: $\quad k \leftarrow$ VALIDATE$((\hat{s}_{t+1}, \ldots, \hat{s}_n), o_{t+1})$
7: $\quad$ **if** $k \geq 1$ **then**
8: $\quad\quad s_{t+1:t+k} \leftarrow \hat{s}_{t+1:t+k}$ ▷ certify k predicates
9: $\quad\quad \tau \leftarrow \tau \cup \{(a_t, s_{t+k})\}$
10: $\quad\quad t \leftarrow t + k,\ r \leftarrow 0$
11: $\quad$ **else**
12: $\quad\quad r \leftarrow r + 1$
13: $\quad\quad$ **if** $r > b$ **then**
14: $\quad\quad\quad (\hat{s}_{t+1}, \ldots, \hat{s}_n) \leftarrow$ REPLAN(s_t, g, τ)
15: $\quad\quad\quad r \leftarrow 0$
16: $\quad\quad$ **end if**
17: $\quad$ **end if**
18: **end while**
19: **return** τ

Cascade. One action can certify several targets. VALIDATE returns an integer rather than a binary verdict because a single action sometimes satisfies multiple consecutive predicates at once, whether from LLM knowledge spanning several sub-goals, predicate overlap, or overly fine-grained decomposition. For example, if the next two targets are "the athlete's average pace is computed" and "the total distance is derived from that pace," a single Python snippet can satisfy both, and treating the first alone as satisfied would force a redundant second action. VALIDATE therefore reports how many consecutive predicates from the head of the plan are jointly satisfied and advances the cursor by that many positions, so certified transitions take the form (s_t, a_t, s_{t+k}), generalizing single-step MDP transitions.

Failure in execution and failure in planning are corrected separately. When an action fails to realize a target, REALIZE retries with a different action against the same predicate while the rest of the plan is untouched. Retries are informed by a record of the actions already rejected at this target, which is local to the target and discarded once the predicate is certified, so the search over A narrows without altering the certified state. The plan itself is revised only when repeated attempts exhaust the budget b, at which point REPLAN discards the plan tail and proposes a new sequence from s_t to g. This separation matters most in long-horizon tasks, since a broken action costs one attempt while a broken plan is repaired locally without restarting from s_0. Reactive agents must reconsider both whenever either fails, whereas SDP corrects each on its own timescale.

3.3 WHAT AN SDP RUN PRODUCES

An SDP run terminates with more than an answer to the task. It returns a structured artifact that records every certified state, the action that produced it, and the cascade depth recorded by each validation. The artifact has the form

$$\big(s_0,\ a_0,\ s_{t_1},\ a_{t_1},\ \ldots,\ a_{T-1},\ s_T\big), \quad s_T \models g,$$

where the indices $t_1 < t_2 < \cdots < T$ mark the steps at which validation discharged at least one target. Each gap $t_{i+1} - t_i$ is the cascade depth k at that step. This artifact closes the gap identified in Section 2 by supplying, item by item, the four objects that MDP analysis requires but that no language environment exposes. Table 1 states the correspondence explicitly.

Table 1: MDP inputs not exposed by language environments and the artifacts SDP produces. The last row adds a credit signal beyond the four.

MDP input	**Supplied by SDP**
S	$\Sigma_g \subseteq \Sigma$, populated by the plan and cascades
$\phi : \mathcal{O} \to S$	VALIDATE, mapping each observation to $k \in \mathbb{N}$
Transitions	Certified (s_t, a_t, s_{t+k}) from each successful validation
Termination	The relation $s_T \models g$
Credit	The integer k returned by VALIDATE

Table 1 is not a claim that we have applied any downstream method to the artifact. The claim is that each such method is now a *well-posed problem* on it, in a way it is not on raw histories of language environments, and we claim no more than that here. The artifact also carries a structural guarantee.

Proposition 1 (Markov property of SDP trajectories)**.** *Let $\tau = (s_0, a_0, s_{t_1}, a_{t_1}, \ldots, s_{t_i})$ be a prefix of a trajectory generated by Algorithm 1, let prefix $= (s_0, a_0, \ldots, a_{t_i-1})$ denote the history before s_{t_i}, and let $P_i = (\hat{s}_{t_i+1}, \ldots, \hat{s}_n)$ be the plan tail at step t_i. Assume the environment's response $P(o_{t_i+1} \mid a_{t_i}, \textit{prefix})$ depends on the prefix only through (s_{t_i}, P_i). Then the next certified state satisfies*

$$P\big(s_{t_{i+1}} \,\big|\, s_{t_i}, P_i, \textit{prefix}\big) = P\big(s_{t_{i+1}} \,\big|\, s_{t_i}, P_i\big). \tag{7}$$

Proof sketch. The next certified state $s_{t_{i+1}}$ depends only on a_{t_i} and $k = \text{VALIDATE}(P_i, o_{t_i+1})$. By Definition 1, REALIZE reads only the current state and the next target, and VALIDATE reads only the plan tail and the new observation, so neither consumes the prefix, and o_{t_i+1} depends on it only through (s_{t_i}, P_i) by assumption. REPLAN receives τ only to produce a new tail P_i', for which the same independence holds, so the prefix enters $s_{t_{i+1}}$ only through (s_{t_i}, P_i). □

Proposition 1 states a design target rather than a discovery about language environments, and the two sides of its condition have different status. The agent side is enforced mechanically, since PROPOSE and REALIZE read only the current state and the next target, a failed attempt leaves the certified state unchanged, and REPLAN regenerates only the tail. The environment side remains an assumption, which is why the proposition states it as one, and where it fails the violation surfaces as a failed validation handled by the recovery path rather than corrupting the trajectory silently.

The enforcement has three practical consequences. Because each call conditions on the certified state rather than the accumulated history, per-call context stays near-constant as the horizon grows (Appendix C.1). Because a failed step leaves the state untouched, a failure is localized to the predicate where certification stopped. And a trajectory can be cut into tuples $(s_{t_i}, a_{t_i}, k, s_{t_{i+1}})$ learned one at a time, rather than each sample carrying its entire history.

4 EXPERIMENTS

We evaluate SDP along two axes: whether organizing an agent around a state plan improves task performance, and whether the certified trajectories support analyses unavailable to reactive agents. The five benchmarks span distinct combinations of environment structure and goal type, including constraint satisfaction over structured option sets, open-ended web reasoning with free-form retrieval, interactive scientific exploration in a text-based simulator, and multi-hop question answering at varying chain depths. For each benchmark we compare SDP against the published baselines evaluated on it, retaining each baseline with the LLM reported in its original paper. Benchmark selection and baseline fairness criteria, including backbone matching, are in Appendices A and B.

4.1 TASK PERFORMANCE ACROSS LANGUAGE ENVIRONMENTS

TravelPlanner. Table 2 reports results on TravelPlanner (Xie et al., 2024). SDP's largest gain is on hard constraints, achieving 97.4% Micro and 93.8% Macro and exceeding ATLAS by 14.8 and 19.4 points despite ATLAS using a larger backbone (Gemini-2.5-Pro). This traces to the predicate structure, since each constraint becomes a separate predicate that must be certified before the plan advances, so violations surface at the step where they arise rather than after the itinerary is assembled. Two failure modes of plan-as-text baselines reinforce the gap. In our reproduction runs, 12–18% of outputs from ReAct, Plan-and-Solve, and Reflexion were malformed, and 20–30% of feasible tasks suffered budget overflow from single-pass generation. SDP eliminates both.

Table 2: Results on TravelPlanner. Metrics are Delivery Rate, Commonsense constraint satisfaction (Micro/Macro), Hard Constraint satisfaction (Micro/Macro), and Final Pass Rate (%).

			Commonsense		Hard Constraint		
Algorithm	LLM	Delivery Rate	Micro	Macro	Micro	Macro	Final Pass Rate
ReAct (Yao et al., 2022)	Gemini-3.1-flash-lite	95.0	78.6	16.1	83.8	61.7	7.2
Reflexion (Shinn et al., 2023)	Gemini-3.1-flash-lite	100.0	80.5	26.7	77.9	55.6	12.8
Plan-and-Solve (Wang et al., 2023b)	Gemini-3.1-flash-lite	99.4	84.7	26.1	78.9	56.7	13.3
MIRROR (Guo et al., 2025)	GPT-4o	100.0	73.2	13.9	27.6	13.3	2.2
GoalAct (Chen et al., 2025)	Gemini-3.1-flash-lite	99.4	83.1	25.0	78.7	56.1	15.0
TDP (Li et al., 2026)	Gemini-3.1-flash-lite	99.4	81.6	27.2	79.2	61.1	13.3
EvoAgent (Yuan et al., 2025)	Gemini-2.5-Pro	100.0	78.1	23.9	57.9	40.6	12.2
PMC (Zhang et al., 2025)	Gemini-2.5-Pro	100.0	78.7	30.6	43.3	37.2	23.3
ATLAS (Choi et al., 2025)	Gemini-2.5-Pro	100.0	88.5	48.3	82.6	74.4	44.4
SDP (Ours)	Gemini-3.1-flash-lite	**100.0**	**95.2**	61.7	**97.4**	93.8	61.7
SDP (Ours)	GPT-4o	**100.0**	**95.6**	**65.6**	**97.4**	**93.9**	**65.6**

AssistantBench. Table 3 reports results on AssistantBench (Yoran et al., 2024). SDP achieves the highest overall accuracy, precision, and EM using only a search API and URL-level scraping, without browser rendering, JavaScript, or form interaction, and roughly 40% of certified steps required at least one rejection-and-retry cycle. SDP does not lead on Hard-tier accuracy, where Magentic-One wins, because those tasks require multi-page browsing sessions a search-only interface cannot replicate. Where this limitation does not apply the advantage is decisive, with 92.8% on Easy-tier tasks exceeding the next best by over 10 points, confirming that structured validation rather than richer tool access is the binding factor.

Table 3: Results on AssistantBench. We report overall Accuracy, Precision, Exact Match (EM), and Accuracy stratified by difficulty (Easy/Medium/Hard).

					Accuracy by Difficulty		
Method	LLM	Accuracy	Precision	EM	Easy	Medium	Hard
Infogent (Reddy et al., 2025)	GPT-4o	14.5	20.4	5.5	63.9	19.3	8.4
RALM-1S→CB (Trivedi et al., 2023)	GPT-4T	19.5	21.0	6.1	81.3	35.0	7.3
CB-1S (Press et al., 2023)	GPT-4T	22.2	24.8	8.3	67.8	49.7	4.2
SeeAct→CB (Zheng et al., 2024)	GPT-4T	23.4	26.1	9.4	82.0	47.7	7.1
SPA-CB (Yoran et al., 2024)	GPT-4T	25.2	27.5	9.9	80.7	42.7	12.4
Magentic-One (Fourney et al., 2024)	GPT-4o	25.3	25.3	11.0	69.9	35.6	**16.9**
ACP-Domain Agents (Bhardwaj et al., 2025)	GPT-4o	28.3	30.0	11.0	67.8	48.5	15.5
SDP (Ours)	GPT-4o	**31.8**	**32.0**	**14.9**	**92.8**	**54.9**	16.0

ScienceWorld. Table 4 reports results on ScienceWorld (Wang et al., 2022). Among training-free methods on the standard 30-task GPT-4 protocol, SDP achieves the highest overall score (59.16), improving over the previous best by 11.3 points. The gap widens with task length. On Long tasks SDP reaches 50.41, leading Plan-and-Act by 15.6 points and Reflexion by over 20 points, consistent with the expectation that explicit state tracking matters more as the horizon grows. The predicate structure also yields plan compression, as the LLM's plan for a task like "boil water" uses 7 predicates that subsume the oracle's 36-step action sequence while preserving causal ordering, derived from the task description alone without in-context trajectory examples. Granularity proved more consequential than length, since overly abstract predicates confused the validator while overly fine-grained ones inflated LLM calls without improving accuracy, and predicates pitched at a single observable state change performed most reliably. The dominant failure mode here is not planning or action selection but inaccurate validation.

Table 4: Results on ScienceWorld across task lengths, reported as average score. All methods are training-free with GPT-4 on the standard 30-task protocol.

Method	Short	Medium	Long	Overall
SayCan (Ahn et al., 2022)	43.83	36.58	23.65	33.82
ReAct (Yao et al., 2022)	48.79	44.01	21.07	36.43
CoT (Wei et al., 2022)	49.54	47.87	23.09	39.23
Reflexion (Shinn et al., 2023)	71.47	35.43	30.17	45.34
EVOAGENT (Yuan et al., 2025)	48.67	36.17	11.38	30.42
Plan-and-Act (Erdogan et al., 2025)	60.52	46.43	34.77	47.86
SDP (Ours)	**73.72**	**53.50**	**50.41**	**59.16**

Table 5: Results on multi-hop QA in the fullwiki open-domain setting. Both benchmarks use BM25 retrieval over a 5M-paragraph Wikipedia corpus and 1,000 validation examples.

		HotpotQA		MuSiQue	
Method	LLM	EM	F1	EM	F1
RankZephyr (Pradeep et al., 2023)	GPT-4o	34.7	35.0	8.6	12.8
RankZephyr + CoT (Pradeep et al., 2023)	GPT-4o	34.0	34.4	9.4	13.3
RankGPT (Sun et al., 2023b)	GPT-4o	34.6	35.3	9.5	13.5
SETR-CoT & IRI (Lee et al., 2025)	GPT-4o	39.2	40.5	12.3	16.9
IRCoT (Trivedi et al., 2023)	GPT-3	49.3	60.7	34.2	43.8
PRISM (Nahid & Rafiei, 2025)	GPT-4o	54.2	67.0	31.2	41.8
SDP	GPT-4o	**58.3**	**67.2**	**41.4**	**51.9**

Multi-hop question answering. Table 5 reports results on HotpotQA (Yang et al., 2018) and MuSiQue (Trivedi et al., 2022). SDP's advantage is most pronounced on MuSiQue, where chains reach three or four hops. VALIDATE blocks hallucinated intermediate findings before they propagate, rejecting evidence from distractor passages so the agent re-attempts with the correct paragraph, whereas reactive baselines let plausible but incorrect evidence silently corrupt the chain. The converse is the primary challenge, since a retriever that surfaces no relevant evidence leaves the loop with no better paragraph to try, and roughly 76% of MuSiQue failures trace to this bottleneck. SDP achieves these results with BM25 alone, whereas the strongest baselines employ hybrid retrieval.

4.2 ANATOMY OF SDP TRAJECTORIES

The previous results evaluated SDP as a prompting strategy. This section examines the artifact it produces. The certified trajectory carries structured information that a reactive trajectory does not, namely whether single actions advanced multiple predicates, where the agent stalls, how much progress was made before failure, and how well the validator's verdicts agree with ground truth.

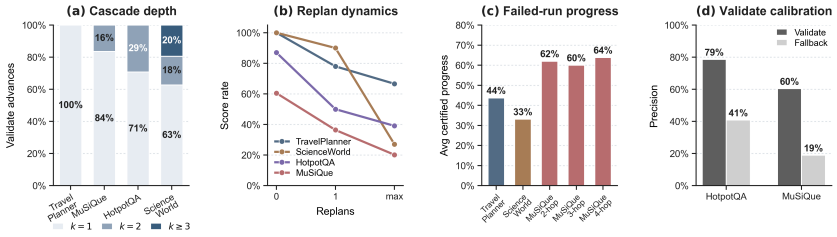


Figure 2: Anatomy of SDP trajectories. **(a)** Cascade depth. **(b)** Score rate vs. replan count. **(c)** Certified progress in failed runs. **(d)** VALIDATE calibration.

Cascade and replan rates are environment properties (Figure 2a,b). Whether a single action satisfies multiple predicates, and whether a failed plan is locally repairable, are environment-determined. Cascade ranges from 0% on TravelPlanner, where each slot demands separate selection, to 37% on ScienceWorld, where bundled sub-goals are common. The replan curve tracks environment recoverability, with ScienceWorld holding full success through one replan while TravelPlanner declines steadily, since replan cannot rescue infeasible option sets. This variation confirms SDP records these rates honestly rather than imposing a single pattern.

Where and how far runs fail (Figure 2c). Because every rejection attaches to a specific predicate, the trajectory records not only that a run failed but where and how far. TravelPlanner failures concentrate on hotel selection and return transport, and MuSiQue failures typically certify the first hop or two before stalling on a bridge entity. A reactive run is binary, while an SDP run is a fraction, with failed TravelPlanner runs certifying 44% of their plan and failed MuSiQue runs 60–64% of their hops. In MuSiQue longer chains fail farther in, surfacing more partial information.

Calibration is auditable (Figure 2d). Because VALIDATE certifies the goal predicate as a discrete decision, every run yields a ground-truth check. Goal certification agrees with a correct final answer

79% of the time on HotpotQA and 60% on MuSiQue, while runs that fall back to forced finalization reach only 41% and 19%, so the certification signal carries information beyond LLM parametric guessing. The gap is measured rather than hidden, since runs where VALIDATE accepts the full chain but the answer is wrong form an identifiable subset the trajectory isolates.

Certification errors are absorbed rather than propagated. Calibration measures how often VALIDATE errs, and the next question is what an error costs. We override rejected validations with probability ϵ, and at $\epsilon = 0.20$ SDP still exceeds uncontaminated ReAct on all four LLM-validated benchmarks. A false certification corrupts the record but not the environment, so the next action built on it fails and the loop recovers through its ordinary retry and replan path (Appendix C.2).

4.3 IS THE STRICT LOOP NECESSARY?

The predicate structure could in principle be obtained by prompting rather than by enforcement. We test this by granting ReAct each element of SDP in prompt form alone: *system-prompt tracking* maintains a running description of what is true, *predicate-plan-only* appends the full chain to the prompt while leaving execution reactive, *expected postcondition* states what the next action should make true, and *validator-only* verifies each action against no committed target. None of the four checks a declaration against the environment before the agent proceeds. Methods here and in the next subsection run on GPT-4o with identical tools, budgets, and task sets over three independent runs, so absolute values differ from Section 4.1 and Appendix C.1 states the protocol in full.

Table 6: Lighter predicate-guided variants of ReAct vs. SDP under the matched protocol, as mean $\pm$ 1 std over three runs. Metrics are Final Pass Rate, average score, accuracy, and F1 respectively.

Method	TravelPl.	SciWorld	AsstBench	HotpotQA	MuSiQue
ReAct	8.9 ± 1.1	36.4 ± 4.5	22.9 ± 0.4	45.0 ± 3.6	35.9 ± 2.9
+ System-prompt tracking	6.1 ± 0.3	26.2 ± 4.9	18.9 ± 1.2	44.9 ± 1.5	30.7 ± 1.2
+ Predicate-plan-only	9.4 ± 1.7	34.8 ± 9.5	23.2 ± 0.6	49.9 ± 5.3	40.5 ± 5.8
+ Expected postcondition	5.6 ± 1.4	29.3 ± 3.3	21.6 ± 0.7	42.0 ± 2.5	36.7 ± 5.3
+ Validator-only	17.2 ± 1.8	41.7 ± 6.0	23.4 ± 0.4	43.6 ± 1.8	36.6 ± 3.4
Full SDP	$\mathbf{64.9 \pm 2.5}$	$\mathbf{54.5 \pm 3.6}$	$\mathbf{31.9 \pm 0.3}$	$\mathbf{66.3 \pm 3.1}$	$\mathbf{52.2 \pm 1.7}$

Declaring a predicate without enforcing it does not help, and often hurts. System-prompt tracking and expected postconditions fall below plain ReAct on three benchmarks, since both have the model assert what is or should become true and then act on state it never verified. An unenforced plan is inert rather than harmful, with predicate-plan-only statistically indistinguishable from ReAct on four of five, since a chain the loop never consults changes nothing.

The loop is what converts the pieces into the gain. Validator-only recovers part of the gap, reaching 17.2 on TravelPlanner and 41.7 on ScienceWorld, so verification does real work even without a plan to verify against, yet it stays far below full SDP everywhere. The two are mutually enabling, since a plan gives validation a committed target and validation makes the plan binding rather than advisory. If these benchmarks rewarded predicate structure as such, the prompted variants would capture some of that reward, and they do not on any of the five.

4.4 ABLATION STUDY

The framework's three loop-level mechanisms are each removable. We replace VALIDATE with always-pass, REPLAN with termination on budget exhaust, and cascade with a cap of $k \leq 1$, and run each variant in full under the matched protocol.

Table 7: Effect of removing each mechanism under the protocol, as mean $\pm$ 1 std over 3 runs.

	TravelPl. (Score)	SciWorld (AS)	AsstBench (Acc)	HotpotQA (F1)	MuSiQue (F1)
−VALIDATE (always pass)	67.9 ± 2.9	11.1 ± 1.9	18.8 ± 0.6	13.6 ± 0.5	8.2 ± 1.5
−REPLAN (terminate on budget)	72.4 ± 1.3	23.6 ± 0.6	21.4 ± 0.5	44.8 ± 0.9	33.2 ± 1.9
−Cascade (force $k \leq 1$)	95.7 ± 0.9	30.5 ± 0.7	20.7 ± 0.7	60.2 ± 1.3	48.0 ± 1.3
Full SDP	$\mathbf{95.7 \pm 0.9}$	$\mathbf{54.5 \pm 3.6}$	$\mathbf{31.9 \pm 0.3}$	$\mathbf{66.3 \pm 3.1}$	$\mathbf{52.2 \pm 1.7}$

VALIDATE is the dominant contributor, except where the environment substitutes for it. Removing VALIDATE produces the largest drop on four of five benchmarks, confirming it is the only mechanism catching uncertified findings before they propagate. TravelPlanner is the exception, since its constraints are formally specifiable and checked deterministically rather than by the validator. Where success criteria are semantic, VALIDATE is the principal source of correctness.

REPLAN pays off in proportion to environment recoverability. REPLAN matters most where local re-attempts can recover progress and least where no alternative exists. ScienceWorld shows the strongest effect since most successful runs needed at least one replan to navigate an unexpected obstacle, while AssistantBench shows the weakest as nearly all tasks complete on the original plan. TravelPlanner presents a distinct pattern: the score drops not because replanning would have rescued the task, but because the certified prefix is shorter when the agent terminates on budget exhaustion.

Cascade is decisive only under tight attempt budgets. Cascade acts as a budget-multiplier, advancing the cursor by $k \geq 2$ and preventing budget exhaustion when actions bundle. ScienceWorld shows the largest effect without cascade (54.5 $\rightarrow$ 30.5), consistent with its 37% cascade rate. On benchmarks where cascade is rare or the budget is generous, disabling it has little impact.

5 RELATED WORK

Language agents without a certification step. The dominant paradigm maps observation history directly to action (Yao et al., 2022; Schick et al., 2023), and subsequent work adds reflections and memories (Shinn et al., 2023; Majumder et al., 2023), plans over action sequences (Wang et al., 2023b; Yao et al., 2023), or internal models of the environment (Wang et al., 2023c; Sun et al., 2023a). In each case the representation is consumed by the module that selects the next action, with no step where the environment confirms it, so nothing blocks progress when it is wrong. SDP keeps the insight that committing before acting beats reacting, and adds the step these systems leave out.

Hierarchical decomposition and subgoals. Decomposing a task into intermediate targets is long established, from options and subgoal discovery in hierarchical reinforcement learning (Sutton et al., 1999; Nachum et al., 2018) to goal-conditioned policies (Schaul et al., 2015; Andrychowicz et al., 2017) and LLM agents that decompose a goal into natural-language subgoals (Huang et al., 2022; Erdogan et al., 2025). SDP's predicates are subgoals in this sense, and we claim no novelty for the decomposition itself. What differs is the execution protocol. A classical subgoal is reached when a learned termination condition fires over a pre-specified state space, and an LLM subgoal is typically consumed as guidance with no test, whereas each SDP subgoal is a proposition the environment must satisfy at its own step, and the agent advances only on the ones it does.

Decision-theoretic formulations for language environments. The standard approach posits an MDP or POMDP and learns or plans within it, which has been productive in dialogue systems (Young et al., 2013; Williams & Young, 2007), game playing (Silver et al., 2017), and gridworld instruction following (Branavan et al., 2009), where the state space is fixed before learning begins. Open-ended language environments require committing to a state space before the task is known, and recent agents that adopt the POMDP formalism (Song et al., 2024; Yang et al., 2025) posit the state space but leave it latent, conditioning the policy on raw observation history so the declared formalism does no work. SDP takes the opposite order, assuming no MDP at the outset and building the state space at runtime, predicate by predicate, so the trajectory carries the objects such analysis requires, populated by the task rather than posited in advance.

6 CONCLUSION

Language environments expose none of the runtime structure MDP analysis requires. No state space, no observation-to-state mapping, no certified transitions, and no termination criterion. SDP closes this gap with an execution protocol that admits a step into the certified state only once the environment has confirmed it, so the agent builds all four predicate by predicate as it acts. Experiments on five benchmarks confirm the structure improves task performance, with the advantage widening as the horizon grows, and that the same predicates declared without enforcement do not. The resulting trajectory supports analyses unavailable to reactive agents, and makes the decision-theoretic tools that language agents have lacked well-posed problems rather than ambitions.

AI USE STATEMENT

Large language models were used to polish prose and check grammar in drafts written by the authors, and to surface candidate related work, every item of which was read in the original before being cited. All technical content was conceived and written by the authors, who take full responsibility for the final text. LLMs also serve as the backbone of the SDP operators and of every baseline we run, which is part of the method rather than the writing process and is described in Section 4 and Appendix A.

ETHICS STATEMENT

This work evaluates language agents on public benchmarks and involves no human subjects, no personal data, and no newly released datasets. The web-facing experiments issue read-only search and scraping requests through a commercial API and take no actions that modify external state. The framework is intended to make agent trajectories more auditable, though we note that a structure which improves reliability on benign tasks would do the same on harmful ones, and that a validator can certify a predicate the environment appears to satisfy without the underlying condition holding.

REPRODUCIBILITY STATEMENT

Every benchmark adapter is specified in Appendix A, including state representation, predicate construction, validation mode, and all hyperparameters. Appendix B states the evaluation protocol, data splits, and baseline selection criteria for each benchmark, and Appendix C.1 defines the matched-backbone protocol under which the controlled comparisons and ablations were run, with all such results reported as the mean and standard deviation of three independent runs. The proof of Proposition 1 states its assumption explicitly and is given in full in Section 3.3. Source code is included in the supplementary materials.

REFERENCES

David Abel, Dilip Arumugam, Lucas Lehnert, and Michael Littman. State abstractions for lifelong reinforcement learning. In *International conference on machine learning*, pp. 10–19. PMLR, 2018.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017.

Devansh Bhardwaj, Arjun Beniwal, Shreyas Chaudhari, Ashwin Kalyan, Tanmay Rajpurohit, Karthik R Narasimhan, Ameet Deshpande, and Vishvak Murahari. Agent context protocols enhance collective inference. *arXiv preprint arXiv:2505.14569*, 2025.

Satchuthananthavale RK Branavan, Harr Chen, Luke Zettlemoyer, and Regina Barzilay. Reinforcement learning for mapping instructions to actions. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pp. 82–90, 2009.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Junjie Chen, Haitao Li, Jingli Yang, Yiqun Liu, and Qingyao Ai. Enhancing llm-based agents via global planning and hierarchical execution. *arXiv preprint arXiv:2504.16563*, 2025.

Jihye Choi, Jinsung Yoon, Jiefeng Chen, Somesh Jha, and Tomas Pfister. Atlas: Constraints-aware multi-agent collaboration for real-world travel planning. *arXiv preprint arXiv:2509.25586*, 2025.

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

Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, Jacob Alber, et al. Magentic-one: A generalist multi-agent system for solving complex tasks. *arXiv preprint arXiv:2411.04468*, 2024.

Robert Givan, Thomas Dean, and Matthew Greig. Equivalence notions and model minimization in markov decision processes. *Artificial intelligence*, 147(1-2):163–223, 2003.

Zikang Guo, Benfeng Xu, Xiaorui Wang, and Zhendong Mao. Mirror: Multi-agent intra-and inter-reflection for optimized reasoning in tool learning. *arXiv preprint arXiv:2505.20670*, 2025.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 8154–8173, 2023.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning*, pp. 9118–9147. PMLR, 2022.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023.

Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

Dahyun Lee, Yongrae Jo, Haeju Park, and Moontae Lee. Shifting from ranking to set selection for retrieval augmented generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 17606–17619, 2025.

Lihong Li, Thomas J Walsh, and Michael L Littman. Towards a unified theory of state abstraction for mdps. *AI&M*, 1(2):3, 2006.

Yunfan Li, Bingbing Xu, Xueyun Tian, Xiucheng Xu, and Huawei Shen. Beyond entangled planning: Task-decoupled planning for long-horizon agents. *arXiv preprint arXiv:2601.07577*, 2026.

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.

Zhihan Liu, Hao Hu, Shenao Zhang, Hongyi Guo, Shuqi Ke, Boyi Liu, and Zhaoran Wang. Reason for future, act for now: A principled framework for autonomous llm agents with provable sample efficiency. *arXiv preprint arXiv:2309.17382*, 2023.

Bodhisattwa Prasad Majumder, Bhavana Dalvi Mishra, Peter Jansen, Oyvind Tafjord, Niket Tandon, Li Zhang, Chris Callison-Burch, and Peter Clark. Clin: A continually learning language agent for rapid task adaptation and generalization. *arXiv preprint arXiv:2310.10134*, 2023.

Kevin P Murphy et al. A survey of pomdp solution techniques. *environment*, 2(10):1–12, 2000.

Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

Md Mahadi Hasan Nahid and Davood Rafiei. Prism: Agentic retrieval with llms for multi-hop question answering. *arXiv preprint arXiv:2510.14278*, 2025.

Ronak Pradeep, Sahel Sharifymoghaddam, and Jimmy Lin. Rankzephyr: Effective and robust zero-shot listwise reranking is a breeze! *arXiv preprint arXiv:2312.02724*, 2023.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 5687–5711, 2023.

Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

Revanth Gangi Reddy, Sagnik Mukherjee, Jeonghwan Kim, Zhenhailong Wang, Dilek Hakkani-Tur, and Heng Ji. Infogent: An agent-based framework for web information aggregation. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 5745–5758, 2025.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International conference on machine learning*, pp. 1312–1320. PMLR, 2015.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

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.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. Trial and error: Exploration-based trajectory optimization of llm agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7584–7600, 2024.

Theodore Sumers, Shunyu Yao, Karthik R Narasimhan, and Thomas L Griffiths. Cognitive architectures for language agents. *Transactions on Machine Learning Research*, 2023.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. Adaplanner: Adaptive planning from feedback with language models. *Advances in neural information processing systems*, 36:58202–58245, 2023a.

Weiwei Sun, Lingyong Yan, Xinyu Ma, Shuaiqiang Wang, Pengjie Ren, Zhumin Chen, Dawei Yin, and Zhaochun Ren. Is chatgpt good at search? investigating large language models as re-ranking agents. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 14918–14937, 2023b.

Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

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.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers)*, pp. 10014–10037, 2023.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023a.

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, 2023b.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024.

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.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. *arXiv preprint arXiv:2302.01560*, 2023c.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Jason D Williams and Steve Young. Partially observable markov decision processes for spoken dialog systems. *Computer Speech & Language*, 21(2):393–422, 2007.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. The rise and potential of large language model based agents: A survey. *Science China Information Sciences*, 68(2):121101, 2025.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. Travelplanner: A benchmark for real-world planning with language agents. *arXiv preprint arXiv:2402.01622*, 2024.

Ke Yang, Yao Liu, Sapana Chaudhary, Rasool Fakoor, Pratik A Chaudhari, George Karypis, and Huzefa Rangwala. Agentoccam: A simple yet strong baseline for llm-based web agents. In *International Conference on Learning Representations*, volume 2025, pp. 97533–97565, 2025.

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.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

Ori Yoran, Samuel Joseph Amouyal, Chaitanya Malaviya, Ben Bogin, Ofir Press, and Jonathan Berant. Assistantbench: Can web agents solve realistic and time-consuming tasks? In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 8938–8968, 2024.

Steve Young, Milica Gašić, Blaise Thomson, and Jason D Williams. Pomdp-based statistical spoken dialog systems: A review. *Proceedings of the IEEE*, 101(5):1160–1179, 2013.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Dongsheng Li, and Deqing Yang. Evoagent: Towards automatic multi-agent generation via evolutionary algorithms. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 6192–6217, 2025.

Cong Zhang, Xin Deik Goh, Dexun Li, Hao Zhang, and Yong Liu. Planning with multi-constraints via collaborative language agents. In *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 10054–10082, 2025.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. Gpt-4v (ision) is a generalist web agent, if grounded. *arXiv preprint arXiv:2401.01614*, 2024.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*, 2023a.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023b.

A IMPLEMENTATION DETAILS

A.1 COMMON SETTINGS

All five benchmarks share the same SDP core loop, and only the adapter layer differs per benchmark.

Core loop. Given a task, the core calls the adapter's PROPOSE to obtain a predicate plan, then iterates through it, calling REALIZE to produce an action for the current predicate and VALIDATE to check whether the resulting observation satisfies it. If validation returns $k \geq 1$, the first k predicates are certified and removed from the plan. Otherwise REALIZE retries up to the attempt budget, after which REPLAN generates a new plan tail from the stuck predicate. The loop terminates when the goal predicate is certified or the global step cap is reached.

Attempt history. Rejected attempts at the current target are recorded with the attempted action and the rejection reason, and this record is included in the next REALIZE prompt so that the model does not repeat a failed choice. The record is local to the target and discarded once the predicate is certified. When the budget is exhausted, REPLAN receives the certified trajectory together with a summary of the failures at the stuck predicate. Failed attempts never modify the certified state.

Validation modes. SDP does not prescribe how VALIDATE is implemented. TravelPlanner uses deterministic constraint checks with no LLM involvement. HotpotQA and MuSiQue use an LLM verifier that checks whether a finding is supported by its cited paragraph, combined with simple deterministic rejection rules. AssistantBench and ScienceWorld use LLM-based verdicts. The choice follows domain structure, with deterministic checks where constraints are formally specifiable and LLM judgment where success criteria are semantic.

LLM backbone and inference. Each experimental condition uses a single backbone for all four operators. Temperature is 0 except where stated, outputs are requested in JSON, and all prompts are zero-shot with no few-shot examples.

A.2 TRAVELPLANNER

State and predicate plan. The certified state holds the task constants (origin, destination, dates, travelers, budget, local constraints), the trip topology, and the decisions made so far, including chosen transports, accommodations, meals, attractions, and the running cost. The predicate plan is determined by the number of days and cities, with predicates for outbound transport, accommodation per city, inter-city transport, meals and attractions per day, return transport, a final budget check, and rendering into the official output format. This yields 11 predicates for 3-day single-city trips, 17 for 5-day two-city trips, and 23 for 7-day three-city trips. Cost-incurring decisions are ordered before per-day content so that the remaining budget is defined at each step.

Realize and Validate. Before each REALIZE call, the adapter presents the candidate options that pass the deterministic constraint checks of the official evaluator given the current state, including local constraints, transport-mode consistency, trip-wide uniqueness of restaurants and attractions, and remaining budget, sorted by cost and capped at 30. VALIDATE is deterministic and applies the same checks to the selection, and the final budget-check predicate confirms completeness, total cost, and cuisine coverage.

Table 8: SDP configuration for TravelPlanner.

Parameter	Value	Description
Attempt budget	3	Max REALIZE attempts per predicate before replan
Max replans	5	Max replan cycles per predicate
Global step cap	60	Max total steps across the episode
Max options shown	30	Max options presented to the LLM per call

A.3 SCIENCEWORLD

ScienceWorld (Wang et al., 2022) is an interactive text-based environment in which the agent performs multi-step science experiments, with milestone-based partial-credit scoring. Unlike Trav-

elPlanner, every operator is LLM-based, with LLM-generated plans, free-form actions, and LLM validation, and the core loop is unchanged.

State and predicate plan. The state holds the latest observation, the current room description, the inventory, the score, and a short action history. At the start of each episode the LLM generates the predicate plan as free-form natural-language conditions, such as "the agent's inventory contains a metal pot" or "the water has been heated to a boil". To keep predicates grounded, the PROPOSE prompt includes the list of reachable rooms, and replanning includes the targets the simulator has previously rejected.

Realize and Validate. REALIZE issues one LLM call with the room description, inventory, target predicate, and recent actions, and returns a free-form action. VALIDATE executes the action and first applies simple rules, certifying all remaining predicates if the task is complete and returning $k = 0$ if the simulator rejects the action. Otherwise an LLM call receives the target predicate, the action, the resulting observation, and the score change, and returns the cascade count k, which cannot advance past the goal predicate unless the environment reports completion.

Replan and hyperparameters. Replanning reuses the PROPOSE prompt with the stuck predicate and its recent failures appended. The attempt budget is higher than in the other benchmarks because free-form actions are frequently rejected by the simulator.

Table 9: SDP configuration for ScienceWorld.

Parameter	Value	Description
Attempt budget	30	Max REALIZE attempts per predicate before replan
Max replans	5	Max replan cycles per predicate
Global step cap	500	Max total steps across the episode

A.4 ASSISTANTBENCH

AssistantBench (Yoran et al., 2024) is an open-ended web QA benchmark in which the agent searches the web, reads pages, and submits a single answer. There is no interactive environment, and the agent acts through external tool calls.

State and predicate plan. The state holds the retrieved search results and pages, the certified findings of each step, and the current answer. A decomposition call breaks the question into up to five steps, each of which collects candidates, filters a set by a criterion, extracts a fact, or computes the answer, followed by a finalize predicate that synthesizes the answer and a goal predicate that submits it.

Realize and Validate. For each step, REALIZE makes an LLM call that, given the question, prior findings, and previous failures on the step, produces a short list of web searches and page scrapes, executed through the Serper API. VALIDATE first has an LLM summarize the new evidence into a finding for the step, then has a second LLM call judge whether the finding addresses the step goal. After two consecutive rejections, a step whose finding contains substantive content is accepted rather than retried further.

Replan and hyperparameters. Replanning generates new queries from the stuck step and what has been tried, and re-attempts the original steps with the new evidence. If the finalize step yields an empty answer, it is retried with an instruction to return the closest match.

Table 10: SDP configuration for AssistantBench.

Parameter	Value	Description
Attempt budget	3	Max REALIZE attempts per predicate before replan
Max replans	2	Max replan cycles per predicate
Global step cap	30	Max total steps across the episode
Max subqueries	5	Max decomposition steps per question

A.5 HOTPOTQA

HotpotQA (Yang et al., 2018) is a multi-hop QA benchmark in which each question requires reasoning over two Wikipedia paragraphs. We use the open-domain setting, retrieving evidence with BM25 over roughly 5.2 million Wikipedia abstracts.

State and predicate plan. The state holds the question, the retrieved paragraphs, the certified hop findings with their supporting paragraph and bridge entity, and the current answer. The plan has four predicates, two hops generated by an LLM decomposition, an answer predicate, and a goal predicate, and the bridge entity certified at the first hop is passed to the second.

Realize and Validate. For each hop, REALIZE generates a search query from the question, the sub-question, and prior findings, retrieves paragraphs with BM25 and title matching, and extracts a finding with its cited paragraph. VALIDATE asks an LLM verifier whether the finding is supported by the cited paragraph and answers the sub-question, rejecting explicit non-answers before the call. If the cited title cannot be resolved, the verifier is applied to the retrieved paragraph that best matches the finding. The answer predicate rejects non-answers such as "unknown", and replanning regenerates only the hops not yet certified.

Table 11: SDP configuration for HotpotQA.

Parameter	Value	Description
Attempt budget	3	Max REALIZE attempts per predicate before replan
Max replans	2	Max replan cycles per predicate
Global step cap	20	Max total steps across the episode
BM25 top-k	10	Retrieved paragraphs per search query

A.6 MUSIQUE

MuSiQue (Trivedi et al., 2022) extends multi-hop QA to compositional chains of two to four hops. The adapter follows HotpotQA with three differences.

Plan length. The number of hops is taken from the hop count encoded in each task's identifier, giving plans of four to six predicates, and replanning regenerates only the hops not yet certified.

Retrieval. BM25 runs over the roughly 84,000 paragraphs of the MuSiQue splits, following the IRCoT protocol (Trivedi et al., 2023). When repeated queries fail, the adapter falls back to entity-only queries and then to a query built from the LLM's guess of the answer.

Answer validation. The answer predicate additionally rejects answers that repeat an entity from the question, return an intermediate bridge entity, exceed 15 words, or mismatch the question type, and a final LLM check confirms consistency with the certified hop chain.

Table 12: SDP configuration for MuSiQue.

Parameter	Value	Description
Attempt budget	3	Max REALIZE attempts per predicate before replan
Max replans	2	Max replan cycles per predicate
Global step cap	25	Max total steps across the episode
BM25 top-k	10	Retrieved paragraphs per search query
Corpus size	$\sim$84K	Unique paragraphs from all splits

B BENCHMARKS

B.1 TRAVELPLANNER

Evaluation protocol. We evaluate on all 180 validation tasks of TravelPlanner (Xie et al., 2024) in the tool-use setting, following the official scoring pipeline.

Baselines. ReAct, Reflexion, Plan-and-Solve, GoalAct, and TDP are reproduced by us with Gemini-3.1-flash-lite on all 180 tasks under the same pipeline. MIRROR (GPT-4o), EvoAgent, PMC, and ATLAS (Gemini-2.5-Pro) are taken from their original papers on the same split and scoring. SDP is run with Gemini-3.1-flash-lite for direct comparison against the reproduced baselines, and with GPT-4o for comparison at the scale of MIRROR.

B.2 ScienceWorld

Evaluation protocol. We follow the standard 30-task protocol of ScienceWorld (Wang et al., 2022) established by SwiftSage (Lin et al., 2023), with GPT-4 as the backbone for all methods and average score reported for Short, Medium, Long, and Overall task groups.

Baselines. SayCan, ReAct, CoT, Reflexion, EvoAgent, and Plan-and-Act are taken from their original papers, all using GPT-4 on the same protocol. TDP and RPMS are excluded because their evaluation protocols differ from the standard 30-task setup.

B.3 AssistantBench

Evaluation protocol. We evaluate on the 181-task test split of AssistantBench (Yoran et al., 2024) through the official leaderboard. The agent uses the Serper search API and URL-level scraping only, without browser rendering or form interaction.

Baselines. All seven baselines are taken from the official leaderboard on the same split and scoring. Infogent, Magentic-One, and ACP-Domain Agents use GPT-4o, as does SDP, and the remaining four use GPT-4-Turbo. Magentic-One additionally uses a browser-controlling sub-agent, which matters most on Hard-tier tasks that require multi-page browsing.

B.4 HotpotQA and MuSiQue

Evaluation protocol. Both HotpotQA (Yang et al., 2018) and MuSiQue (Trivedi et al., 2022) are evaluated in the open-domain setting on 1,000 validation examples each, with BM25 retrieval over a 5M-paragraph Wikipedia corpus as in IRCoT and PRISM, using GPT-4o as the backbone. For MuSiQue we use the answerable subset.

Baselines. All six baselines are taken from their original papers. RankZephyr, RankZephyr + CoT, and RankGPT are retrieve-then-read methods with reranking, SETR-CoT & IRI adds iterative retrieval, IRCoT interleaves retrieval with chain-of-thought, and PRISM uses agentic multi-step retrieval. IRCoT uses GPT-3 and all others use GPT-4o. PRISM additionally employs a learned reranker, whereas SDP uses BM25 alone.

C Experiment Details

C.1 Cost Accounting

Matched-backbone protocol. The main results follow each benchmark's established protocol, so backbones vary across benchmarks and baselines. To isolate the framework itself, the measurements here use a separate matched protocol in which every method runs on GPT-4o with identical tools, attempt budgets, and task sets, over three independent runs. Absolute scores therefore differ from the main tables. On ScienceWorld, for instance, SDP scores 54.5 with GPT-4o against 59.16 with GPT-4, and ReAct scores 36.4 under the same conditions. Section 4.3, Section 4.4, and Appendix C.2 use the same protocol.

Measurement. We compare SDP against ReAct on all five benchmarks. Calls and tokens are summed from the API usage fields, latency is wall-clock time per task including retrieval, and cost uses the official GPT-4o rates of $2.50 and $10.00 per 1M input and output tokens.

Cost tracks tokens, not calls. SDP issues more calls than ReAct on every benchmark, but cost does not follow call count. What determines tokens is how much context each call carries. SDP's operators read only local inputs, the current state and the next predicate, so each call sends a small

Table 13: Per-task cost of SDP against ReAct under the matched-backbone protocol. Cells are mean LLM calls with total tokens in parentheses. Latency is seconds per task and cost is US dollars per task, each reported as SDP / ReAct.

	SDP operators							
Benchmark	PROPOSE	REALIZE	VALIDATE	REPLAN	SDP total	ReAct total	Latency	Cost
TravelPlanner	0.0 (0.0k)	20.3 (15.8k)	0.0 (0.0k)	0.3 (0.2k)	20.6 (16.0k)	1.0 (10.4k)	9.5 / 9.4	0.040 / 0.032
ScienceWorld	1.0 (1.4k)	65.3 (111.9k)	44.8 (62.2k)	1.2 (1.8k)	112.3 (177.4k)	20.3 (17.1k)	80.8 / 20.0	0.456 / 0.050
AssistantBench	1.0 (0.8k)	6.5 (12.4k)	10.3 (21.4k)	0.2 (0.3k)	18.1 (35.0k)	11.3 (27.9k)	49.5 / 19.2	0.094 / 0.074
HotpotQA	1.0 (0.3k)	14.8 (15.5k)	7.6 (2.4k)	1.0 (0.5k)	24.8 (18.9k)	8.4 (15.0k)	71.9 / 65.5	0.059 / 0.042
MuSiQue	1.0 (0.7k)	16.1 (20.8k)	7.8 (4.3k)	1.0 (0.5k)	26.4 (26.5k)	20.0 (49.6k)	21.6 / 27.8	0.073 / 0.133

input and returns a short output. ReAct carries its growing interaction history at every step, so its per-call input expands with the horizon.

Certification compresses context, so the advantage grows with horizon. MuSiQue makes this visible. SDP is both cheaper and faster there despite issuing more calls. ReAct re-sends every paragraph it has read at each step, and its per-call input grows from roughly 1.2k to 4–5k tokens over a four-hop chain. SDP discards the source paragraph once a hop is certified and retains only the one-line conclusion, holding per-call input near 1k throughout. Certification acts as a compression point, and the advantage accumulates as the chain lengthens.

Overhead concentrates where verification is active. The extremes bound the range. ScienceWorld is the most expensive case, where all four operators are LLM-based and every attempt is judged, costing roughly nine times ReAct. This buys the gain from 36.4 to 54.5 reported above. TravelPlanner is the converse, since its validators are deterministic constraint checks, so VALIDATE contributes zero LLM cost and SDP runs at 1.25 times ReAct. The overhead is therefore not a fixed tax of the framework but a function of how much of the validation is deterministic.

C.2 VALIDATOR ROBUSTNESS

Injection protocol. VALIDATE is LLM-based on four of the five benchmarks, so a false positive certifies a predicate the observation does not support. We measure how far such errors propagate by injecting them directly. With probability ϵ, a rejected validation is overridden and forced to certify, so the agent advances on a premise the environment never supported. The sweep runs at $\epsilon \in \{0.05, 0.10, 0.15, 0.20\}$ against the uninjected baseline at $\epsilon = 0$, under the matched-backbone protocol of Appendix C.1. TravelPlanner is excluded because its validators are deterministic constraint checks with no LLM judgment to corrupt. The last two columns are measured over the injected cases only, reporting how often the agent subsequently rejected an action built on the false premise, and how often it recovered through replan to certify the goal.

Table 14: Score under injected validator false positives at rate ϵ, with the fate of the injected cases. Scores are mean $\pm$ 1 std over three runs. Rejected is the fraction of injected false positives that surfaced as a rejection at a later step, and Recovered the fraction from which the run still reached the goal.

Benchmark	$\epsilon = 0$	0.05	0.10	0.15	0.20	Rejected (%)	Recovered (%)
ScienceWorld (AS)	54.5 ± 3.6	52.9 ± 1.4	52.4 ± 1.2	48.8 ± 1.8	45.9 ± 3.8	87.3	72.6
AssistantBench (Acc)	31.9 ± 0.3	29.9 ± 0.9	26.3 ± 1.2	25.7 ± 0.6	24.7 ± 0.7	32.1	49.5
HotpotQA (F1)	66.3 ± 3.1	65.0 ± 0.9	61.9 ± 2.7	57.1 ± 2.1	54.3 ± 1.5	56.3	61.7
MuSiQue (F1)	52.2 ± 1.7	51.1 ± 0.6	49.5 ± 2.2	46.1 ± 1.1	42.9 ± 2.5	78.9	63.8

A false positive corrupts the record, not the environment. Degradation is graded rather than catastrophic on every benchmark, and at $\epsilon = 0.20$ SDP still exceeds uncontaminated ReAct on all four. The reason is that certification is a claim about the world, not an intervention in it. A false positive writes an unsupported predicate into the certified state, but the environment is unchanged, so the next action is built on a premise the environment will not honor. That action fails, VALIDATE rejects it, and the loop enters the same retry and replan path it uses for any other failure. The mechanism that catches an ordinary execution failure is the mechanism that catches a corrupted state.

How much is caught depends on how strongly the environment can refute. The rejection rates vary widely across benchmarks, and the variation is structural. ScienceWorld surfaces the most at 87.3%, since the simulator holds persistent physical state and a false claim that a container holds water is contradicted the moment an action needs that water. MuSiQue follows at 78.9%, where a false bridge entity poisons the query of every remaining hop. HotpotQA is lower at 56.3% for a reason of length rather than kind, as a two-hop chain leaves little room after an injection near the end. AssistantBench is lowest at 32.1%, since open-ended web findings constrain the state loosely enough that a plausible but empty summary can pass unchallenged. The relative score loss mirrors this ordering, with the benchmarks that surface less losing a larger share of their uninjected score, which identifies downstream validation rather than validator accuracy as the protective mechanism.

Implications for validator design. Two consequences follow. First, soundness of a single validation is not what bounds the framework, since the loop re-tests every premise at the next step and a corrupted state has to survive repeated contact with the environment to persist. Second, the residual risk concentrates in environments whose observations constrain the state weakly, and the remedy there is to make validation decidable rather than to make the validator more accurate. Definition 1 fixes VALIDATE's interface and not its implementation, and TravelPlanner illustrates that end of the spectrum, with deterministic constraint checks that leave no LLM judgment to corrupt.

D DISCUSSION

Limitations. SDP inherits the limits of its LLM-based operators. VALIDATE can produce false positives on superficially matching observations, so the certified-state guarantee is only as strong as its soundness, and PROPOSE bounds plan quality by the LLM's ability to decompose goals into reachable predicates. Natural-language predicates also restrict which conditions can become states, though many admit an observable proxy and the rest fail conservatively, stalling the loop rather than advancing on a guess, and this boundary belongs to the current instantiation since Definition 1 fixes VALIDATE's interface and not its implementation. Finally, SDP issues more LLM calls per step than reactive baselines, though cost tracks tokens rather than calls (Appendix C.1).

Future work. MDP structure is valuable because it makes a class of operations well-defined, and SDP brings three to language agents: per-predicate credit assignment through k, modular operator replacement through the fixed interfaces of Definition 1, and progress measurement through the certified trajectory. Follow-up work can train a state generator for PROPOSE, learn an action policy over Σ for REALIZE, or apply offline RL to certified tuples (s_t, a_t, k, s_{t+k}). The unboundedness of Σ does not obstruct this, since value and transition functions take predicate text as input and only the tens of predicates a task populates are ever evaluated. The simplest is a reachability head predicting whether a candidate predicate will be certified, with every certified tuple as a supervised sample.