A MINIMAL AGENT FOR PLANNING BASED ON REUSABLE DOMAIN STRATEGIES

Anonymous authors
Paper under double-blind review

ABSTRACT

In planning, an *agent* must compute a sequence of actions that transforms an initial state into a goal state. Frontier large language models (LLMs) now compete with state-of-the-art planners, but they are proprietary and expensive to run, while small open-weight LLMs remain behind. Planning problems, however, are organized into domains whose instances share the same actions and structure. In this paper, we exploit this structure with a minimal planning agent that adds two components to a small LLM, a *domain strategy* and feedback on its mistakes. The domain strategy is a natural-language text that explains how to compute a plan for any instance of the domain. The agent uses the strategy to guide the LLM, which returns a plan. A sound plan validator either accepts the plan or rejects it with feedback, and the LLM revises the plan with this feedback until the plan is valid or the call budget runs out. A frontier LLM writes the strategy once per domain, and the agent reuses it for every instance of that domain. We evaluate the agent with two small open-weight LLMs on the International Planning Competition (IPC) 2023 benchmark and on a benchmark of novel domains. The strategy increases the number of solved instances for both models, on both benchmarks, and at every budget. With up to five calls and feedback, GPT-OSS-20B improves from 28.3% to 40.6% on the IPC benchmark and from 0.4% to 17.4% on the novel benchmark. On the IPC benchmark, Gemma-4-31B solves 62.5% of the instances, more than a strong planner (56.7%) and GPT-5 (56.9%). To reach the coverage of GPT-5, it costs 9.8 times less than GPT-5 and 13.0 times less than Gemini 3.1 Pro. With feedback and up to 42 calls, Gemma-4-31B solves the same number of instances as the frontier model Gemini 3.1 Pro at a fraction of the cost. Overall, with only a strategy and feedback, small open-weight LLMs compete with planners and frontier models, and the computation spent once per domain is reused across all of its instances. This suggests that the same agent could be applied to problems beyond planning whose instances share a common structure from which a strategy can be defined.

1 INTRODUCTION

In planning, an agent computes a sequence of actions that transforms an initial state into a state satisfying a goal condition (Ghallab et al., 2004). Finding such a sequence can be computationally hard: even deciding whether one exists is PSPACE-complete for propositional STRIPS planning (Bylander, 1994). This makes planning an interesting benchmark for large language models (LLMs), which need to choose and order hundreds of actions while tracking the state after each one. In LLM *end-to-end planning*, the model receives a planning problem and must return a valid plan. Until recently, however, LLMs could not reliably solve planning instances (Valmeekam et al., 2023b).

Recent frontier LLMs now compete with strong planners while remaining far less efficient (Corrêa et al., 2025). On 360 instances from eight domains of the IPC 2023 Learning Track (Taitler et al., 2024), GPT-5 (OpenAI, 2025) solves 205 instances, while LAMA (Richter & Westphal, 2010), a strong planner commonly used in recent LLM planning research, solves 204 (Corrêa et al., 2025). Gemini 3.1 Pro (Google DeepMind, 2026) solves 245 instances, surpassing Scorpion Maidu (Corrêa et al., 2023), the winner of the IPC 2023 satisficing track, which solves 234 (Corrêa et al., 2025). These frontier models are proprietary and expensive to run. Small open-weight models such as

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

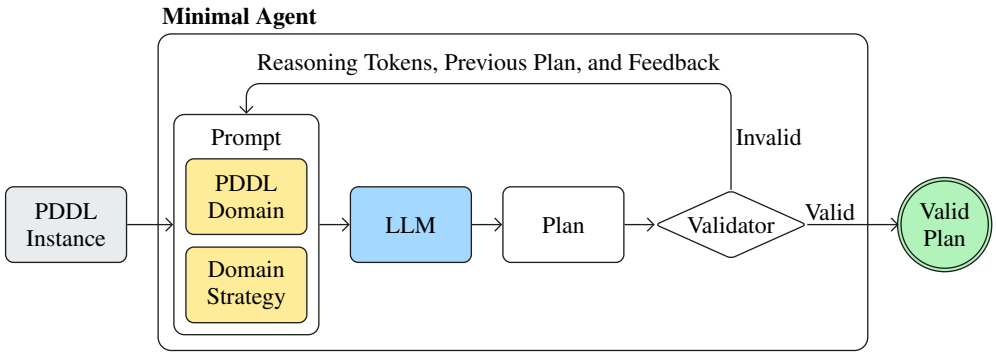

Figure 1: The agent receives a PDDL instance as input and returns a plan as output. The prompt combines the instance with the PDDL domain and the domain strategy. The LLM returns a plan, and a sound plan validator either accepts it or rejects it with feedback identifying the first inapplicable action or an unsatisfied goal. After a rejection, the LLM is called again with its reasoning tokens, the invalid plan, and the feedback, until the plan is valid or the call budget runs out.

GPT-OSS-20B (OpenAI, 2025) and Gemma-4-31B (Gemma Team, 2026) cost less and allow local deployment and closer inspection, but their planning performance remains behind.

Planning problems, however, are organized into domains whose instances share the same actions and structure, while differing in their objects, initial states, and goals. This shared structure enables the description of a *domain strategy*, a natural-language text that explains how to compute a plan for instances of the domain. In Miconic, for example, a lift must carry passengers to their destination floors. A strategy can tell the model to serve one passenger at a time, moving the lift to the passenger's floor, boarding them, and taking them to their destination before serving the next passenger. The same strategy applies regardless of the number of passengers or floors.

We exploit such strategies to build a minimal agent, as in Requena et al. (2026), by adding a domain strategy and feedback on mistakes to a small open-weight model. A frontier LLM writes the strategy once per domain, and the agent reuses it for all instances of that domain. The agent takes a planning instance as input and uses its domain description and strategy to guide the small model in producing a plan. A sound validator (Howey & Long, 2003) accepts the plan or rejects it with feedback identifying the first inapplicable action or an unsatisfied goal. The model uses this feedback to revise the plan until it is valid or the call budget runs out. Figure 1 shows the structure of the minimal agent.

We evaluate the agent with GPT-OSS-20B and Gemma-4-31B, two small open-weight models from different families, on two benchmarks. The IPC benchmark contains the 360 instances above, which allows a direct comparison with the planners and frontier models reported by Corrêa et al. (2025). The novel benchmark contains 270 instances from three domains released after the models were trained, which rules out data contamination. We compare end-to-end planning with *strategy-guided planning*, which adds the domain strategy to the input of the model. Each model runs under three inference budgets: one call (pass@1), five independent calls (pass@5), and up to five calls in which the model revises its plan with the feedback of the validator (feedback@5). We measure *coverage*, the number of instances for which the model returns a valid plan within the budget, as well as the cost, the number of output tokens, and the plan length.

The domain strategy increases the coverage of both models on both benchmarks and at every budget. At feedback@5, it raises the coverage of GPT-OSS-20B from 102 to 146 IPC instances and from 1 to 47 novel instances, and that of Gemma-4-31B from 211 to 225 IPC instances and from 109 to 174 novel instances. With the strategy at feedback@5, Gemma-4-31B solves more IPC instances than LAMA (204) and GPT-5 (205) (Corrêa et al., 2025). To reach the coverage of GPT-5, it costs 9.8 times less than GPT-5 and 13.0 times less than Gemini 3.1 Pro. With feedback and up to 42 calls, Gemma-4-31B solves the same number of instances as the frontier model Gemini 3.1 Pro and surpasses Scorpion Maidu, a state-of-the-art planner. Therefore, with our minimal agent, a small open-weight model competes with planners and proprietary frontier models at a fraction of the cost.

2 BACKGROUND

In this paper, we consider *classical planning*, in which a single agent acts in a fully observable, discrete environment with deterministic actions (Ghallab et al., 2004). A state is a set of atoms, the facts that hold in the world. An action has a precondition, the atoms that must hold for the action to be applicable, and effects, the atoms it adds to or removes from the state. A *planning problem* consists of the actions, an initial state, a goal, and a set of atoms that must hold in the final state. A plan is a sequence of actions, and it is valid if each action is applicable in the state reached by the previous ones, and the final state satisfies the goal. In this work, we focus on *satisficing planning*, in which any valid plan is accepted.

Planning problems are described in the Planning Domain Definition Language (PDDL) (McDermott, 2000; Haslum et al., 2019), which splits each problem into two files. The PDDL domain defines the types of objects, the relations between them (called predicates), and the actions, and is shared by every instance of the domain. The PDDL instance defines the objects, the initial state, and the goal. In Miconic, for example, the types are passenger and floor, and the predicate `lift-at` says where the lift is, as in the atom `(lift-at f1)`, where `f1` is a floor. The action `(up f1 f2)` moves the lift up from floor `f1` to floor `f2`. An instance gives the passengers and floors, where the lift and each passenger start, and the goal is that every passenger reaches their destination.

Deciding whether a plan exists is PSPACE-complete for propositional STRIPS planning (Bylander, 1994) and EXPSPACE-complete for lifted STRIPS planning, in which actions have parameters (Erol et al., 1995). Since the state space is too large to search exhaustively, a common way to solve planning problems is via heuristic search in the state space (Bonet & Geffner, 2001; Hoffmann & Nebel, 2001; Helmert, 2006b). More informative heuristic functions make the search more efficient and lead to better planners. Many planning domains, however, admit polynomial algorithms that compute a plan when optimality is not required (Helmert, 2003; 2006a), such as the Miconic strategy of the introduction. Others, such as Sokoban, are PSPACE-complete (Culberson, 1997), and our benchmarks include both kinds.

3 A MINIMAL PLANNING AGENT

In this section, we describe our minimal planning agent. Requena et al. (2026) show that, in automated theorem proving, a minimal agent that revises its proofs with the feedback of the Lean compiler competes with far more complex provers at a fraction of their cost. We follow the same principle for planning. Our agent adds only two components to a small open-weight LLM: a domain strategy and feedback on its mistakes. A frontier LLM writes the strategy once per domain, and the agent reuses it for every instance of that domain. The agent receives a PDDL instance as input and prompts the model with the PDDL domain, the PDDL instance, and the domain strategy. The model returns a complete plan in a single call, and a sound plan validator either accepts the plan or rejects it with feedback that identifies the first inapplicable action or the goal atoms that do not hold. When the plan is invalid, the agent calls the model again with the reasoning trace of that call, the invalid plan, and the feedback. The model sees only its last attempt, so the prompt does not grow with the number of calls. The agent repeats this until the plan is valid or the inference budget runs out. The agent uses two prompts. The initial prompt asks the model for a first plan, and after each rejection, the feedback prompt asks it for a revised plan. The strategy is written beforehand, once per domain, by a frontier LLM with a separate prompt. We describe the two prompts, plan validation, and the generation of the strategies below.

Initial Prompt. The prompt instructs the model to compute a plan by following the strategy. It then contains:

1. the definitions of the planning terms used in the prompt;
2. the format of the response;
3. the PDDL domain and the PDDL instance;
4. the domain strategy;
5. a checklist of common pitfalls.

Items 1, 2, and 5 are the same across all domains and instances, item 3 is the planning problem, and item 4 is the domain strategy in natural language. When we evaluate an LLM in end-to-end planning, without the domain strategy, the prompt omits item 4 and the instructions that refer to the strategy, and is otherwise identical.

Plan Validation. We check every plan with VAL (Howey & Long, 2003), a sound plan validator. VAL receives the PDDL domain, the PDDL instance, and the plan, and applies the actions one at a time from the initial state. At the first action whose precondition does not hold, VAL stops, and its feedback names that action and its unsatisfied precondition. If every action applies, VAL checks the goal in the final state. It accepts the plan if the goal holds, and otherwise, its feedback lists the goal atoms that do not hold.

Feedback Prompt. When VAL rejects a plan, the agent calls the model again with the feedback prompt. This prompt repeats the initial prompt and appends the reasoning trace and the invalid plan of the previous call, followed by VAL's feedback. It asks the model to use the feedback to compute a new plan that avoids the mistakes of the previous one. Appendix A gives the end-to-end planning prompt, the strategy-guided planning prompt, and the feedback prompt in full.

Strategy Generation. Before the agent runs, a frontier LLM writes the strategy of each domain in two calls. The first call asks for a Python program that solves instances of the domain. We test the program on the training instances of the domain and validate every plan with VAL. The training instances are out of distribution with respect to the test instances: they are much smaller and differ in structure (Table 1). The prompt for the program contains:

1. the PDDL domain;
2. three instances from the training set of the domain;
3. for each of the two example domains, Gripper and Logistics: the PDDL domain, one instance, and a Python program that solves instances of the domain.

Items 1 and 2 describe the domain, and item 3 shows what such a program looks like. The second call asks the frontier LLM to translate the program into a natural-language strategy. The strategy-guided planning prompt contains only this translation, never the program. Appendix B gives the prompts we used. Section 4.1 describes three of the strategies.

4 Experiments

We evaluate our minimal agent following recently proposed guidelines for LLM planning research (Katz et al., 2025): we compare with state-of-the-art planners, evaluate on novel domains to rule out data contamination, and accept only the plans that a sound validator confirms.

Benchmarks. We evaluate on two benchmarks, for a total of eleven domains, each with instances of increasing difficulty. Table 1 gives the range of the main parameter of each domain over its training and test instances. Appendix C describes each domain. The *IPC benchmark* consists of eight domains from the IPC 2023 Learning Track (Taitler et al., 2024): Blocksworld, Childsnack, Floortile, Miconic, Rovers, Sokoban, Spanner, and Transport. We use the training instances of the Learning Track as the training set and the instances from Corrêa et al. (2025) as the test set, 45 per domain for a total of 360, so that our results are directly comparable to the planners and frontier models they report (Table 3).

The *novel benchmark* consists of three domains released after the models we evaluate were trained: Cisterns, Orrery, and Qubit-Routing. This rules out data contamination, so the gains on these domains show that the strategy helps on domains the models have never seen. Each domain has 90 test instances, for a total of 270, and the PDDL files of all three will be released with the code. In *Cisterns*, water is distributed unevenly across cisterns connected by siphons. Transfers are permitted only under level constraints, and the goal specifies the final levels of selected cisterns. In *Orrery*, coupled wheels each display one of K marks. An action advances one wheel while retreating another, and the goal returns every wheel to its reference mark while preserving the total position. In *Qubit-Routing*, logical qubits occupy positions on a hardware connectivity graph. Swaps move them until the operands of each gate are adjacent, allowing the gates to be executed in order.

The Agent. The agent uses two small open-weight models from different families: GPT-OSS-20B (OpenAI, 2025) and Gemma-4-31B (Gemma Team, 2026). We run both models in their default

	Domain	Parameter	Training	Test
IPC	Blocksworld	blocks	$[2, 29]$	$[5, 477]$
	Childsnack	children	$[1, 10]$	$[4, 284]$
	Floortile	tiles	$[2, 40]$	$[12, 899]$
	Miconic	passengers	$[1, 10]$	$[1, 470]$
	Rovers	rovers	$[1, 4]$	$[1, 29]$
	Sokoban	boxes	$[1, 4]$	$[1, 78]$
	Spanner	spanners	$[1, 10]$	$[1, 474]$
	Transport	vehicles	$[1, 7]$	$[3, 49]$
Novel	Cisterns	tanks	$[6, 16]$	$[16, 50]$
	Orrery	wheels	$[4, 16]$	$[14, 46]$
	Qubit-Routing	qubits	$[6, 12]$	$[15, 36]$

Table 1: The main parameter of each domain over its training and test instances.

configuration. The models have no access to external tools and cannot execute code. In Tables 2 and 3, the agent uses domain strategies written by GPT-5.6 Sol (OpenAI, 2026), one strategy per domain. The 22 calls that produce the eleven strategies used on the 630 test instances cost $1.75 in total. Section 4.1 repeats the generation with Claude Opus 5 (Anthropic, 2026).

Frontier LLMs. We compare our agent with several proprietary LLMs. We call *frontier models* the best model of each family at the time of its evaluation: GPT-5 (OpenAI, 2025) and Gemini 3.1 Pro (Google DeepMind, 2026). We also report earlier models of the same families, GPT-4.1 (OpenAI, 2025), Gemini 2.5 Pro (Comanici et al., 2025), and Gemini 3 Pro (Google DeepMind, 2025), to show how quickly LLM planning has improved. Each model receives the PDDL domain, the PDDL instance, two example planning problems with valid plans from other domains, and a checklist of common pitfalls, and returns a plan in a single call, without a strategy or feedback. We take their results from Corrêa et al. (2025) and do not rerun them.

Planners. We compare our agent with two classical planners, LAMA (Richter & Westphal, 2010) and Scorpion Maidu (Corrêa et al., 2023). Both receive the PDDL domain and the PDDL instance and solve the problem with heuristic search in the state space (Section 2), so every plan they return is valid. LAMA is a strong planner baseline common in studies of LLM planning. Scorpion Maidu won the satisficing track of the IPC 2023 and combines, among other components, a portfolio of search configurations, search algorithms that reward exploration, and several heuristic functions. We take their results from Corrêa et al. (2025), who ran them with a limit of 30 minutes and 8 GiB of memory per instance.

Budget and Metrics. The *inference budget* is the maximum number of LLM calls per instance. Pass@k generates k independent plans for one instance, whereas feedback@k generates one plan and revises it with the feedback prompt until it is valid or k calls have been made. Pass@5 and feedback@5 have the same inference budget of five calls, but feedback@5 stops at the first valid plan and may therefore use fewer. We report *coverage*, the number of instances for which the model returns at least one valid plan within the budget, under pass@1, pass@5, and feedback@5, and for Gemma-4-31B on the IPC benchmark, also under feedback@42. We also report the cost of the calls and the number of output tokens, and we measure plan quality by plan length, the number of actions in a plan, where shorter plans are better.

4.1 Evaluation of the Minimal Agent

In this section, our aim is to measure how much each component of the minimal agent, the domain strategy, and the feedback improves the planning performance of a small open-weight LLM. We first compare end-to-end planning with strategy-guided planning, which differ only in whether the prompt includes the strategy, at pass@1, pass@5, and feedback@5. We then compare pass@5 with feedback@5, which have the same budget but differ in whether the model receives feedback on its invalid plans. Strategy-guided planning at feedback@5 corresponds to the full agent. We then analyze how the strategy affects output tokens and plan length, and qualitatively analyze representative

		GPT-OSS-20B						Gemma-4-31B					
		pass@1		pass@5		feedback@5		pass@1		pass@5		feedback@5	
	Domain	E2E	Strat.	E2E	Strat.	E2E	Strat.	E2E	Strat.	E2E	Strat.	E2E	Strat.
IPC	Blocksworld	4	**10**	11	**19**	8	**19**	25	**29**	29	**31**	29	**34**
	Childsnack	12	**16**	**25**	22	24	**27**	26	**32**	**34**	33	34	**36**
	Floortile	0	**1**	**5**	3	2	**6**	2	**5**	7	**9**	9	**14**
	Miconic	16	**20**	22	**29**	20	**27**	25	**34**	30	**35**	31	**34**
	Rovers	3	3	6	6	**8**	7	**8**	7	12	12	15	15
	Sokoban	0	**5**	1	**9**	4	**6**	11	**12**	14	14	14	14
	Spanner	9	**25**	22	**34**	19	**36**	38	**44**	45	45	45	45
	Transport	5	**8**	**13**	12	17	**18**	22	**25**	27	**31**	**34**	33
	Total (IPC)	49	**88**	105	**134**	102	**146**	157	**188**	198	**210**	211	**225**
Novel	Cisterns	0	0	0	0	0	0	0	0	**1**	0	**10**	3
	Orrery	0	**16**	2	**54**	0	**44**	12	**86**	33	**90**	21	**90**
	Qubit-Routing	1	1	**3**	2	1	**3**	45	**51**	72	**74**	78	**81**
	Total (Novel)	1	**17**	5	**56**	1	**47**	57	**137**	106	**164**	109	**174**
	Total (Combined)	50	**105**	110	**190**	103	**193**	214	**325**	304	**374**	320	**399**

Table 2: Coverage of GPT-OSS-20B and Gemma-4-31B on the eight IPC domains, with 45 instances each, and on the three novel domains, with 90 instances each. E2E is end-to-end planning, Strat. is strategy-guided planning, and Strat. at feedback@5 is the full agent. Each cell is the number of instances solved, and bold marks the higher of E2E and Strat. at the same budget, with ties unmarked.

domain strategies. Finally, we test whether the gains depend on the frontier LLM that produces the strategies. Table 2 gives the coverage of each model in every domain.

Including the domain strategy in the prompt increases the coverage of both models on both benchmarks and at every budget. With pass@1, GPT-OSS-20B nearly doubles its coverage on the IPC benchmark, from 49 to 88 instances, and Gemma-4-31B goes from 157 to 188. Both models improve in every IPC domain except Rovers, where GPT-OSS-20B solves the same number of instances and Gemma-4-31B loses one. On the novel benchmark, whose domains the models have never seen, GPT-OSS-20B goes from 1 instance to 17, and Gemma-4-31B from 57 to 137. Thus, the domain strategy helps both models, even in domains they have never seen.

We now evaluate the effect of the feedback. With the same budget of five calls, pass@5 generates five independent plans, whereas feedback@5 revises the last invalid plan with the feedback of the validator. With the strategy, the coverage of GPT-OSS-20B increases from 190 instances at pass@5 to 193 with feedback, and that of Gemma-4-31B from 374 to 399. Thus, the full agent, with both the strategy and the feedback, has the highest coverage of all configurations for both models.

Beyond coverage, we examine how the strategy affects the number of output tokens and the plan length. Since the strategies are general rather than optimal, the plans could be longer. Figure 2(a) and (b) compare both measures at pass@1 on the IPC instances that a model solves both with and without the strategy: 40 instances for GPT-OSS-20B and 151 for Gemma-4-31B. With the strategy, both models spend fewer output tokens on most of these instances, GPT-OSS-20B on 26 of its 40 and Gemma-4-31B on 95 of its 151, and their totals fall by 20.9% and 5.7%. Plan length changes little. With the strategy, the plans of GPT-OSS-20B are shorter on 14 of the 40 instances and longer on 13, with 11.0% fewer actions in total, and those of Gemma-4-31B are shorter on 29 of the 151 and longer on 60, with 4.7% more actions in total. Thus, the gain in coverage comes with fewer output tokens and a small change in plan length.

We now describe the strategies that GPT-5.6 Sol generated for three IPC domains, Floortile, Transport, and Rovers. In *Floortile*, robots paint the tiles of a grid in two colors. A robot moves onto a clear tile, one that is neither painted nor occupied, and paints only the tile directly above or below it. Since a painted tile cannot be entered, painting in the wrong order can cut a robot off from the tiles it still has to paint. The strategy instructs the model to paint the rows from the top down, each tile from the tile below it, and to move the robots of a row down before painting it, so the robots stay in the unpainted rows and never need to cross a painted tile. In *Transport*, vehicles with limited capacity

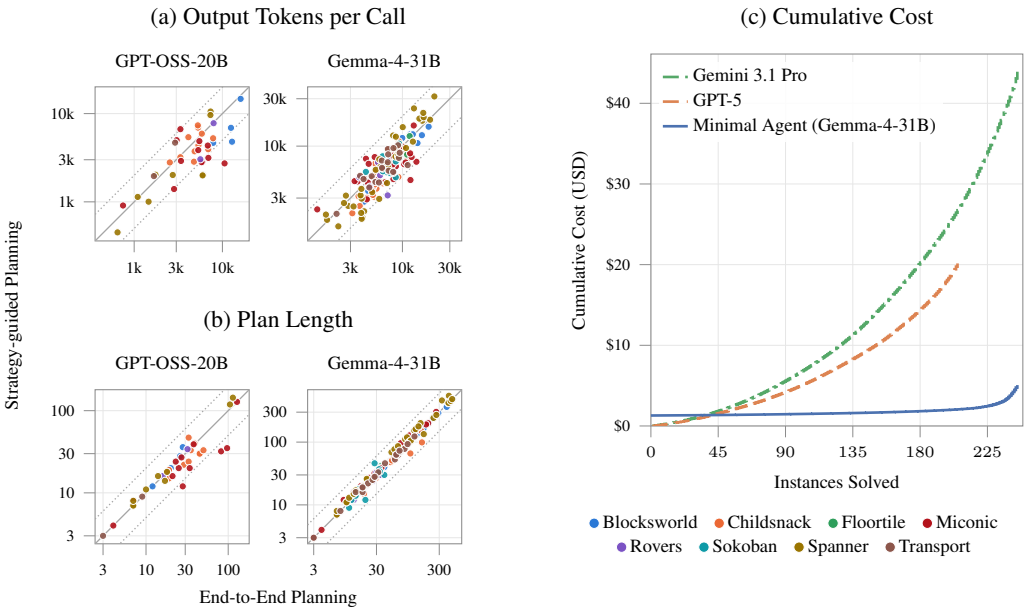

Figure 2: Output tokens per call (a) and plan length (b) of strategy-guided planning against end-to-end planning at pass@1. Each point is an IPC instance that the model solves both with and without the strategy, 40 instances for GPT-OSS-20B and 151 for Gemma-4-31B. Both axes are logarithmic, each panel is scaled to its own data, and the color in (a) and (b) identifies the domain. (c) Cumulative cost of the IPC instances solved by GPT-5 and Gemini 3.1 Pro, with one call per instance, and by the minimal agent with Gemma-4-31B.

deliver packages over a network of roads. The strategy instructs the model to deliver the packages one at a time, each with the vehicle that minimizes the drive to the package plus the drive from there to its destination. In *Rovers*, rovers drive along their own paths between waypoints, collect soil and rock samples and images of objectives, and must send this data to a lander. The strategy breaks the task into higher-level operations, each a shortest drive followed by sampling, calibration, imaging, or communication. It instructs the model to compare different orders of these operations and to prefer those with fewer actions so far and fewer goals whose data is still to be collected or sent, until every goal is communicated.

The strategies in our experiments were written by GPT-5.6 Sol. To test whether the gains depend on this choice, we also generate the eight IPC strategies with Claude Opus 5, following Section 3, and evaluate both models with them at pass@1, pass@5, and feedback@5.[1] The GPT-5.6 Sol strategies lead to higher coverage at every budget, for GPT-OSS-20B (88 vs. 72 at pass@1, 134 vs. 121 at pass@5, and 146 vs. 131 at feedback@5) and for Gemma-4-31B (188 vs. 165, 210 vs. 204, and 225 vs. 222). Still, with the Claude Opus 5 strategies, both models solve more instances at every budget than in end-to-end planning. Thus, the gains of the strategy are not specific to GPT-5.6 Sol, although the model that produces the strategy affects the improvement.

4.2 Comparison with Planners and Frontier LLMs

We now compare the full agent with the planners and frontier LLMs on the IPC benchmark, first in coverage and then in cost. To evaluate how much it costs to match the coverage of the best frontier model, we also raise the budget of Gemma-4-31B, the stronger of the two models, until it solves as many instances as Gemini 3.1 Pro, which it does at feedback@42. Table 3 gives the coverage of each planner and LLM in every domain, and Figure 2(c) gives the cost.

Table 3 shows the coverage of the planners, the proprietary LLMs, and the full agent on the IPC benchmark. The planners solved 204 (LAMA) and 234 (Scorpion Maidu) of the 360 instances. Within each family of proprietary LLMs, coverage improves quickly from one version to the next: from 20

[1]We will release both sets of domain strategies with the code upon publication.

		Planners		GPT		Gemini Pro			Minimal Agent OSS	Minimal Agent Gemma	
	Domain	LAMA	Maidu	4.1	5	2.5	3	3.1	f.@5	f.@5	f.@42
IPC	Blocksworld	28	29	3	21	28	35	34	19	34	**38**
	Childsnack	18	23	6	38	41	41	**44**	27	36	39
	Floortile	10	21	0	10	4	**26**	25	6	14	16
	Miconic	**45**	**45**	6	35	20	31	36	27	34	34
	Rovers	**34**	31	0	14	6	11	16	7	15	20
	Sokoban	21	**22**	0	15	11	14	15	6	14	16
	Spanner	15	30	3	**45**	31	**45**	**45**	36	**45**	**45**
	Transport	33	33	2	27	20	30	30	18	33	**37**
	Total (IPC)	204	234	20	205	161	233	**245**	146	225	**245**

Table 3: Coverage of the minimal agent on the IPC benchmark against the planners and LLMs of Corrêa et al. (2025), which call each LLM once per instance. OSS is GPT-OSS-20B, and Gemma is Gemma-4-31B, both at feedback@5 (f.@5), and Gemma-4-31B is also shown at feedback@42 (f.@42). Maidu is Scorpion Maidu. Bold marks the highest coverage in each row.

instances with GPT-4.1 to 205 with GPT-5, and from 161 with Gemini 2.5 Pro to 233 with Gemini 3 Pro and 245 with Gemini 3.1 Pro. In end-to-end planning with one call, GPT-OSS-20B solves 49 instances (Table 2), more than GPT-4.1 but far fewer than GPT-5. With the full agent and only five calls, it solves 146, much closer to GPT-5. In end-to-end planning with one call, Gemma-4-31B solves 157 instances, slightly fewer than Gemini 2.5 Pro (161). With the full agent and five calls, it solves 225, more than Gemini 2.5 Pro, LAMA, and GPT-5, and close to Gemini 3 Pro (233). With 42 calls, it solves 245, as many as Gemini 3.1 Pro and more than Scorpion Maidu. Thus, the full agent considerably improves the coverage of small open-weight models, which then compete with a state-of-the-art planner.

The agent with Gemma-4-31B reaches the coverage of Gemini 3.1 Pro with many more calls, up to 42 per instance, instead of one. To compare their cost, we price every call at the OpenRouter list rates of each model. Figure 2(c) shows the least each model spends to solve a given number of instances. To solve as many instances as GPT-5 (205), the agent with Gemma-4-31B spends $2.06, against $20.15 for GPT-5 and $26.74 for Gemini 3.1 Pro, 9.8 and 13.0 times less. To solve as many as Gemini 3.1 Pro (245), it spends $5.07, against $43.92 for Gemini 3.1 Pro, 8.7 times less. These costs include the $1.30 spent once to write the eight strategies. Although the agent makes more calls, each call uses a much smaller model and is much cheaper. Thus, the agent brings a small open-weight model to the coverage of the best frontier LLM at a fraction of the cost.

4.3 Limitations

The planners remain far more efficient than the agent: they run on a standard CPU with at most 8 GiB of memory, whereas running Gemma-4-31B requires GPUs with tens of gigabytes of memory. Still, this difference is much smaller than for frontier LLMs, which presumably require vastly more computational resources. The strategy guides the model but does not constrain it, and the model can still deviate from the strategy and return an invalid plan. The validator rejects such plans, and the feedback lets the model revise them. Finally, the agent relies on three properties of classical planning: a formal description of each problem, a sound validator that explains why a solution is invalid, and families of problems that share a structure from which a strategy can be written once. Applying the agent to other problems requires the same three properties.

5 Related Work

Early LLMs, when asked for a plan directly, solved only a few planning instances (Valmeekam et al., 2023a;b). Later work improved their planning by adding information to the prompt, such as examples (Bohnet et al., 2024), including examples whose plans resemble a draft plan written by the model (Zhao et al., 2025). Frontier models now compete with state-of-the-art planners (Corrêa et al., 2025), but small open-weight models are still behind.

Feedback from a validator can help a model repair an invalid plan, as shown for the frontier models of their time on small Blocksworld instances (Valmeekam et al., 2023b; 2024). The feedback, however, must come from a sound validator. Stechly et al. (2025) show that GPT-4 cannot validate its own plans, since it rejects valid plans and accepts invalid ones. They also show that calling the model again with the initial prompt until the validator accepts a plan already solves many of the instances it first failed, even though the model is never told what is wrong. Our agent uses VAL as the validator, and we show that, together with the strategy, feedback also helps small open-weight models on instances whose plans have more than a thousand actions (Section 4.1).

Other works also give an LLM knowledge that is written once for a task and reused on each of its instances. This knowledge can take the form of prompts or guidelines induced from examples or trajectories of the task (Gao et al., 2024; Agrawal et al., 2026; Fu et al., 2024), or of a manual or pseudocode that a stronger model writes for a weaker one (Chen et al., 2024; Chae et al., 2024). The evidence for this transfer comes from earlier models on other tasks and is mixed, since a weaker model does not always benefit from principles written by a stronger one (Zhang et al., 2024). Our strategy explains how to compute a plan for every instance of a domain, and we show that small open-weight models can follow it over hundreds of actions to produce plans that a sound validator accepts.

The work closest to ours is Borro et al. (2025), which also gives a smaller model a strategy written by a larger one, call it again with feedback that names the first action that cannot be executed, and show that this reduces cost. Their planning experiments use Blocksworld with small instances and two proprietary models of the same family, with o1 writing the strategies and o1-mini following them. Our work differs in three ways. The agent uses small open-weight models from different families. Each strategy is derived from a program whose plans we check on training instances, and we evaluate the agent on eleven domains, including three released after the models were trained and instances whose plans have more than a thousand actions. We also compare it with state-of-the-art planners and frontier LLMs.

6 Conclusion

We presented a minimal agent that helps a small open-weight LLM plan with two components: a domain strategy that a frontier LLM produces once per domain, and feedback from a sound validator on invalid plans. The strategy increases the coverage of both models at every budget, on the IPC benchmark, and on the benchmark of novel domains released after the models were trained. At feedback@5, it raises the coverage of GPT-OSS-20B from 102 to 146 IPC instances and from 1 to 47 novel instances, and that of Gemma-4-31B from 211 to 225 and from 109 to 174. Combined with the strategy, the feedback gives the full agent the highest coverage of all configurations for both models.

On the IPC benchmark, the full agent with Gemma-4-31B solves more instances than LAMA and GPT-5, at a cost 9.8 times lower than that of GPT-5, and, with more calls, as many as Gemini 3.1 Pro and more than Scorpion Maidu. The planners remain more efficient, but the agent brings a small open-weight model close to the state of the art. The strategy helps both models even in domains they have never seen, where data contamination is ruled out, and it does so with fewer output tokens. Its gains are also not specific to GPT-5.6 Sol, since the strategies produced by Claude Opus 5 also increase the coverage of both models. More broadly, the computation of a frontier LLM can be spent once per domain and reused by a much smaller model across all instances of that domain.

AI Use Statement

The LLMs in our experiments, GPT-OSS-20B and Gemma-4-31B in the agent and GPT-5.6 Sol and Claude Opus 5 for the strategies, are part of what we study. Beyond them, we used Claude Code and Codex to implement the method and the experiments and to write the scripts that compute and plot the results, and generative AI tools to generate data sets, search the literature, draft sections of the paper, and edit the text. We did not use them to propose research ideas or hypotheses, develop theoretical models or conceptual frameworks, design or give feedback on the methodology and the experiments, clean or reformat data sets, translate text, analyze the results qualitatively, or interpret the results. The paper makes no mathematical claims, so formulating and proving such claims does

not apply to this work. We reviewed and revised all AI-assisted work and take full responsibility for the content of the paper.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *The Fourteenth International Conference on Learning Representations (ICLR 2026)*, 2026.

Anthropic. Claude Opus 5 System Card. `https://www.anthropic.com/claude-opus-5-system-card`, 2026.

Bernd Bohnet, Azade Nova, Aaron T Parisi, Kevin Swersky, Katayoon Goshvadi, Hanjun Dai, Dale Schuurmans, Noah Fiedel, and Hanie Sedghi. Exploring and benchmarking the planning capabilities of large language models. arXiv:2406.13094 [cs.CL], 2024.

Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1–2):5–33, 2001.

Andrey Borro, Patricia J Riddle, Michael W Barley, and Michael J Witbrock. Bridging the reasoning gap: Small LLMs can plan with generalised strategies. arXiv:2501.18817 [cs.AI], 2025.

Tom Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1–2):165–204, 1994.

Hyungjoo Chae, Yeonghyeon Kim, Seungone Kim, Kai Tzu-iunn Ong, Beong-woo Kwak, Moohyeon Kim, Sunghwan Kim, Taeyoon Kwon, Jiwan Chung, Youngjae Yu, and Jinyoung Yeo. Language models as compilers: Simulating pseudocode execution improves algorithmic reasoning in language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 22471–22502. Association for Computational Linguistics, 2024.

Minghao Chen, Yihang Li, Yanting Yang, Shiyu Yu, Binbin Lin, and Xiaofei He. AutoManual: Constructing instruction manuals by LLM agents via interactive environmental learning. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*, 2024.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. arXiv:2507.06261 [cs.CL], 2025.

Augusto B Corrêa, Guillem Francès, Markus Hecher, Davide Mario Longo, and Jendrik Seipp. Scorpion Maidu: Width search in the Scorpion planning system. In *Tenth International Planning Competition (IPC-10): Planner Abstracts*, 2023.

Augusto B Corrêa, André G Pereira, and Jendrik Seipp. Frontier large language models rival state-of-the-art planners. arXiv:2511.09378 [cs.AI], 2025.

Joseph C. Culberson. Sokoban is PSPACE-complete. Technical Report TR 97-02, Department of Computing Science, The University of Alberta, Edmonton, Alberta, Canada, 1997.

Kutluhan Erol, Dana S. Nau, and V. S. Subrahmanian. Complexity, decidability and undecidability results for domain-independent planning. *Artificial Intelligence*, 76(1–2):75–88, 1995.

Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. AutoGuide: Automated generation and selection of context-aware guidelines for large language model agents. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*, 2024.

Chang Gao, Haiyun Jiang, Deng Cai, Shuming Shi, and Wai Lam. StrategyLLM: Large language models as strategy generators, executors, optimizers, and evaluators for problem solving. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*, 2024.

Google DeepMind Gemma Team. Gemma 4 technical report. arXiv:2607.02770 [cs.CL], 2026.

Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004.

Google DeepMind. Gemini 3 Pro model card. https://deepmind.google/models/model-cards/gemini-3-pro/, 2025. Model card, accessed 2025-12-08.

Google DeepMind. Gemini 3.1 Pro model card. https://deepmind.google/models/model-cards/gemini-3-1-pro/, 2026.

Patrik Haslum, Nir Lipovetzky, Daniele Magazzeni, and Christian Muise. *An Introduction to the Planning Domain Definition Language*, volume 13 of *Synthesis Lectures on Artificial Intelligence and Machine Learning*. Morgan & Claypool, 2019.

Malte Helmert. Complexity results for standard benchmark domains in planning. *Artificial Intelligence*, 143(2):219–262, 2003.

Malte Helmert. New complexity results for classical planning benchmarks. In Derek Long, Stephen F. Smith, Daniel Borrajo, and Lee McCluskey (eds.), *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS 2006)*, pp. 52–61. AAAI Press, 2006a.

Malte Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26: 191–246, 2006b.

Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

Richard Howey and Derek Long. VAL's progress: The automatic validation tool for PDDL2.1 used in the International Planning Competition. In *Proceedings of the ICAPS Workshop on the Competition: Impact, Organization, Evaluation, Benchmarks*, 2003.

Michael Katz, Harsha Kokel, Christian Muise, Shirin Sohrabi, and Sarath Sreedharan. Make planning research rigorous again! *arXiv preprint arXiv:2505.21674*, 2025.

Drew McDermott. The 1998 AI Planning Systems competition. *AI Magazine*, 21(2):35–55, 2000.

OpenAI. Introducing GPT-4.1 in the API. https://openai.com/index/gpt-4-1/, 2025.

OpenAI. GPT-5 system card, 2025. URL https://openai.com/index/gpt-5-system-card/.

OpenAI. gpt-oss-120b & gpt-oss-20b model card. arXiv:2508.10925 [cs.CL], 2025.

OpenAI. GPT-5.6 system card. https://deploymentsafety.openai.com/gpt-5-6, 2026.

Borja Requena, Austin Letson, Krystian Nowakowski, Izan Beltran-Ferreiro, and Leopoldo Sarra. A minimal agent for automated theorem proving. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*, 2026.

Silvia Richter and Matthias Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39:127–177, 2010.

Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati. On the self-verification limitations of large language models on reasoning and planning tasks. In *International Conference on Learning Representations*, volume 2025, pp. 98190–98243, 2025.

Ayal Taitler, Ron Alford, Joan Espasa, Gregor Behnke, Daniel Fišer, Michael Gimelfarb, Florian Pommerening, Scott Sanner, Enrico Scala, Dominik Schreiber, Javier Segovia-Aguas, and Jendrik Seipp. The 2023 International Planning Competition. *AI Magazine*, 45(2):280–296, 2024. doi: 10.1002/aaai.12169.

Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. PlanBench: An extensible benchmark for evaluating large language models on planning and reasoning about change. *Advances in Neural Information Processing Systems*, 36:38975–38987, 2023a.

Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. On the planning abilities of large language models–a critical investigation. *Advances in Neural Information Processing Systems*, 36:75993–76005, 2023b.

Karthik Valmeekam, Kaya Stechly, Atharva Gundawar, and Subbarao Kambhampati. Planning in strawberry fields: Evaluating and improving the planning and scheduling capabilities of LRM o1. arXiv:2410.02162 [cs.AI], 2024.

Tianjun Zhang, Aman Madaan, Luyu Gao, Steven Zheng, Swaroop Mishra, Yiming Yang, Niket Tandon, and Uri Alon. In-context principle learning from mistakes. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 59520–59558. PMLR, 2024.

Xinran Zhao, Hanie Sedghi, Bernd Bohnet, Dale Schuurmans, and Azade Nova. Improving large language model planning with action sequence similarity. In *International Conference on Learning Representations*, 2025.

A PLANNING AND FEEDBACK PROMPTS

Listing 1: End-to-end planning prompt.

```
This is your task:
<task>
Given a solvable planning task, write a plan that transforms the initial state into a goal
     state.

The plan may be long. Do not stop after an intermediate phase and do not return an empty plan.

If you cannot construct the complete plan, do not give up. Return the longest grounded plan
     prefix you can construct, making sure that every returned action is applicable and makes
     the state closer to the goal.
</task>

Planning terminology, defined here for reference:
<planning-concepts>
1. A planning task is defined by a PDDL domain together with a PDDL instance.
2. A PDDL domain defines predicates, types, constants, and action schemas. An action schema
     lists parameters, preconditions, and effects.
3. A PDDL instance defines objects, the initial state, and the goal condition.
4. A fact is a predicate with each argument bound to an object or constant whose type matches
     the argument's type.
5. A state is the set of facts that hold at one point in time.
6. A goal state is a state in which every fact of the goal condition holds.
7. A grounded action is an action schema with each parameter bound to an object or constant
     whose type matches the parameter's type.
8. A grounded action is applicable in a state when its precondition holds in that state.
9. Applying a grounded action to a state produces the next state: remove its delete-effects,
     add its add-effects, leave every other fact unchanged.
10. A plan is a sequence of grounded actions. Starting from the initial state and applying the
      plan in order must reach a goal state, and each action must be applicable in the state
     where it is used.
</planning-concepts>

This is how the plan must be formatted:
<output-format>
1. Return the plan wrapped in <plan>...</plan>.
2. Each non-empty line inside the tags is one grounded action written as (action_name object1
     object2 ...).
3. Do not write anything before <plan> or after </plan>. Do not use markdown or code fences.
</output-format>

The context you need to solve this task:
<input>
PDDL domain:
<pddl-domain>$domain_file</pddl-domain>

PDDL instance:
<pddl-instance>$instance_file</pddl-instance>
</input>

Before returning the plan, make sure that:
<checklist>
1. The plan starts from the initial state.
2. The precondition of each action holds in the state where the action is applied.
3. The state after the last action satisfies the goal condition.
4. Each action uses an action name from the domain, and each parameter is bound to an object
     or constant whose type matches.
5. The plan is wrapped in <plan>...</plan> and contains nothing else.
6. If you cannot satisfy every goal fact, still return the longest grounded plan prefix you
     can construct that makes the state closer to the goal; never fall back to an empty plan.
</checklist>
```

Listing 2: Strategy-guided planning prompt.

```
This is your task:
<task>
Given a solvable planning task and a natural-language strategy, write a plan that transforms
     the initial state into a goal state.

Follow the strategy carefully and maintain the current state after every action. Whenever the
     strategy uses a role or placeholder, bind it to an actual object or constant from the
     PDDL instance. Never copy a role or placeholder literally into the plan.

The plan may be long. That is expected: keep applying the strategy until every goal fact is
     satisfied. Do not stop after an intermediate phase and do not return an empty plan.
```

```
If you cannot construct the complete plan, do not give up. Return the longest grounded plan
     prefix you can construct by following the strategy, making sure that every returned
     action is applicable and makes the state closer to the goal.
</task>

Planning terminology, defined here for reference:
<planning-concepts>
1. A planning task is defined by a PDDL domain together with a PDDL instance.
2. A PDDL domain defines predicates, types, constants, and action schemas. An action schema
     lists parameters, preconditions, and effects.
3. A PDDL instance defines objects, the initial state, and the goal condition.
4. A fact is a predicate with each argument bound to an object or constant whose type matches
     the argument's type.
5. A state is the set of facts that hold at one point in time.
6. A goal state is a state in which every fact of the goal condition holds.
7. A grounded action is an action schema with each parameter bound to an object or constant
     whose type matches the parameter's type.
8. A grounded action is applicable in a state when its precondition holds in that state.
9. Applying a grounded action to a state produces the next state: remove its delete-effects,
     add its add-effects, leave every other fact unchanged.
10. A plan is a sequence of grounded actions. Starting from the initial state and applying the
      plan in order must reach a goal state, and each action must be applicable in the state
     where it is used.
</planning-concepts>

This is how the plan must be formatted:
<output-format>
1. Return the plan wrapped in <plan>...</plan>.
2. Each non-empty line inside the tags is one grounded action written as (action_name object1
     object2 ...).
3. Do not write anything before <plan> or after </plan>. Do not use markdown or code fences.
</output-format>

The context you need to solve this task:
<input>
PDDL domain:
<pddl-domain>$domain_file</pddl-domain>

PDDL instance:
<pddl-instance>$instance_file</pddl-instance>

The strategy to follow:
<strategy>$strategy</strategy>
</input>

Before returning the plan, make sure that:
<checklist>
1. The plan starts from the initial state.
2. The precondition of each action holds in the state where the action is applied.
3. The state after the last action satisfies the goal condition.
4. Each action uses an action name from the domain, and each parameter is bound to an actual
     object or constant from the PDDL instance whose type matches. Never output a role or
     placeholder from the strategy literally.
5. The plan is wrapped in <plan>...</plan> and contains nothing else.
6. If you cannot satisfy every goal fact, still return the longest grounded plan prefix you
     can construct that makes the state closer to the goal; never fall back to an empty plan.
</checklist>
```

Listing 3: Feedback prompt.

```
This is your task:
<task>
$seed_task

You are also given the immediately preceding call, which returned an invalid plan. Use the
     feedback to find where that plan went wrong, and write a new plan that avoids those
     mistakes while still satisfying the task above.

The prompt includes:
1. The reasoning trace from the immediately preceding call;
2. The invalid plan returned by that call;
3. VAL's feedback on that plan.
</task>

Planning terminology, defined here for reference:
<planning-concepts>
1. A planning task is defined by a PDDL domain together with a PDDL instance.
2. A PDDL domain defines predicates, types, constants, and action schemas. An action schema
     lists parameters, preconditions, and effects.
3. A PDDL instance defines objects, the initial state, and the goal condition.
```

```
4. A fact is a predicate with each argument bound to an object or constant whose type matches
     the argument's type.
5. A state is the set of facts that hold at one point in time.
6. A goal state is a state in which every fact of the goal condition holds.
7. A grounded action is an action schema with each parameter bound to an object or constant
     whose type matches the parameter's type.
8. A grounded action is applicable in a state when its precondition holds in that state.
9. Applying a grounded action to a state produces the next state: remove its delete-effects,
     add its add-effects, leave every other fact unchanged.
10. A plan is a sequence of grounded actions. Starting from the initial state and applying the
      plan in order must reach a goal state, and each action must be applicable in the state
     where it is used.
</planning-concepts>

This is how the plan must be formatted:
<output-format>
1. Return the plan wrapped in <plan>...</plan>.
2. Each non-empty line inside the tags is one grounded action written as (action_name object1
     object2 ...).
3. Do not write anything before <plan> or after </plan>. Do not use markdown or code fences.
</output-format>

The context you need to solve this task:
<input>
$seed_input

Reasoning trace from the immediately preceding call:
<previous-reasoning>$previous_reasoning_trace</previous-reasoning>

Plan returned by the immediately preceding call:
<returned-answer>$previous_plan_text</returned-answer>

Feedback on this plan:
<val-feedback>$val_diagnostic_report</val-feedback>
</input>

Before returning the plan, make sure that:
<checklist>
1. The plan starts from the initial state.
2. The precondition of each action holds in the state where the action is applied.
3. The state after the last action satisfies the goal condition.
4. Each action uses an action name from the domain, and each parameter is bound to an object
     or constant whose type matches.
5. The plan is wrapped in <plan>...</plan> and contains nothing else.
6. If you cannot satisfy every goal fact, still return the longest grounded plan prefix you
     can construct that makes the state closer to the goal; never fall back to an empty plan.
7. The plan fixes the mistakes reported by VAL for the immediately preceding call.
</checklist>
```

B STRATEGY GENERATION PROMPTS

Listing 4: Strategy generation prompt.

```
This is your task:
<task>
Write a code strategy in Python for the $domain_name PDDL domain.

This prompt contains the $domain_name PDDL domain and 3 representative training instances. Use
      them to understand the structure of the instances that the strategy must solve.

It also contains Gripper and Logistics examples, each with a PDDL domain, one example instance,
      and a code strategy.

Implement this function:

def solve(domain: str, instance: str) -> list[str]:
    ...

The inputs are:
1. domain: the PDDL domain as a string.
2. instance: the PDDL instance as a string.

The function must return a valid plan as a list of grounded PDDL action strings in execution
     order. Each string must have this format:
"(action_1 object_1 ... object_N)"

The plan must be applicable from the initial state and must satisfy the goal.
```

```
Implement a deterministic $domain_name strategy. Use the domain invariants and action ordering
     to construct the plan.

Requirements:
1. Return only Python source code. Do not use markdown fences or explanations.
2. Use only the Python standard library.
3. Do not call external planners, subprocesses, VAL, files outside the inputs, or network APIs.

4. Do not hardcode object names, object counts, or layouts. The strategy must solve any
    solvable instance of the $domain_name domain.
5. The code must be syntactically valid when saved verbatim as a .py file.
6. Use only action names that appear in the PDDL domain. Do not invent new action names.
7. If the initial state does not already satisfy the goal and the strategy cannot construct a
    plan, raise an exception instead of returning an empty list.
8. Keep the strategy clear and compact.
</task>

Here is an example for the Gripper domain:
<example>
Here is the Gripper domain:
<example-pddl-domain>$gripper_domain_file</example-pddl-domain>

Here is one example instance of the Gripper domain:
<example-pddl-instance>$gripper_instance_file</example-pddl-instance>

Here is the Gripper code strategy:
<example-strategy>$gripper_strategy_code</example-strategy>
</example>

Here is an example for the Logistics domain:
<example>
Here is the Logistics domain:
<example-pddl-domain>$logistics_domain_file</example-pddl-domain>

Here is one example instance of the Logistics domain:
<example-pddl-instance>$logistics_instance_file</example-pddl-instance>

Here is the Logistics code strategy:
<example-strategy>$logistics_strategy_code</example-strategy>
</example>

Now write the $domain_name code strategy.

Here is the $domain_name domain:
<pddl-domain>$domain_file</pddl-domain>

Here are 3 representative training instances for the $domain_name domain. Use them to
    understand the structure of the domain's instances. They are examples, not special cases:
     do not hardcode their particular object names or sizes.

Training example 1:
<example-pddl-instance>$example_instance_1</example-pddl-instance>

Training example 2:
<example-pddl-instance>$example_instance_2</example-pddl-instance>

Training example 3:
<example-pddl-instance>$example_instance_3</example-pddl-instance>

Return only the Python source code for the code strategy.
```

Listing 5: Translation prompt.

```
Translate this code strategy into a natural-language strategy so I can send it to an LLM.

$python_strategy_code
```

C DOMAINS

For each domain, we describe its objects, actions, and goal. In each description, n is the parameter of Table 1.

C.1 IPC DOMAINS

C.1.1 BLOCKSWORLD

A robot arm rearranges n blocks on a table. Initially the blocks form a set of towers, and the goal is another set of towers. The arm holds one block at a time: it picks up a clear block from the table or unstacks it from the block below, and puts it down on the table or stacks it on a clear block.

C.1.2 CHILDSNACK

The goal is to serve a sandwich to each of n children who wait at tables, and a child allergic to gluten must get a gluten-free one. A sandwich is made in the kitchen from a bread and a content portion, and is gluten-free when both are. It is put on a tray in the kitchen, and the tray is moved to the table of the child, who is served from it. There is one bread and one content portion per child, and the gluten-free portions are exactly enough for the allergic children, so one spent on a child who is not allergic leaves an allergic child without a sandwich.

C.1.3 FLOORTILE

Robots paint a grid of n tiles in two colors, and the goal is a checkerboard on every row except the bottom one. A robot moves up, down, left, or right onto a clear tile, that is, a tile neither painted nor occupied by a robot. It paints only the tile directly above or below it, in the color it holds, and can switch to the other color. Since a painted tile cannot be entered, painting in the wrong order can cut a robot off from the tiles it still has to paint.

C.1.4 MICONIC

A lift carries n passengers in a building, each from an origin floor to a destination floor. The lift moves between any two floors in one action and has no capacity limit. A passenger boards when the lift is at the origin floor and leaves at the destination floor, and the goal is to deliver every passenger. The passengers can be served in any order.

C.1.5 ROVERS

A team of n rovers explores the waypoints of a planet, and each rover can drive only along its own set of paths. The rovers collect data, analyses of soil and rock samples and images of objectives in given modes, and the goal is to send given data to a lander. A rover analyzes a sample only if it is equipped for it and its single store is empty, takes an image only with a camera that supports the mode and was calibrated since its last image, and sends data only from a waypoint from which the lander is visible.

C.1.6 SOKOBAN

A robot pushes n boxes in a maze of cells, and the goal assigns each box a cell. The robot moves to a neighboring cell without a box, and pushes a box by moving into its cell, which moves the box one cell further in the same direction if that cell is free. Boxes cannot be pulled, so a box pushed into a corner can never move again, and one push can make the goal unreachable.

C.1.7 SPANNER

A man walks along a one-way corridor from a shed to a gate, where loose nuts must be tightened, and n spanners lie along the corridor. The man picks up spanners and can carry any number of them, and each spanner tightens one nut and then becomes unusable. The man can neither walk back nor drop a spanner, so a spanner left behind is lost, and the man must reach the gate with at least one spanner per nut.

C.1.8 TRANSPORT

A fleet of n vehicles delivers packages over a network of two-way roads. Each vehicle carries a limited number of packages at once, and picks up and drops packages at its location. Initially every

vehicle is empty, and the goal places each package at its destination. Since the roads connect all locations, any vehicle can deliver any package on its own.

C.2 Novel Domains

C.2.1 Cisterns

A set of n tanks is connected by hoses, and each tank holds water at one of a fixed set of levels. The only action siphons one level of water through a hose from a tank into a tank that holds at least two levels less. The total water is thus constant, levels only even out, and no siphon can be undone. The goal fixes the levels of some of the tanks.

C.2.2 Orrery

An orrery has n wheels meshed in pairs, and each wheel shows one of k marks arranged in a cycle. The only action twists two meshed wheels, turning one forward by one mark and the other backward by one mark, so no wheel turns alone. The goal is to bring every wheel to its reference mark.

C.2.3 Qubit-Routing

A quantum computer holds n qubits on the nodes of a grid, one qubit per node, and must execute a program, a sequence of gates that each operate on two qubits, in order. A gate can be executed only when its two qubits are on adjacent nodes, and a swap exchanges the qubits of two adjacent nodes. The goal is to execute every gate.