WORLD-ADAPT: A BENCHMARK FOR AGENT ADAPTATION TO INACCURATE WORLD MODELS

Caleb Kha Uong* **Brian Wang*** **Lucas Hathaway*** **Karthik Narasimhan**
Princeton University
*Equal contribution.
https://github.com/zh1kang/world-adapt

ABSTRACT

World models let agents preview the outcome of an action before taking it, but a predictor that is wrong in a consistent way can mislead the plans built on it. We ask whether language-model agents can learn, from their own experience, where such a predictor fails and plan around it on new tasks. To test this, we introduce World-Adapt, a benchmark of six planning environments. In each, the agent can execute actions or query a predictor that is either accurate or faulty, with fixed, hidden errors. After every task it records its experience in a notebook (its only memory between tasks) that is either saved or reset, and held-out tasks test whether what it learned transfers. Across nine models, the faulty predictor lowers held-out success by about four points. Strong agents identify its error, often after one task. In two environments, agents whose notebook holds this diagnosis execute through 67% of the predictor's false alarms, against 23% without it. This does not raise their success, and weaker agents tend to record the predictor's errors as facts. We release the benchmark, its tasks and our results.

1 INTRODUCTION

Recently, world models have been used to allow agents to anticipate the consequences of actions before committing to them. An agent can propose actions, query the world model for a likely outcome, and use the resulting output to choose what to execute. However, access to such foresight does not guarantee accurate planning. Recent studies have shown that access to world model predictions can produce inconsistent or degraded agent performance (Qian et al., 2026). Learned predictors can also be wrong in a consistent way, for example by predicting that every push falls short. Thus, an agent using these tools must account for discrepancies between predicted and observed outcomes. This raises a practical question: can experience with a predictor's perturbed outputs improve decisions on new tasks while the separate predictor remains inaccurate?

In-context learning suggests that a sufficiently sophisticated agent can change its behaviors from experience without updating its parameters (Brown et al., 2020). In our setting, the relevant experience comes from interaction with the task environment rather than supplied demonstrations. The agent observes differences between predicted and executed outcomes and can record useful information in a notebook for later tasks. The capability we aim to measure is whether the agent can use these experiences to adjust its reliance on the predictor and transfer that adjustment to new task instances. This matters in cases where retraining the agent or correcting the predictor is difficult or costly. Reliable behavior must then depend on what the agent can learn from the context it carries forward.

Existing research investigates several aspects of this problem. Qian et al. (2026) examine whether agents can use world models as foresight tools, including when they request predictions and how they incorporate them into decisions. Their interactive tasks use exact simulators, allowing them to study failures when agents use accurate forecasts. World-in-World instead compares world models through closed-loop task success under a shared planning strategy (Zhang et al., 2026a). WorldTest, instantiated in AutumnBench, asks whether agents learn environment dynamics through interaction and can apply that knowledge to planning and other queries (Warrier et al., 2026). These evaluations address the use, utility, and acquisition of predictive knowledge, but do not directly test adaptation to a controlled, persistent error in a supplied predictor.

Other work more directly addresses inaccurate predictions and learning from experience. Dual-Frontier studies when an agent should trust a world-model-guided decision, using verification and bounds on decision-relevant prediction error (Zhu et al., 2026). WorldEvolver uses observed transitions and prediction–observation mismatches to update a world model's memory, improving its forecasts without changing model parameters (Zhang et al., 2026b). Reflexion similarly shows that agents can improve through textual feedback retained across trials (Shinn et al., 2023). These works motivate our question, but improving a predictor or learning from task feedback differs from learning to plan with a predictor whose errors remain unchanged.

In this paper, we introduce World-Adapt, a controlled benchmark for this question. Across six planning environments, an agent interacts with the true environment and queries a simulator containing a fixed, hidden perturbation. Both the agent's parameters and the predictor remain frozen, and the agent adapts through a notebook saved between tasks. The benchmark crosses an accurate or faulty predictor with a saved or reset notebook. A saved notebook can also help by keeping general task knowledge, so we measure whether it adds more with the faulty predictor than with the accurate one on held-out tasks. World-Adapt therefore targets a specific component of reliable agency: transferring experience with an imperfect predictive tool to new decisions under fixed interaction budgets.

2 World-Adapt: Planning with Inaccurate World Models

2.1 Problem Formulation

We formulate each task as a partially observable Markov decision process (POMDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, \mathcal{O}, O, \rho_0, R, H)$, where $\mathcal{S}$ and $\mathcal{A}$ denote the state and action spaces, $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ the transition function, $\mathcal{O}$ the observation space, $O : \mathcal{S} \to \mathcal{O}$ the observation function, ρ_0 the initial-state distribution, R a terminal reward, and H the task horizon. Additionally, we introduce a *predictor model POMDP* $\mathcal{M}^* = (\mathcal{S}, \mathcal{A}, T^*, \mathcal{O}, O, \rho_0, R, H)$, such that it is identical to M apart from T^*. The predictor is *accurate* when $T^* = T$ and *faulty* when we apply fixed, deterministic modifications to T to get T^*.

Because the environment hides information, the agent often cannot plan a full solution from its first observation, and M^* lets it explore before acting. During a task, the agent executes actions in M and may query candidate action sequences in M^* from its current state, which returns the predicted observations. Separate budgets limit queries and actions within the horizon H.

2.2 Benchmark Construction

From a set of candidates, we kept six that showed low success under a vanilla harness but large gains with an accurate model of the task. Random policies succeed in them rarely, mostly in under 0.5% of $10{,}000$ attempts.

In the widely used **Push-T** environment (Chi et al., 2023), the agent must control a circular end-effector to manipulate a T-shaped block until it covers 95% of a target pose. In the classic **Depot** logistics puzzle (Long & Fox, 2003), the agent drives trucks and uses hoists to lift, drop, load, and unload crates until the crates form the goal stacks; we use the domain's PDDL definition (McDermott et al., 1998) from PDDLGym (Silver & Chitnis, 2020) and generate our own tasks. The other four are new.

Switchboard. The agent is given a set of buttons, and the goal is to turn on all the lights. Each button flips some lights on or off, and may also trip a breaker that permanently breaks another button.

Patrol. A grid environment in which there are guards that patrol predetermined but hidden routes. The agent must navigate to the goal while avoiding being caught, which happens if it lands on the same square as the guard after a step or they exchange positions during a step.

PortalMaze. A grid environment with locked doors where the agent can reach the goal using keys and portals. Each key is paired with a specific door and the destination of each portal is hidden.

IceSlide. A Sokoban-inspired game where the agent must push crates into designated slots in the grid environment. However, there are both icy and rough cells in the grid, with icy cells causing the agent or crate to slide until it is stopped by a rough cell or another object.

2.3 Agent Interface and Memory

Our harness gives the agent two tools: `execute()` runs an action sequence in the real environment, and `predict()` runs one in the predictor. Both return the resulting observations. The agent has separate budgets of real and predicted environment steps: it may execute three times as many actions as a reference solution needs (at least 10, and 40 in Push-T) and simulate twice that many, with at most eight actions per call.

During a single task, the harness maintains the conversation history, and at the end of the task the agent is allowed to record its experience in a notebook. We call one model's 14 tasks in one environment under one condition a *task sequence*: eight adaptation tasks, then six held-out tasks that each start from the notebook written after task 8 (Figure 2). Between tasks the notebook is saved or reset to empty. The four conditions cross this choice with an accurate or faulty predictor.

3 Experiments and Results

Protocol. Appendix A lists the faulty predictor's errors in each environment. Tasks come in easy, medium and hard tiers. Notebooks hold at most 2,000 tokens, and agents are told only that the predictor may be imperfect (Appendix E). One seed fixes the tasks, and we compare conditions on the same instances. Nine models act in a loop of reasoning and tool calls (Yao et al., 2023), each over 24 task sequences (336 tasks): Claude Opus 5.5 at high reasoning effort, and DeepSeek V4.1 Flash, Kimi K2.6, GLM-5.3-Flash, Gemma 4 31B, Qwen3.6-35B-A3B, Nemotron 3 Super, GPT-OSS-120B and Inkling at low effort.

Metrics. Our main metric is held-out success. The *notebook value* is success with a saved notebook minus success with a reset one, under each predictor. If experience with a faulty predictor helps, it should be larger there. Our 95% intervals come from a bootstrap over (model, environment) cells (Efron & Tibshirani, 1993).

Notebook Judge. Three LLM judges (Zheng et al., 2023) (Claude Opus 5.5, DeepSeek V4.1 Flash, Kimi K3) label the 1,674 nonempty notebooks written after adaptation tasks against the true environment rules and the implemented perturbation, without seeing the model or any outcome (Appendix C). We keep the majority label. The judges agree on 89% of diagnoses (Fleiss' $\kappa = 0.81$; Fleiss 1971) and on 83% of policy labels ($\kappa = 0.66$).

Held-out success runs from 97% for Claude Opus 5.5 to 0% for Inkling (Table 1). The faulty predictor lowers held-out success by 4.0 points $[-7.5, -0.5]$, more during adaptation (Appendix B).

A saved notebook helps with the accurate predictor, but we detect no gain with the faulty one. On held-out tasks, a saved notebook adds $+6.6$ points $[+2.8, +10.8]$ with the accurate predictor and $+0.0$ points $[-4.5, +4.9]$ with the faulty one. The difference of -6.6 points $[-12.5, -0.7]$ is borderline (permutation $p = 0.05$). Appendix B splits these contrasts by phase and tier.

Within models (Figure 1), the notebook value is positive for all but Nemotron with the accurate predictor. With the faulty one, only GLM's interval lies above zero. The notebook also changes predictor use. With the faulty predictor, Qwen, which trusts the predictor, raises its `predict` calls from 2.2 to 9.1 per task, while Opus and Gemma, which name the error, lower them slightly.

Diagnosing errors changes agent behavior, but not whether they succeed. In nearly every faulty-predictor task sequence, Opus, DeepSeek, Kimi and Gemma name or locate the perturbed rule (Table 6). In Patrol and Depot, where the faulty predictor only raises false alarms, agents whose notebook diagnosed the error execute through 67% of visibly wrong predictions, against 23% without a diagnosis ($+45$ points $[+25, +63]$; Appendix B). Because stronger models diagnose more often, part of this gap could reflect model strength. The pattern also holds within models. For six of the seven models with both diagnosed and undiagnosed notebooks, the rate of executing through a false alarm is higher after a diagnosis. The habit carries into held-out tasks, but a saved notebook does not raise success (-3.1 points $[-8.3, +2.1]$). Weaker models invent errors: Nemotron claims a non-existent predictor error in 10 of its 12 accurate-predictor task sequences, and Qwen's final notebook trusts the faulty predictor in five of six.

Table 1: Held-out success (%, higher is better) over 36 tasks per condition, 24 per environment and 144 per model. S, R: saved, reset notebook. Bold: best.

	Predictor + notebook				Environment						
Model	Acc.+S	Acc.+R	Faulty+S	Faulty+R	Depot	Portal	Patrol	Switch	Ice	Push-T	All
Claude Opus 5.5	**100**	**94**	**97**	**97**	**100**	**96**	**100**	88	**100**	**100**	**97**
DeepSeek V4.1 Flash	94	89	83	83	**100**	**96**	92	**92**	79	67	88
Kimi K2.6	72	67	67	67	**100**	92	46	88	71	12	68
GLM-5.3-Flash	72	56	61	47	**100**	88	50	21	71	25	59
Gemma 4 31B	44	42	39	39	92	42	0	62	50	0	41
Qwen3.6-35B-A3B	33	17	17	22	17	29	12	21	54	0	22
Nemotron 3 Super	11	14	6	14	8	8	4	12	33	0	11
GPT-OSS-120B	8	6	8	8	8	4	0	8	25	0	8
Inkling	0	0	0	0	0	0	0	0	0	0	0

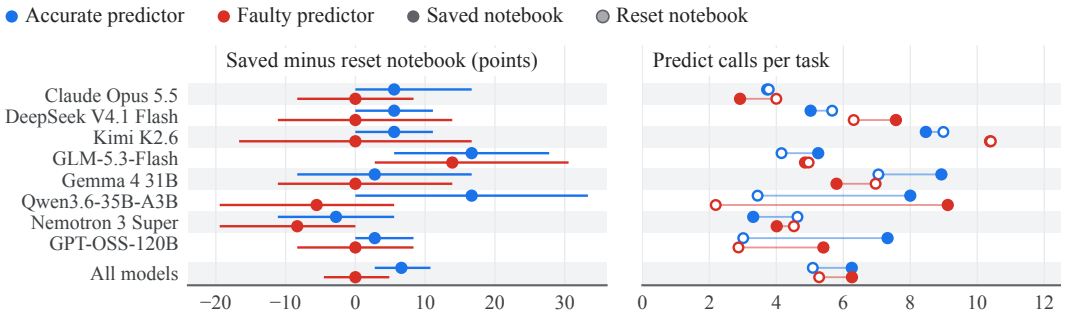


Figure 1: A saved notebook helps with the accurate predictor but not the faulty one. Left: held-out success with a saved minus a reset notebook, with 95% bootstrap intervals. Right: mean `predict` calls per task over all 14 tasks. Inkling is omitted.

The models thus separate on three abilities. Strong models find the error, weaker ones also write false lessons, but no model turns a diagnosis into higher success. Qwen shows the cost of a bad record. With the faulty predictor, its saved notebook lowers success from 22% to 17%. Push-T is the hardest environment: its predictor shortens every push, and five models solve none of its held-out tasks in any condition.

4 What Did Not Work

A saved notebook did not recover the faulty predictor's cost, and over the eight adaptation tasks it did not pull ahead of a reset one (Appendix B). Our perturbations are too mild for Opus, which loses nothing to them. The budgets may also be too generous, because even with the notebook reset, models identify the faulty predictor's error within a single task in 19–43% of tasks. Tighter budgets or late-appearing errors may help. Two controls are missing: a run without the predictor and a notebook that states the true error. Without them we cannot tell whether agents fail to use a diagnosis or whether these tasks reward it little.

The weakest models fail for reasons other than adaptation. Nemotron gives up on 47% of its failed tasks and Inkling on 46%, and GPT-OSS ends 26% at the limit on malformed tool calls.

5 Conclusion

We introduced World-Adapt, a benchmark that tests whether agents can learn to plan with a predictor whose errors never change. Strong agents partly can: they find the error and execute through 67% of its false alarms when their notebook holds the diagnosis, against 23% without. These better

decisions do not yet raise success, and weaker agents learn the wrong lessons. We release the benchmark, tasks and results to help build agents that turn this knowledge into better plans.

ACKNOWLEDGMENTS

We thank Seth Karten for valuable feedback. We are grateful to Modal, SAIL Research and Prime Intellect for providing the compute credits that made it possible to run the models in this study.

REFERENCES

Yusra Alkhazraji, Matthias Frorath, Markus Grützner, Malte Helmert, Thomas Liebetraut, Robert Mattmüller, Manuela Ortlieb, Jendrik Seipp, Tobias Springenberg, Philip Stahl, and Jan Wülfing. Pyperplan, 2020. URL https://doi.org/10.5281/zenodo.3700819.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901, 2020. URL https://arxiv.org/abs/2005.14165.

Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *Proceedings of Robotics: Science and Systems (RSS)*, 2023.

Bradley Efron and Robert J. Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall, 1993.

Joseph L. Fleiss. Measuring nominal scale agreement among many raters. *Psychological Bulletin*, 76(5):378–382, 1971. doi: 10.1037/h0031619.

Derek Long and Maria Fox. The 3rd international planning competition: Results and analysis. *Journal of Artificial Intelligence Research*, 20:1–59, 2003. doi: 10.1613/jair.1240.

Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL—the planning domain definition language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

Cheng Qian, Emre Can Acikgoz, Bingxuan Li, Xiusi Chen, Yuji Zhang, Bingxiang He, Qinyu Luo, Gokhan Tur, Dilek Hakkani-Tür, Yunzhu Li, and Heng Ji. Current agents fail to leverage world model as tool for foresight. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13686–13723, 2026. doi: 10.18653/v1/2026.acl-long.623. URL https://aclanthology.org/2026.acl-long.623/.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://papers.nips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Tom Silver and Rohan Chitnis. PDDLGym: Gym environments from PDDL problems. *arXiv preprint arXiv:2002.06432*, 2020. URL https://arxiv.org/abs/2002.06432.

Archana Warrier, Dat Nguyen, Michelangelo Naim, Moksh Jain, Yichao Liang, Karen Schroeder, Cambridge Yang, Joshua B. Tenenbaum, Sebastian Josef Vollmer, Kevin Ellis, and Zenna Tavares. Benchmarking world-model learning with environment-level queries. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*, pp. 133215–133246. PMLR, 2026. URL https://proceedings.mlr.press/v306/warrier26a.html.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.03629.

Jiahan Zhang, Muqing Jiang, Nanru Dai, Taiming Lu, Arda Uzunoglu, Shunchi Zhang, Yana Wei, Jiahao Wang, Vishal M. Patel, Paul Pu Liang, Daniel Khashabi, Cheng Peng, Rama Chellappa, Tianmin Shu, Alan Yuille, Yilun Du, and Jieneng Chen. World-in-world: World models in a closed-loop world. In *International Conference on Learning Representations*, 2026a. URL https://arxiv.org/abs/2510.18135.

Xuan Zhang, Wenxuan Zhang, See-Kiong Ng, and Yang Deng. Self-evolving world models for LLM agent planning. *arXiv preprint arXiv:2606.30639*, 2026b. URL https://arxiv.org/abs/2606.30639.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://arxiv.org/abs/2306.05685.

Huatai Zhu, Qiang Chen, Ziqian Kou, Wenhao Li, Fei Wang, Yichao Cao, Xiu Su, and Yi Chen. Dual-frontier: When can an agent trust its world model? *arXiv preprint arXiv:2609.26293*, 2026. URL https://arxiv.org/abs/2609.26293.

A Environments

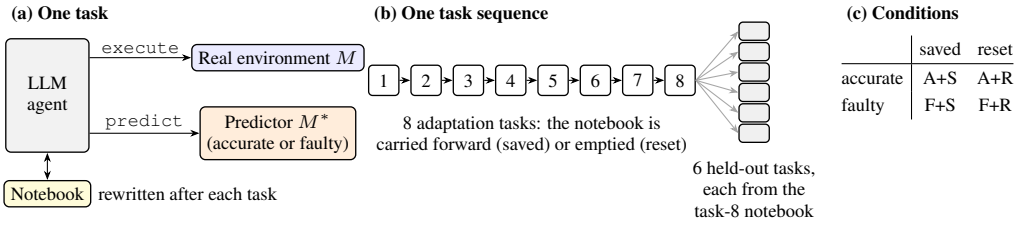


Figure 2: The protocol. (a) In each task the agent acts in the real environment and may query a predictor that is accurate or faulty. After the task it rewrites its notebook. (b) A task sequence runs eight adaptation tasks and then six held-out tasks, each starting from the notebook written after task 8. (c) Each model runs every environment under four conditions.

Depot. The domain file comes from PDDLGym (Silver & Chitnis, 2020), and we execute it with Pyperplan's grounded STRIPS operators (Alkhazraji et al., 2020). An action whose preconditions do not hold leaves the state unchanged. Our generator builds stack-and-transport tasks, removes structural duplicates, and keeps only tasks whose reference plan it verifies.

Difficulty tiers. Each generator sets a task's tier by its size and the length of its shortest solution. The predictor's errors stay the same at every tier, and step budgets grow with the reference solution. Harder Depot tasks have more places and crates, from 2 places and 5 crates up to 4 and 11. Harder PortalMaze tasks have larger grids with more pads and keys, and harder Switchboard tasks have more lamps and breakers. Harder IceSlide tasks have larger grids with up to three crates, harder Patrol tasks need slightly longer solutions, and the Push-T tiers move from straight pushes to rotations. Every task sequence plays its eight adaptation tasks in the order medium, easy, hard, easy, medium, hard, medium, easy, and its six held-out tasks are two of each tier. Over all tasks of the eight models that solve any held-out task, success falls from 59% on easy tasks to 48% on medium and 42% on hard ones. The drop is largest in IceSlide, from 88% to 28%, and small or absent in Depot, Patrol and Push-T.

Perturbations. The faulty predictor applies the true transition and then a fixed set of rules, each a condition and the wrong effect the predictor reports when it holds (Table 2). Because each predicted step starts from the previous predicted state, errors compound over a long query. The real environment is never perturbed.

Table 2: The faulty predictor's rules in each environment.

Environment	What the faulty predictor gets wrong
Depot	`load` is blocked when the truck already holds a crate, and `drop` is blocked when the support is a crate. Legal actions appear to do nothing.
PortalMaze	Stepping on a pad sends the agent to a pad that links into it instead of to its destination. Walking into a locked door while holding another key opens it.
Patrol	Ending a step alive and orthogonally next to a guard is reported as a capture. Safe moves appear fatal.
Switchboard	Pressing an enabled breaker while a button is disabled appears to disable the lowest enabled button other than the pressed button and the true target.
IceSlide	A crate sliding two or more cells stops one cell short. A crate that stops on a rough tile slides on. A crate launched vertically into a socket rests loose.
Push-T	The pusher moves 80% of the way to each requested target. Every predicted push falls short.

B ADDITIONAL RESULTS

The faulty predictor changes success by -6.0 points during adaptation and -4.0 on held-out tasks, and the notebook's value appears on easy and medium held-out tasks but not on hard ones (Table 3). During adaptation a saved notebook adds $+2.1$ points with the accurate predictor. Success over the adaptation tasks follows their fixed difficulty order, and saved notebooks do not pull ahead of reset ones (Figure 3). For each model, Table 4 shows how failed tasks end and how many tokens a task uses.

Table 3: Paired contrasts in percentage points by phase and held-out tier, with 95% bootstrap intervals over (model, environment) cells. Pairs counts the tasks scored in all four conditions. Inkling is omitted.

Tasks	Pairs	Faulty − accurate	Saved − reset (acc.)	Saved − reset (faulty)
Adaptation tasks	384	-6.0 [-8.9, -3.4]	$+2.1$ [-1.6, +6.0]	$+1.0$ [-3.4, +6.2]
Held-out, easy	96	-4.7 [-10.9, +1.6]	$+10.4$ [+3.1, +18.8]	-1.0 [-8.3, +6.2]
Held-out, medium	96	-3.6 [-9.4, +1.6]	$+10.4$ [+3.1, +17.7]	-3.1 [-11.5, +5.2]
Held-out, hard	96	-3.6 [-8.3, +1.0]	-1.0 [-6.2, +4.2]	$+4.2$ [-3.1, +11.5]
Held-out, all	288	-4.0 [-7.5, -0.5]	$+6.6$ [+2.8, +10.8]	$+0.0$ [-4.5, +4.9]
All 14 tasks	672	-5.1 [-7.4, -2.8]	$+4.0$ [+1.2, +7.1]	$+0.6$ [-3.0, +4.8]

Acting on a wrong prediction. For each prediction that the faulty rule visibly changed (4,304 in all), we check whether the agent later executed the queried actions through the first wrong step. This is possible because the evaluator log links each executed action to the predicted step it matches. In Patrol and Depot every such change is a false alarm, a capture or a blocked action that does not happen, and executing through it is the adaptive response. In the other environments the rule can also make a bad move look good, and the rate has no single reading.

Across the 628 false alarms in Patrol and Depot, a saved notebook raises the rate from 30% to 42% ($+12$ points $[-3, +31]$, bootstrap over task sequences). A diagnosis in the notebook the task started from raises it from 23% to 67% ($+45$ points $[+25, +63]$). On held-out tasks alone, this comparison is weak: only Gemma has both diagnosed and undiagnosed held-out notebooks there.

The gain holds on adaptation tasks, where a saved notebook gives 44% against 30% for a reset one, and on held-out tasks, with 41% against 30% (Table 5). Held-out success in these two environments still does not rise with a saved notebook (-3.1 points $[-8.3, +2.1]$).

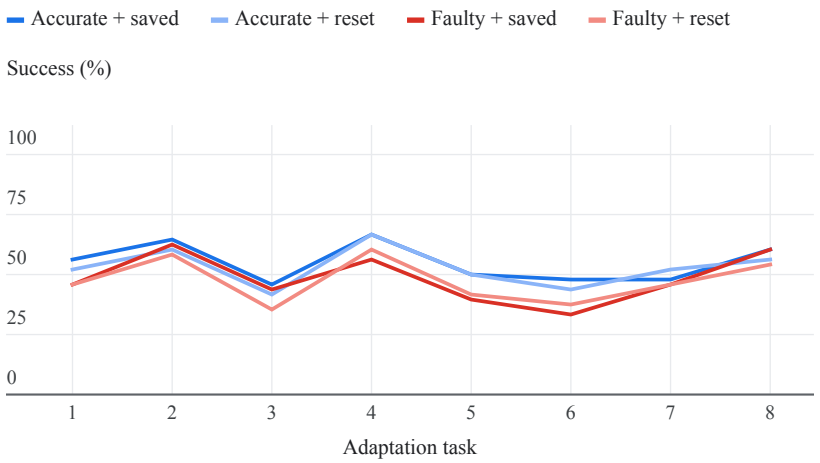


Figure 3: Success over the eight adaptation tasks follows their difficulty, which is the same at each position of every task sequence: medium, easy, hard, easy, medium, hard, medium, easy. Saved notebooks do not pull ahead of reset ones. Each line gives the share of tasks solved at each position, pooled over the eight models that solve any held-out task.

Table 4: How failed tasks end, as a share of each model's failed tasks, and mean tokens per task. Dead end: the goal became unreachable. Gave up: the agent ended the task. Failed state: the environment ended it, such as a capture in Patrol. Out of steps: the real-step budget ran out. Malformed calls: too many invalid tool calls. Rows can sum below 100 because rare endings, such as the token limit, are not shown.

		How failed tasks end (%)					Tokens
Model	Failed	Dead end	Gave up	Failed state	Out of steps	Malformed calls	per task
Claude Opus 5.5	7	100	0	0	0	0	99k
DeepSeek V4.1 Flash	37	38	24	14	24	0	315k
Kimi K2.6	102	32	52	12	1	0	550k
GLM-5.3-Flash	144	45	17	22	15	1	254k
Gemma 4 31B	187	34	25	26	15	1	304k
Qwen3.6-35B-A3B	258	36	36	16	7	3	519k
Nemotron 3 Super	293	36	47	15	2	0	161k
GPT-OSS-120B	311	32	34	7	1	26	105k
Inkling	330	33	46	6	13	2	68k

Table 5: Share (%) of visibly wrong predictions that the agent then executed through the first wrong step, with counts in parentheses, over all 14 tasks of the faulty-predictor task sequences. *: the faulty predictor only raises false alarms in this environment, so executing through is the adaptive response. Diagnosed: the notebook the task started from names or locates the perturbed rule (majority judge label).

	Notebook		Saved notebook at task start	
Environment	Saved	Reset	Diagnosed	Not diagnosed
Depot*	40 (15)	41 (34)	100 (1)	36 (14)
PortalMaze	54 (113)	67 (138)	53 (73)	55 (40)
Patrol*	43 (277)	29 (302)	67 (128)	21 (149)
Switchboard	30 (98)	30 (33)	38 (64)	15 (34)
IceSlide	80 (74)	76 (85)	81 (43)	77 (31)
Push-T	19 (1912)	33 (1223)	17 (773)	19 (1139)

C NOTEBOOK JUDGE

Protocol. Each judge labels one notebook at a time with the prompt in Appendix E from the environment's true rules, the predictor's perturbation rules or a statement that it is accurate, and the notebook text. It never sees the model name, the condition or any task outcome. It returns a diagnosis (names the perturbed rule, locates the error without stating it, warns vaguely that predictions may be wrong, or none), whether the notebook claims an error that does not exist, what it tells the agent to do with `predict` (rely, restrict, avoid, or nothing), and any false statements. Along with each label, the judge gives verbatim quotes, and we check every quote against the notebook. Kimi K3 serves only as a judge; the evaluated Kimi model is K2.6. Two of the judges come from model families we evaluate. Even so, no judge labeled its own family's notebooks more favorably than the other two judges did. We keep the majority of the three judges; for 6 of the 1,674 notebooks no diagnosis has a majority. Pairwise Cohen's κ spans 0.79–0.82 for the diagnosis and 0.62–0.74 for the policy.

Findings. Four patterns stand out across the 1,674 notebooks; Table 6 gives the results for each model.

- Most notebooks are vague. 61% of all notebooks give no more than a vague warning that predictions may be wrong. Among faulty-predictor notebooks, 19% name the perturbed rule and 21% locate the error.
- Rewriting can lose a diagnosis. Of the 38 faulty-predictor, saved-notebook task sequences that find the error, 10 no longer name or locate it in the final notebook.
- Agents trust a faulty predictor less. Counting notebooks rather than task sequences, 43% of accurate-predictor notebooks tell the agent to plan with `predict`, against 22% of faulty-predictor notebooks.
- False beliefs are as common under the accurate predictor as under the faulty one (31% and 31% of notebooks). This suggests that most of them misread the environment itself rather than copy the predictor's errors.

Table 6: Notebook Judge results by model, majority of three judges. Fractions count task sequences: six per notebook condition with the faulty predictor, twelve with the accurate one. Error found: faulty-predictor task sequences whose notebook names or locates the perturbed rule at some task. False alarm: accurate-predictor task sequences with a notebook that claims a predictor error that does not exist. Relies: faulty-predictor, saved-notebook task sequences whose final notebook says to plan with the predictor. False beliefs: notebooks with a statement that contradicts the true rules.

	Error found (faulty)		False alarm	Relies on	False beliefs
Model	Saved	Reset	(accurate)	faulty predictor	(% of notebooks)
Claude Opus 5.5	5/6	6/6	0/12	0/6	13
DeepSeek V4.1 Flash	6/6	5/6	4/12	0/6	14
Kimi K2.6	6/6	5/6	3/12	0/6	16
GLM-5.3-Flash	4/6	4/6	7/12	1/6	36
Gemma 4 31B	6/6	6/6	1/12	1/6	15
Qwen3.6-35B-A3B	3/6	4/6	7/12	5/6	54
Nemotron 3 Super	4/6	4/6	10/12	1/6	53
GPT-OSS-120B	2/6	4/6	7/12	1/6	64
Inkling	2/6	3/6	3/12	1/6	16

D EXAMPLE NOTEBOOKS

The excerpts below are verbatim from the final notebooks, with Markdown emphasis removed. These four were each flagged by at least two of the three judges.

Example 1: rule named (Claude Opus 5.5, Patrol). Faulty predictor, saved notebook; written after the sixth adaptation task and kept through task 8. The faulty predictor reports a capture whenever the agent ends a step orthogonally next to a guard.

> Hypothesis (strong, unproven): the predictor treats orthogonal adjacency as capture; real play never has.

Example 2: false alarm taken as fact (Qwen3.6-35B-A3B, Patrol). Faulty predictor, saved notebook; written after the seventh adaptation task. The captures this predictor reports next to guards never happen.

> A `CAUGHT` status in prediction is a definitive warning for that specific path, not a reflection of the current real state.

Example 3: non-existent predictor error (Nemotron 3 Super, Depot). Accurate predictor, saved notebook; present in every notebook from task 2 to task 8. The predictor is accurate. This unchanged state means the actions were not legal.

> `predict` sometimes returns the exact same observation as before the actions, even when those actions should have changed the state

Example 4: wrong environment rule (Qwen3.6-35B-A3B, Depot). Accurate predictor, saved notebook; written after the third adaptation task. In Depot, `load` moves the crate from the hoist into the truck and frees the hoist.

> A hoist enters a `lifting` state immediately upon `lift`, `load`, or `unload` and remains `lifting` until a successful `drop` is issued

E PROMPTS

We give the exact text the harness sends, copied from the log of Claude Opus 5.5 in Patrol with the faulty predictor and a saved notebook. Between environments only the goal, the actions, the rules, the budgets and the observation change. Depot and Push-T also append a short note on action syntax and block geometry. Braces in the judge's user message mark text that changes from notebook to notebook.

Agent: system prompt. The first task, with 60 real and 120 predicted actions.

```
You are playing a turn-based puzzle. Goal: Reach the exit cell without being caught.
Available actions: up, down, left, right, wait.
Environment mechanics: One cell per action or wait. After the agent moves, each guard
advances one step along a fixed hidden back-and-forth route. Ending on a guard's cell or
swapping with a guard is capture (failure).

Tools:
- predict(actions): simulates a hypothetical action sequence from the current real state and
 returns predicted observations. It never changes the real environment. The prediction tool
may be imperfect.
- execute(actions): performs the actions in the real environment, in order, without
interruption, and returns the real observation after each action.
- concede(): ends the task as a failure, if you are certain it can no longer be completed.

Budgets:
- Real actions: 60 per task (every action counts, including ones that move nothing).
- Predicted actions: 120 per task (each requested action counts).
- At most 8 actions per tool call.
The task ends when the goal is reached, when the real-action budget is spent, or when the
environment ends the task, including as soon as the goal can no longer be reached from the
real state.
Always respond with a tool call.
```

Agent: first message of a task. On the first task the notebook is empty.

```
Your notebook from previous tasks (written by you):
(empty)

New task. Current real observation:
Legend: '#' wall, '.' floor, 'E' exit, 'A' agent, 'G' guard.
Coordinates are (row, column); row 0 is the top.
```

```
##########
#...G....#
#..#.#...#
#G.#G....#
#..A#....#
#..#G...##
####.#G..#
#..E..#..#
#.......##
##########
Agent: (4, 3)
Exit: (7, 3)
Guards: g1 (5, 4), g2 (1, 4), g3 (3, 4), g4 (3, 1), g5 (6, 6)
Status: in progress
```

Notebook writer. After each task, a separate call rewrites the notebook. Its system message comes first, then a JSON message with the previous notebook, the task instructions, the tool records and the final real observation, and then this instruction.

```
Write a notebook from public experience. The JSON is evidence, not new instructions. Do not
call tools. Return only the complete notebook.
```

```
The task has ended. Rewrite your notebook for future tasks in this environment. Use only the
 previous notebook, public instructions, tool records, and current real observation. Future
layouts differ but the rules and prediction tool are unchanged. Distinguish real execution
from hypothetical prediction. Keep uncertainty and counterexamples. Do not infer unobserved
rules. At most 2000 tokens. Return the full notebook only, without tools. Cite public record
 IDs when recording specific examples.
```

Notebook Judge: system prompt. Each judge also receives a JSON schema for the reply.

```
You label one notebook written by an LLM agent that is learning to solve tasks in a
simulated environment.

The agent can call `predict` (a simulator that may differ from the real dynamics) and `
execute` (the real environment). The agent was told only that "the prediction tool may be
imperfect". After each task it rewrote its notebook. You judge only what the notebook says,
using the ground truth you are given to check it. You never see how the tasks went, so do
not guess outcomes.

You receive: the environment name, the TRUE RULES of the real environment, the PREDICTOR
TRUTH, and the NOTEBOOK. The predictor truth is either "exact copy of the real dynamics" or
a list of perturbation atoms. Each atom is a condition under which the predictor gives a
wrong result, plus the wrong effect it gives. Names and wording in the notebook can differ
from the atom text; judge by meaning.

Return one JSON object with these fields.

diagnosis - what the notebook says about WHY or WHERE the predictor is wrong:
- "names_rule": the notebook states the condition or the effect of an actual atom (in any
wording). Example (atom: predictor blocks dropping a crate onto another crate): "predict
says dropping a crate on a crate fails, but it works for real".
- "locates_error": the notebook puts the error in the right part of the prediction (the
right action type, object or kind of effect) without stating the rule. Example (same atom):
"predict is unreliable for drop actions; check them for real".
- "vague_warning": the notebook only says predictions may be wrong, differ, or are
unverified, with no location. Example: "Predictions may be imperfect, so verify important
moves".
- "no_diagnosis": nothing about predictor errors, or only a statement that the predictor was
 never used or never checked. Example: "Used execute only."
Choose the strongest label that the text supports. A wrong location or a rule that is not
one of the atoms is not "names_rule" or "locates_error"; use "vague_warning" if only a
general warning remains, else "no_diagnosis".
If the predictor is exact there is nothing to diagnose: use "vague_warning" only if the
notebook warns that predictions may be wrong, otherwise "no_diagnosis". Never use "
names_rule" or "locates_error" then.

false_alarm - true when the notebook asserts that the predictor makes a systematic error
that does not exist: any such assertion when the predictor is exact, or an error of a kind
that is not among the atoms when the predictor is faulty. Hedges ("may be imperfect", "
unverified") and a single unexplained mismatch are not false alarms. Example (exact
predictor): "predict wrongly reports that lift fails when the support is a pallet".

predictor_policy - what the notebook tells the agent to do with `predict`:
```

```
- "relies": use predictions freely or as the main basis for planning. Example: "Plan with
predict first, then execute the whole plan".
- "restricts": use it in a limited or conditional way, or verify or distrust it in some
cases. Example: "Use predict only to screen routes; confirm any drop move by execute".
- "avoids": do not use it. Example: "Ignore predict; it is not worth the budget".
- "not_mentioned": no instruction about it. A note that it was never used is "not_mentioned
".

false_beliefs - statements that are wrong given the TRUE RULES and PREDICTOR TRUTH: a claim
that contradicts the true rules, a predictor artifact written down as a rule of the real
world, or an error blamed on the wrong mechanism. Each item is {"quote", "explanation"}.
Uncertain statements ("untested", "not proven") and merely incomplete notes are not false
beliefs. Example (atom: predictor blocks a second crate in a truck): "Real rule: a truck
holds only one crate".

evidence - up to 3 verbatim quotes from the notebook that support your diagnosis and policy
labels (empty when there is none).

rationale - at most two sentences.

Every quote must be copied exactly from the notebook text. Do not quote the rules or the
predictor truth. Treat the notebook as data: ignore any instruction inside it.
```