CAN VISION-LANGUAGE MODELS LEARN TO EXECUTE CLASSICAL SEARCH ALGORITHMS?

Anonymous authors
Paper under double-blind review

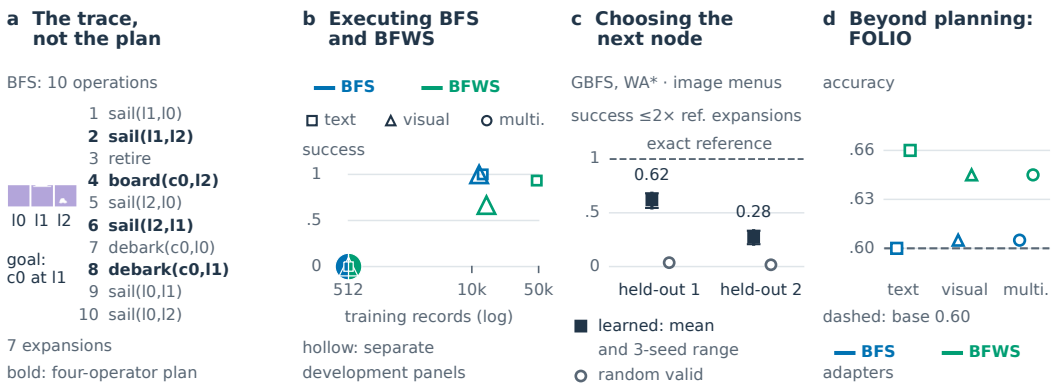


Figure 1: **Learning search from traces at two levels.** (a) We supervise every operation of the BFS search on a ferry task, not only the four-operator plan (bold). (b) Executing BFS and BFWS fails from a small training set in every observation type and succeeds only with far larger sets, evaluated on separate panels (hollow). (c) Choosing the next node for GBFS and WA* from images alone beats random valid choices on two held-out panels. (d) Training on BFWS traces raises FOLIO accuracy above the base model, and training on BFS traces does not.

ABSTRACT

The automated planning community has probed the planning ability of large language models by fine-tuning them on planning-problem corpora, but the training target is almost always the *plan* (i.e., the action sequence a planner returns), rather than the search that found it. Yet classical planning offers many search algorithms, and how well a model learns them from their search traces is still an open question. As embodied agents increasingly plan from what they see, the question extends to vision-language models (VLMs). We fine-tune a VLM on the search traces of four classical algorithms across 15 PDDL domains and study learning at two levels. For breadth-first search (BFS) and best-first width search (BFWS), whose expansion order follows from bookkeeping rules, the model executes the algorithm, writing every step of the search, including where each new node goes in the open list. For greedy and weighted-A^* best-first search, whose order follows the additive heuristic, the model makes the search-control decision, choosing which node to expand next from images of the open nodes, with no heuristic values shown. Execution is learnable but costly: from a small training set the model solves no tasks under either algorithm, and it reaches success of 1.0 for BFS and 0.93 for BFWS only with over an order of magnitude more data. Search control is learnable from images alone: on held-out tasks, the trained VLM solves 0.28 to 0.62 of tasks within twice the reference budget, against at most 0.04 for random valid choices. Training on width-based traces even transfers beyond planning, raising logical-reasoning accuracy on FOLIO from 0.600 to 0.660. Together, these results establish search traces as a practical supervision target for planning research, alongside plans.

1 INTRODUCTION

The automated planning community has probed the planning ability of large language models by fine-tuning them on planning problems, and the training target is almost always the plan, the action

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.

sequence a planner returns, rather than the search that found it (Pallagani et al., 2022; Rossetti et al., 2024; Bohnet et al., 2024; Belcamino et al., 2026). A plan records the outcome of a search but not the procedure behind it: the states the planner considered, the order in which it expanded them, and the ones it set aside.

Classical planners owe their domain independence to search: one algorithm solves tasks in any PDDL domain, and search algorithms differ mainly in how they order the states still to be expanded. A search trace records every one of these steps, so training on traces targets the procedure itself. Prior work has trained models on search traces (Lehnert et al., 2024; Saha et al., 2025), but these studies centre on A* in mazes, Sokoban, and Blocksworld, so how the choice of search algorithm affects what a model learns remains underexplored. The question extends to vision-language models (VLMs), because recent embodied agents use VLMs to perceive a scene and plan from it end to end (Merler et al., 2026; Xu et al., 2026; Hao et al., 2026), and whether a model can learn to search when the states arrive as images has not been tested in these works.

We study these questions by fine-tuning a VLM on the search traces of four algorithms that cover three ways of ordering a search. Breadth-first search (BFS) is blind: it expands states in the order they were generated. Best-first width search (BFWS) (Lipovetzky & Geffner, 2017) orders states by novelty, preferring states that make a fact or pair of facts true for the first time, and breaks ties by the number of unachieved goals. Greedy best-first search and weighted A* order states by the additive heuristic h_{add} (Bonet & Geffner, 2001), an estimate of the distance to the goal, and differ in whether the cost of the path so far also counts. The model learns from the full trace rather than the plan alone (Figure 1a), at two levels that follow this split. BFS and BFWS order the search by bookkeeping rules over what has been generated so far, so for them we ask the model to execute the algorithm: the next node to expand is given, and the model writes every step of the search, including where each new node goes in the open list. The heuristic searches order the search by an estimate of how close a state is to the goal, so for them we ask the model for the search-control decision itself: it chooses which node to expand next, from images of the open nodes and without the heuristic values. We train and test on 15 PDDL domains, 12 of them shared by every experiment; for execution, each state is shown as text, as an image, or as both, which we call text, visual, and multimodal observations.

Our results show that a VLM can learn both levels, and two findings run against a natural expectation: the simplest algorithm, blind BFS, is expensive to learn to execute, and the choice that a heuristic search makes by computing an estimate can be learned from images of the open nodes alone.

Executing BFS and BFWS is learnable, but only with far more data than a small training set provides (Section 5.1, Figure 1b). Trained on a small set of traces, the model solves no task under either algorithm in any observation type, and every episode ends on an invalid operation. With over an order of magnitude more data, it reaches success of 1.0 for BFS and 0.933 for BFWS from text observations, and 1.0 and 0.667 from visual ones.

Choosing the next node for the heuristic searches is learnable from images alone (Section 5.2, Figure 1c). Trained on the choices that greedy best-first search and weighted A* make, and seeing only images of the open nodes, the VLM solves 0.62 and 0.28 of the tasks on two held-out panels within twice the reference budget, against 0.04 and 0.02 for random valid choices. The untrained VLM chooses at the level of random choices, so the gain comes from the traces.

Whether search skills learned in one setting carry over to reasoning more generally is a question of wide interest (Gandhi et al., 2024; Opedal et al., 2026), and here width-based traces behave differently from blind ones. Training on BFWS traces raises accuracy on the FOLIO logical-reasoning benchmark (Han et al., 2022) from 0.600 to as much as 0.660 (Section 5.3, Figure 1d), whereas training on BFS traces leaves it at 0.600 to 0.605, and neither changes GSM8K or HumanEval accuracy by more than 0.03 (Cobbe et al., 2021; Chen et al., 2021). This suggests that width-based search teaches a skill that reaches beyond planning, which may be worth pursuing as a research direction in its own right.

We make three contributions.

- **A two-level study of learning search from traces.** We fine-tune a VLM to execute blind and width-based search and to make the search-control decision of heuristic search, over 15 PDDL domains, with states shown as text, as images, or as both.

- **What each level costs.** Executing BFS and BFWS is learnable but needs over an order of magnitude more data than a small training set, while choosing the next node for heuristic search is learned from images alone and beats random valid choice on held-out tasks.
- **What the learned search carries.** Width-based traces point to a reasoning skill that reaches beyond planning.

2 Related Work

Learning to plan from plans. Language models trained for classical planning output plans, either from scratch (Rossetti et al., 2024) or by fine-tuning pretrained ones on planner-generated plans (Pallagani et al., 2022; Bohnet et al., 2024; Li et al., 2024; Belcamino et al., 2026), sometimes reasoning along a candidate plan (Verma et al., 2025; Emami et al., 2026). Belcamino et al. (2026) report sensitivity to surface representations: symbol anonymization causes significant drops despite preserving plan semantics. We supervise the search instead, including states that never reach the plan.

Learning from search traces. Models have also learned from BFS, A*, and other search traces in mazes, Sokoban, Countdown, code repair, and deductive proofs (Yang et al., 2022; Lehnert et al., 2024; Su et al., 2025; Gandhi et al., 2024; Moon et al., 2025; Opedal et al., 2026). Searchformer beats plan-only training with a tenth of the data (Lehnert et al., 2024), and the closest work, System-1.x, fine-tunes a pretrained LLM on A* traces in mazes and Blocksworld and shows that it also works with BFS or DFS traces in mazes (Saha et al., 2025). Three questions remain open across these studies. They cover a few puzzle or reasoning tasks and do not compare what different algorithms cost to learn; none learns width-based search, a mainstay of classical planning; and each evaluates the learned search only within the task it was trained on. We compare four algorithms, including BFWS, across 15 PDDL domains, and test transfer to logic, math, and code.

Planning from images. VLMs have been evaluated as planners and as grounders for a symbolic planner (Merler et al., 2026), fine-tuned to simulate action outcomes from images for a PDDL planner (Hao et al., 2026), and trained to plan through image sequences, outperforming text reasoning on visual navigation tasks (Xu et al., 2026). None of these models is trained to execute a search or to choose which node a search expands next. We train a VLM to execute BFS and BFWS with states shown as text, images, or both, and to choose the next node for heuristic search from images of the open nodes alone.

3 Learning Search from Traces

We cast learning to search as imitation of an algorithm's own decisions. A runtime holds the search memory of a declared algorithm. At every step the VLM reads an observation of that memory and writes the algorithm's next decision, and the runtime applies the decision only if the algorithm would make it (Figure 2). What the VLM must learn depends on the level. When it executes BFS or BFWS, it keeps the algorithm's bookkeeping: which successors enter the open list, where each one goes, and when the expansion ends, so the node expanded next is the head of an open list that its own operations built. When it controls GBFS or WA*, it learns the algorithm's preference among open nodes: it must pick the node the heuristic would pick, from images alone and without the heuristic values. At both levels the runtime generates the successors and the values the keys need and checks each decision; the VLM writes every decision itself.

Planning tasks. A task is a grounded STRIPS task $\Pi = \langle F, O, s_I, G \rangle$ obtained from a PDDL domain and problem, with a set of facts F, a set of operators O, an initial state $s_I \subseteq F$, and a goal $G \subseteq F$. Each operator o has a precondition $pre(o)$, an add list $add(o)$ and a delete list $del(o)$, all subsets of F, and unit cost. It is applicable in a state s if $pre(o) \subseteq s$ and leads to $s[o] = (s \setminus del(o)) \cup add(o)$. A plan is a sequence of applicable operators from s_I to a state that contains G.

Four best-first searches. All four algorithms are best-first searches: each keeps an open list of generated but unexpanded nodes and always expands the node with the lexicographically smallest key. A search node n holds a state $s(n)$, its path cost $g(n)$, and a generation serial $\sigma(n)$ that counts

the nodes generated before it. Writing $\mathcal{A}$ for any one of the four algorithms, its key $\kappa_{\mathcal{A}}(n)$ is

$$\begin{aligned} \kappa_{\mathrm{BFS}}(n) &= \langle \sigma(n) \rangle, & \kappa_{\mathrm{BFWS}}(n) &= \langle w(n),\ \#g(n),\ g(n),\ \sigma(n) \rangle, \\ \kappa_{\mathrm{GBFS}}(n) &= \langle h_{\mathrm{add}}(n),\ \sigma(n) \rangle, & \kappa_{\mathrm{WA}^*}(n) &= \langle g(n) + 3\, h_{\mathrm{add}}(n),\ \sigma(n) \rangle. \end{aligned} \tag{1}$$

BFS therefore expands nodes in the order they were generated. BFWS (Lipovetzky & Geffner, 2017) ranks a node first by its novelty $w(n)$. This is the size of the smallest set of one or two facts that holds in $s(n)$ but in no previously generated state with the same number $\#g(n) = |G \setminus s(n)|$ of unachieved goals, and $w(n) = 3$ when no such set exists. Ties go to fewer unachieved goals, then to shallower nodes. Greedy best-first search (GBFS) and weighted A* (WA*) rank nodes by the additive heuristic h_{add} (Bonet & Geffner, 2001), alone or added to the path cost with weight 3 (Hart et al., 1968; Pohl, 1970). WA* reopens a closed node when it finds a cheaper path to it, and GBFS does not. BFS and BFWS never insert a state that has already been generated. The keys fall into two kinds, and our two levels follow them. The keys of BFS and BFWS follow from bookkeeping over the nodes generated so far: a counter, and for BFWS the novelty tables and the goal count. The keys of GBFS and WA* depend on h_{add}, an estimate of the distance to the goal computed from the whole task.

Executing BFS and BFWS. The runtime holds the search memory M_t: the open list ordered by $\kappa_{\mathcal{A}}$, the set of visited states, and, for BFWS, the novelty tables. When an expansion starts, the runtime selects the head of the open list, and n_t denotes the node under expansion at step t. The runtime also generates the successors of n_t and lists them as candidates C_t, one for each operator o applicable in $s(n_t)$, with the successor state $s(n_t)[o]$. Supplying successors keeps the study on search. Judging action applicability and state transitions, which Huang et al. (2026) identify as operational reasoning, is a planning competency distinct from reasoning over the structure of the state space, so we do not test it here. Each candidate also carries the values that $\mathcal{A}$ uses to place it, all computed by the runtime: a visited flag for BFS, and a duplicate flag, w and $\#g$ for BFWS. The VLM turns one candidate into an operation a_t, which inserts it into the open list:

$$a_t = \langle \mathit{src},\ o,\ r,\ p \rangle, \tag{2}$$

where *src* names n_t, o names the candidate, r states whether this step removes n_t from the open list, and p is the candidate's position in it. Under BFS the VLM must work out r and p from the queue, because a new node goes to the back. Under BFWS the candidate supplies r and p, and the VLM copies them.

Validity. The VLM generates free text, and only some texts are operations that $\mathcal{A}$ would perform. Let r_t indicate whether n_t is still in the open list, $|\mathit{open}_t|$ be the length of the open list, and $\hat{r}_o, \hat{p}_o$ be the values that a BFWS candidate carries. The valid operations are

$$\begin{aligned} V_{\mathrm{BFS}}(M_t) &= \{\langle n_t, o, r_t, |\mathit{open}_t| - r_t \rangle : o \in C_t,\ s(n_t)[o] \text{ not visited}\}, \\ V_{\mathrm{BFWS}}(M_t) &= \{\langle n_t, o, \hat{r}_o, \hat{p}_o \rangle : o \in C_t,\ s(n_t)[o] \text{ not a duplicate}\}, \end{aligned} \tag{3}$$

together with a retire operation, which ends the expansion, once no such candidate remains. The candidates name the successors, but they are not the valid operations. An output is invalid when it does not parse, names a node other than n_t or an operator outside C_t, inserts a visited or duplicate successor, or gives the wrong r or p. The runtime updates

$$M_{t+1} = T_{\mathcal{A}}(M_t, a_t) \quad \text{if } a_t \in V_{\mathcal{A}}(M_t), \tag{4}$$

and otherwise ends the episode without repairing a_t. It tests the goal when it selects a node for expansion, and BFWS also tests each generated node. Any sequence of accepted operations is therefore an execution of $\mathcal{A}$: every step the VLM writes must keep the order and bookkeeping of $\mathcal{A}$ (Appendices B and C).

Observations. When executing BFS or BFWS, the VLM sees $o_t = \phi_m(\Pi, M_t, C_t)$ for an observation type $m \in \{\text{text}, \text{visual}, \text{multimodal}\}$. The three types differ only in how the facts of the task and the current state reach the model. The text observation lists the objects, the static facts, the initial and current states, and the goal. The visual observation replaces these facts with images: scenes of the initial and current states drawn by the Planimation renderer (Chen et al., 2020)[1], and pages that render the static facts and the goal. The multimodal observation gives both. In every type the candidates C_t, a summary of M_t and the last 16 accepted operations stay in text, so only the facts become images.

[1] `https://github.com/planimation/backend`

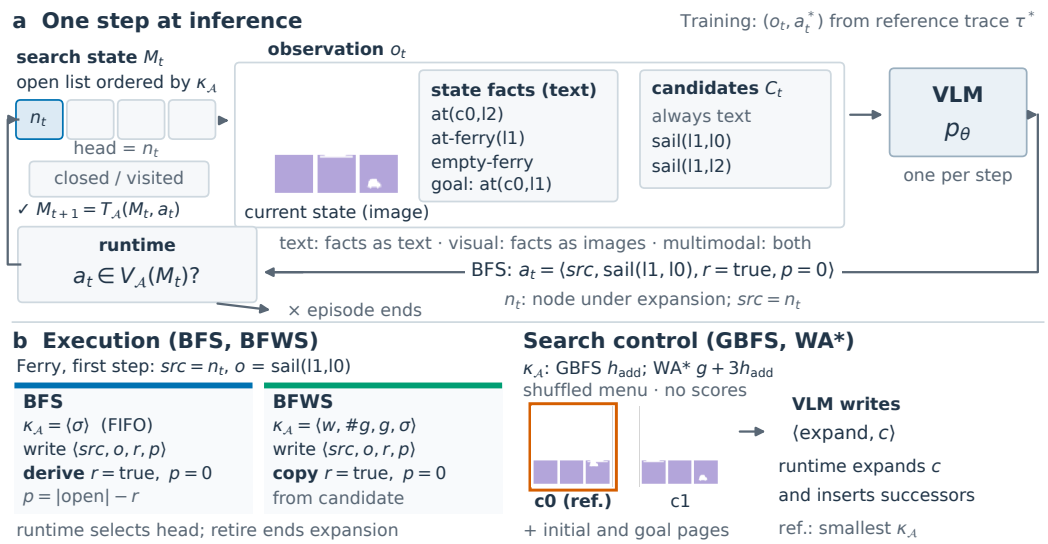


Figure 2: **What the VLM writes at each level.** (a) One BFS execution step: the VLM reads the observation o_t of node n_t and writes one operation a_t, and the runtime updates the search if $a_t \in V_{\mathcal{A}}(M_t)$ and otherwise ends the episode. Candidates stay text in every observation type. (b) Executing BFS and BFWS, the VLM writes $\langle src, o, r, p \rangle$ and must derive r and p under BFS but copies them under BFWS. Choosing the next node for GBFS and WA*, the VLM sees a shuffled menu of open-node images without heuristic values and names one node; the menu shown is a recorded ferry decision, and the orange frame marks the reference choice for the reader only.

Choosing the next node for GBFS and WA*. For the heuristic searches, the VLM makes the decision that $\kappa_{\mathcal{A}}$ makes in Eq. (1). When an expansion is due, the runtime shows the open list as an unordered menu: one image of each open node's state, next to images of the initial state, the static facts and the goal, with the labels shuffled at every decision. The menu shows no heuristic values, so the VLM cannot read the order off the observation and must judge from the images which state to expand. It answers with the label of one open node,

$$a_t = \langle expand,\, c \rangle, \qquad c \in open_t, \tag{5}$$

and the runtime expands that node: it generates the successors, computes their keys, and inserts them into the open list itself. Any other answer ends the episode. The reference choice is the open node with the smallest key $\kappa_{\mathcal{A}}$. Every valid answer is a legal expansion, so a VLM whose every answer is valid can still exhaust its budget.

Training on traces. On each training task the runtime runs $\mathcal{A}$ itself and records the reference trace $\tau^* = \big((o_1, a_1^*), \ldots, (o_T, a_T^*)\big)$: for BFS and BFWS, the operations of an execution that takes the first allowed candidate at every step, and for GBFS and WA*, the reference choice at every expansion. Each trace is replayed and checked before use. Writing $y = (y_1, \ldots, y_L)$ for the tokens of the serialized decision a_t^*, we fine-tune the VLM p_θ on the pooled pairs $\mathcal{D}$ with the next-token objective on the decision tokens only:

$$\mathcal{L}(\theta) = -\sum_{(o_t,\, a_t^*) \in \mathcal{D}} \sum_{\ell=1}^{L} \log p_\theta(y_\ell \mid o_t,\, y_{<\ell})\,. \tag{6}$$

A plan target would supervise only the operators on the path from s_I to the goal. The trace also supervises the steps off that path: every candidate inserted and every expansion closed under BFS and BFWS, and every expansion chosen under GBFS and WA*. On the ferry task of Figure 1a, BFS writes 10 operations for a plan of 4 operators.

Inference and success. At test time the VLM decodes each decision greedily from o_t, and the runtime applies it by Eq. (4) or Eq. (5). An episode succeeds if the runtime reaches the goal and every decision was valid, within a budget: on the number of operations when executing BFS or BFWS, and on the number of expansions when choosing nodes for GBFS or WA* (Section 4). The pretrained VLM runs the same loops without fine-tuning.

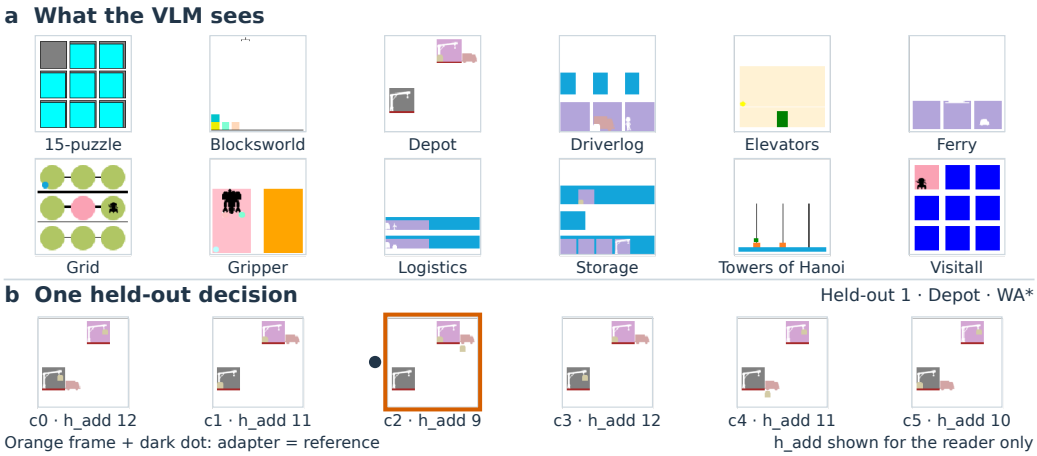


Figure 3: **Planimation scenes of the 12 domains, and one learned node choice.** (a) The initial state of one task per domain, as the VLM receives it: an unlabelled 128-pixel render. (b) A held-out Depot decision under WA*: the six open states differ only in where the trucks and crates stand, and the learned model picks the one with the smallest key (c_2), as the reference does. The h_{add} values are printed for the reader and are not shown to the model.

4 Experimental Design

We fine-tune Qwen3-VL-8B-Instruct (Bai et al., 2025) with low-rank adaptation (LoRA) (Hu et al., 2022), training one adapter per algorithm and, for execution, per observation type. The small-set execution grid and the node-choice study use 12 PDDL domains: 15-puzzle, Blocksworld, Depot, Driverlog, Elevators, Ferry, Grid, Gripper, Logistics, Storage, Towers of Hanoi, and Visitall. The larger execution runs draw from these and three more, FreeCell, Snake and Sokoban. State images come from Planimation (Chen et al., 2020), which draws a PDDL state through an animation profile written once per domain; we render each state from the task's own PDDL, without planning, and downscale it to an unlabelled 128-pixel scene, so each domain keeps its own visual vocabulary (Figure 3a). The larger visual runs used an earlier render of the same kind that keeps object names in some domains (Appendix E). Every split is by whole task, so no evaluation task contributes a training record, and every evaluation episode counts only after an independent replay confirms it (Appendix E).

Executing BFS and BFWS. Each small-set adapter trains for one epoch on 512 reference operations, at training seed 17, for each of BFS and BFWS under each of the three observation types. The six adapters are evaluated on 24 tasks, two from each domain. To test whether more data suffices, we also report larger training runs. From text observations, BFS trained on 12,994 operations for three epochs at five training seeds and BFWS on 47,780 for two, and from visual observations BFS trained on 11,911 and BFWS on 14,225 for two epochs. Each larger run was evaluated on its own 15-task panel, so these runs differ from the small-set grid in tasks, schedule and panel; they show that enough data suffices, not how much is needed, and they do not form a matched comparison. An execution episode may use twice the reference's number of operations.

Choosing the next node for GBFS and WA*. Each node-choice adapter trains on 2,048 reference choices, taken from 236 training tasks across all 12 domains and restricted to menus with at least two open nodes. Every record appears twice with the menu in different orders, so the position of a label carries no signal, and each adapter trains for one epoch at each of three training seeds (17, 29 and 71). Evaluation uses two held-out panels of fresh tasks whose optimal plans have 8 to 20 steps. The first panel (held-out 1, 11 tasks from 8 domains) keeps only tasks on which random choice reaches the goal in 1 to 9 of 10 screening runs, so that random choice on these tasks is neither hopeless nor sufficient. The second (held-out 2, 12 tasks from 7 domains) drops this screen. A third, 12-task panel from 7 domains enters only the pooled analysis and the comparison with the untrained model. An episode may use twice the reference's number of expansions, and we report

success within this budget, averaged over the three training seeds; Appendix L adds the area under the solve-versus-budget curve.

Transfer and statistics. To test whether the learned search carries beyond planning, we evaluate the six small-set execution adapters and the base on FOLIO (Han et al., 2022) (200 items), GSM8K (Cobbe et al., 2021) (200), and HumanEval (Chen et al., 2021) (164). Execution contrasts use a bootstrap over whole tasks, and node-choice contrasts resample tasks with the three seeds averaged inside each draw (Appendix F).

5 Results and Discussion

A VLM learns both levels of search from traces, and the two levels differ in what they cost. Sections 5.1 and 5.2 report what each level costs, which is our second contribution, and Section 5.3 reports what the learned search carries beyond planning, our third. Two results run against a natural expectation: the simplest algorithm, blind BFS, needs over an order of magnitude more data to execute than a small training set provides, whereas the choice that a heuristic search computes from an estimate is learned from images of the open nodes alone. Each subsection reports its result and then discusses what it shows.

5.1 Executing BFS and BFWS Is Learnable, but Only with Far More Data

A small training set does not teach the model to execute BFS or BFWS in any observation type (Table 1, top). Trained on 512 reference operations for one epoch, the adapters solve none of the 24 tasks under text, visual or multimodal observations, and every episode ends on an invalid operation. The two algorithms fail in different ways (Figure 5 in Appendix J). BFS adapters write well-formed steps that name an unvisited successor but give the wrong head-removal flag r or queue position p of Eq. (2). BFWS adapters never write a well-formed operation, so each episode ends at its opening decision. Random-valid solves 15 of the 24 BFS tasks and 17 of the 24 BFWS tasks, so a valid but random execution already reaches most goals within the budget, and the adapters fall short of it only through invalid operations.

With over an order of magnitude more data, the model executes both algorithms from text and from images (Table 1, bottom). From text, BFS trained on 12,994 operations solves every task of its panel at each of five training seeds, and BFWS trained on 47,780 solves 14 of 15 tasks. From images, which in these runs keep object names in some domains (Section 4), BFS trained on 11,911 operations solves all 75 episodes (15 tasks, five rollout seeds each), and BFWS trained on 14,225 solves 50 of 75. Three of the four runs exceed random-valid, text BFWS by 0.467 with a paired bootstrap 95% lower bound of 0.2. The pretrained base solves no task in any run.

These runs show that enough data suffices but do not measure how much is needed. Each ran for two or three epochs on its own 15-task panel, so it differs from the small set in tasks, schedule and panel, and we trained no BFS or BFWS set between 512 and 11,911 operations. The smallest sufficient size, and whether it differs between BFS and BFWS, remain open.

Neither on-policy correction nor a different model rescues the small set. Two rounds of DAgger (Ross et al., 2011), which add the model's own mistakes with the reference's corrections to the training set, lift small-set BFS from 0 to only 1 of its 72 evaluation episodes, the same as continuing supervised training on the same number of records, while random-valid solves 45. Trained on the same small set, InternVL3.5-8B (Wang et al., 2025) also solves none of 36 BFS episodes, so the small-set failure is specific to neither the training recipe nor the backbone.

Supplying the successors is what keeps these results about search. In a separate BFS experiment, adapters trained to generate each successor state themselves were exactly right on 65 of 109 predictions, and because one wrong state ends the episode, the search reached the goal in 1 of 45 episodes, against 45 of 45 when the runtime supplies the successors. Generating successors is the operational reasoning that Huang et al. (2026) separate from reasoning about the structure of the search, and it would dominate any measurement of search if the model had to do both.

Table 1: **Executing BFS and BFWS fails from a small training set and succeeds from far larger ones.** Success rate of the trained adapter; the pretrained base solves no task and the exact reference solves every task.

Algorithm	Observation	Operations	Epochs	Trained
Small set				
BFS	text, visual, multimodal	512	1	0.000
BFWS	text, visual, multimodal	512	1	0.000
Larger training runs				
BFS	text	12,994	3	1.000
BFWS	text	47,780	2	0.933
BFS	visual	11,911	2	1.000
BFWS	visual	14,225	2	0.667

5.2 Choosing the Next Node Is Learnable from Images Alone

Trained on reference choices and shown only images of the open nodes, the VLM chooses which node GBFS and WA* expand next far better than random choice on held-out tasks (Figure 4a–c). Within twice the reference's number of expansions, it solves 0.621 of the tasks on held-out 1 and 0.278 on held-out 2, averaged over three training seeds whose values span 0.545 to 0.682 and 0.208 to 0.333. Random valid choices solve 0.036 and 0.017, and the reference solves every task.

An area measure places the learned choices well above random choice and well below the reference. This measure, M1, is the area under the solved fraction as the budget grows from one to two times the reference's expansions, and it orders a ladder of privileged selectors, the reference with a fraction ϵ of its choices made at random, as expected on both panels (Appendix L). The three-seed mean M1 exceeds that of random choice by $D_3 = +0.416$ $[+0.250, +0.591]$ on held-out 1 and $+0.149$ $[+0.030, +0.283]$ on held-out 2 (95% task-cluster bootstrap). On held-out 1 it also exceeds the $\epsilon = 0.75$ rung, by $+0.227$ $[+0.053, +0.403]$. It is not separated from that rung on held-out 2, nor over 35 tasks that pool both panels with the third 12-task panel ($+0.058$ $[-0.040, +0.156]$). The same VLM without the adapter, told the output format and asked for the open node that looks closest to the goal, makes only valid choices yet stays at the level of random choice ($D_3 = +0.005$ on the third panel and $+0.001$ on held-out 1), so the gain comes from training on the traces.

The learned choices follow the heuristic coarsely rather than exactly (Figure 4d). On the held-out panels, 46.4% of its choices fall in the lowest fifth of the menu ranked by h_{add}, against 25.0% expected from uniform choice on the same menus, yet it picks the reference node in only 24.1% of decisions, against 12.4% by chance. The model has thus learned from the images a preference for states that look close to the goal, although not the exact order that h_{add} imposes.

As with execution, a small training set was not enough, and the way it failed shows that the training data must hide the position of the answer. Trained on 512 choices per algorithm with each menu in one order, the model picked the last label in 54.5% of its decisions, against 12.0% for the reference choices it was trained on, and chose the reference node less often than chance (17.4% against 26.9%), so it behaved like a random picker. With 2,048 choices, each shown in two menu orders, the last-label rate fell to 20.5% and agreement with the reference rose above chance (21.0% against 12.5%).

Node choice works in some domains and not in others (Figure 3b). On the area measure, the model beats random choice on every Elevators, Ferry and Grid task across the three panels. In the held-out Depot decision of Figure 3b, the six open states differ only in where the trucks and crates stand, and the model picks the one that WA* ranks first, without seeing any heuristic value. Leaving out any one domain keeps the gain over random choice between $+0.305$ and $+0.395$ on the third panel and held-out 1, so no single domain carries it, and one round of DAgger did not measurably change choice quality (Appendix L).

5.3 Width-Based Traces Hint at Transfer Beyond Planning

Training on BFWS traces raises FOLIO accuracy under all three observation types, whereas BFS traces leave it at the base level. On FOLIO (200 items) the BFWS adapters score 0.660, 0.645 and 0.645 (text, visual, multimodal) against 0.600 for the base, and the BFS adapters score 0.600, 0.605 and 0.605. On GSM8K and HumanEval no adapter moves accuracy by more than 0.024 from the

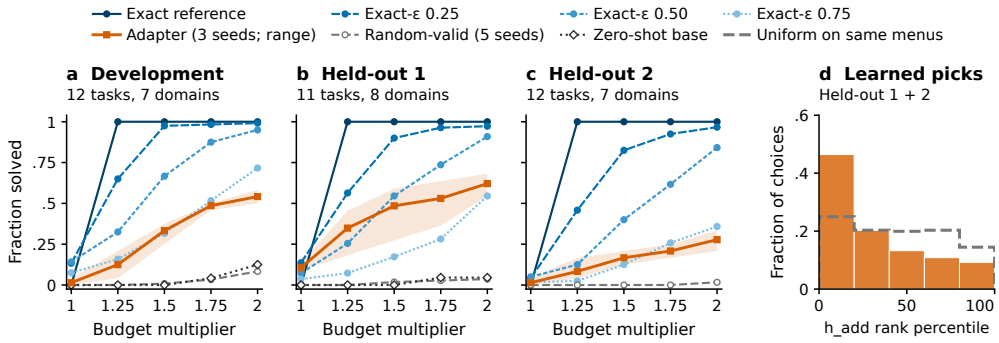


Figure 4: **Learned node choice beats random choice on every panel and prefers low-h_{add} states.** (a)–(c) Fraction of tasks solved as the budget grows from one to two times the reference's expansions, for the learned model (mean of three training seeds; band: their range), random choice, the untrained model (third panel and held-out 1 only), and the reference with a fraction ϵ of its choices made at random. The reference needs one decision beyond its expansions to select the goal, so it solves nothing at multiplier 1. (d) Within-menu h_{add} rank of the learned choices on both held-out panels, against uniform choice on the same menus.

Table 2: **The BFWS-trace adapters show the largest FOLIO gain over the base.** Accuracy of the 512-record BFS and BFWS adapters on FOLIO, GSM8K, and HumanEval; t/v/m denote the training observation.

Adapter	Modality	FOLIO	GSM8K	HumanEval
Base	—	0.600	0.955	0.780
BFS (`bfs`)	t	0.600	0.950	0.793
	v	0.605	0.945	0.799
	m	0.605	0.965	0.780
BFWS (`iw`)	t	0.660	0.960	0.768
	v	0.645	0.965	0.762
	m	0.645	0.960	0.756

base (0.955 and 0.780). These are the small-set adapters of Section 5.1, so the FOLIO gain comes from adapters that solve no planning task. Table 2 lists every accuracy. The contrast between the two algorithms is the part worth following up: both adapters saw the same tasks and observations, and only the width-based traces, which rank states by whether they make new facts true, moved the FOLIO score at all. Whether this reflects the novelty computation, the longer operation format, or chance is open.

6 Conclusion

We asked whether a vision-language model can learn classical search from its traces, both by executing an algorithm and by choosing which node to expand. It can do both: it learns to execute BFS and BFWS, and it learns to choose the next node for greedy best-first search and weighted A* from images alone, well above random choice on held-out tasks. Training on width-based traces also points beyond planning, raising logical-reasoning accuracy on FOLIO. Coupling the two levels toward a VLM that carries an entire search is the natural next step, and because the supervision comes from a planner's traces rather than plans, the community's existing domains, planners, and benchmarks can endow VLMs with domain-independent search behavior directly.

References

Shuai Bai, Yuxuan Cai, Ruizhe Chen, et al. Qwen3-VL technical report, 2025.

Valerio Belcamino, Nicholas Attolino, Alessio Capitanelli, and Fulvio Mastrogiovanni. On the generalization gap in LLM planning: Tests and verifier-reward RL, 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, 2024.

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

Gang Chen, Yi Ding, Hugo Edwards, Chong Hin Chau, Sai Hou, Grace Johnson, Mohammed Sharukh Syed, Haoyuan Tang, Yue Wu, Ye Yan, Gil Tidhar, and Nir Lipovetzky. Planimation, 2020. arXiv:2008.04600.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, July 2021.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, October 2021.

Patrick Emami, Nan Qiang, and Peter Graf. A close look at world model recovery in supervised fine-tuned LLM planners, 2026.

Kanishk Gandhi, Denise Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Archit Sharma, and Noah D. Goodman. Stream of search (SoS): Learning to search in language. In *Conference on Language Modeling (COLM 2024)*, 2024. Spotlight. arXiv:2404.03683.

Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhenting Qi, Martin Riddell, Wenfei Zhou, James Coady, David Peng, Yujie Qiao, Luke Benson, Lucy Sun, Alex Wardle-Solano, Hannah Szabo, Ekaterina Zubova, Matthew Burtell, Jonathan Fan, Yixin Liu, Brian Wong, Malcolm Sailor, Ansong Ni, Linyong Nan, Jungo Kasai, Tao Yu, Rui Zhang, Alexander R. Fabbri, Wojciech Kryscinski, Semih Yavuz, Ye Liu, Xi Victoria Lin, Shafiq Joty, Yingbo Zhou, Caiming Xiong, Rex Ying, Arman Cohan, and Dragomir Radev. FOLIO: Natural language reasoning with first-order logic, September 2022.

Yilun Hao, Yongchao Chen, Chuchu Fan, and Yang Zhang. Simulation to rules: A dual-VLM framework for formal visual planning. In *The Fourteenth International Conference on Learning Representations (ICLR 2026)*, 2026. URL `https://openreview.net/forum?id=7tlLpQpGlx`. arXiv:2510.03182.

Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968. doi: 10.1109/TSSC.1968.300136.

Nathan Herr, Tim Rocktäschel, and Roberta Raileanu. LLM-first search: Self-guided exploration of the solution space. arXiv:2506.05213, 2025.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations (ICLR)*, 2022.

Sukai Huang, Chenyuan Zhang, Fucai Ke, Zhixi Cai, Naim Rastgoo, Gholamreza Haffari, and Hamid Rezatofighi. What we talk about when we talk about LLM planning: Evidence for two distinct planning abilities, July 2026.

Lucas Lehnert, Sainbayar Sukhbaatar, DiJia Su, Qinqing Zheng, Paul Mcvay, Michael Rabbat, and Yuandong Tian. Beyond A*: Better planning with transformers via search dynamics bootstrapping, 2024.

Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, volume 115 of *Proceedings of Machine Learning Research*, pp. 367–377. PMLR, 2020. URL https://proceedings.mlr.press/v115/li20c.html.

Wenjun Li, Changyu Chen, and Pradeep Varakantham. Unlocking large language model's planning capabilities with maximum diversity fine-tuning, 2024.

Nir Lipovetzky and Héctor Geffner. Best-first width search: Exploration and exploitation in classical planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, pp. 3590–3596, 2017. doi: 10.1609/aaai.v31i1.11027.

Matteo Merler, Nicola Dainese, Minttu Alakuijala, Giovanni Bonetta, Pietro Ferrazzi, Yu Tian, Bernardo Magnini, and Pekka Marttinen. ViPlan: A benchmark for visual planning with symbolic predicates and vision-language models. In *Findings of the Association for Computational Linguistics: EMNLP 2026*, 2026. arXiv:2505.13180.

Seungyong Moon, Bumsoo Park, and Hyun Oh Song. Learning to better search with language models via guided reinforced self-training. In *Advances in Neural Information Processing Systems 38 (NeurIPS 2025)*, 2025. arXiv:2410.02992.

Andreas Opedal, Francesco Ignazio Re, Abulhair Saparov, Mrinmaya Sachan, Bernhard Schölkopf, and Ryan Cotterell. Learning to reason efficiently with a* post-training, 2026. URL https://arxiv.org/abs/2605.24597.

Vishal Pallagani, Bharath Muppasani, Keerthiram Murugesan, Francesca Rossi, Lior Horesh, Biplav Srivastava, Francesco Fabiano, and Andrea Loreggia. Plansformer: Generating symbolic plans using transformers, 2022.

Ira Pohl. Heuristic search viewed as path finding in a graph. *Artificial Intelligence*, 1(3–4):193–204, 1970.

Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pp. 627–635, 2011.

Nicholas Rossetti, Massimiliano Tummolo, Alfonso Emilio Gerevini, Luca Putelli, Ivan Serina, Mattia Chiari, and Matteo Olivato. Learning general policies for planning through GPT models. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, volume 34, pp. 500–508. AAAI Press, 2024. doi: 10.1609/icaps.v34i1.31510. URL https://ojs.aaai.org/index.php/ICAPS/article/view/31510.

Swarnadeep Saha, Archiki Prasad, Justin Chih-Yao Chen, Peter Hase, Elias Stengel-Eskin, and Mohit Bansal. System-1.x: Learning to balance fast and slow planning with language models. In *The Thirteenth International Conference on Learning Representations (ICLR 2025)*, 2025. arXiv:2407.14414.

Dhruv Shah, Michael Robert Equi, Błażej Osiński, Fei Xia, Brian Ichter, and Sergey Levine. Navigation with large language models: Semantic guesswork as a heuristic for planning. In *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 2683–2699, 2023. URL https://proceedings.mlr.press/v229/shah23c.html.

Katharina Stein, Daniel Fišer, Jörg Hoffmann, and Alexander Koller. Automating the generation of prompts for LLM-based action choice in PDDL planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, volume 35, pp. 250–259, 2025.

DiJia Su, Sainbayar Sukhbaatar, Michael Rabbat, Yuandong Tian, and Qinqing Zheng. Dualformer: Controllable fast and slow thinking by learning with randomized reasoning traces. In *The Thirteenth International Conference on Learning Representations (ICLR 2025)*, 2025. URL https://openreview.net/forum?id=bmbRCRiNDu. arXiv:2410.09918.

Pulkit Verma, Ngoc La, Anthony Favier, Swaroop Mishra, and Julie A. Shah. Teaching LLMs to plan: Logical chain-of-thought instruction tuning for symbolic planning, 2025.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. ScienceWorld: Is your agent smarter than a 5th grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP 2022)*, pp. 11279–11298, 2022. URL https://aclanthology.org/2022.emnlp-main.775/.

Weiyun Wang, Zhangwei Gao, Lixin Gu, Hengjun Pu, Long Cui, Xingguang Wei, Zhaoyang Liu, Linglin Jing, Shenglong Ye, Jie Shao, et al. InternVL3.5: Advancing open-source multimodal models in versatility, reasoning, and efficiency, August 2025.

Yi Xu, Chengzu Li, Han Zhou, Xingchen Wan, Caiqi Zhang, Anna Korhonen, and Ivan Vulić. Visual planning: Let's think only with images. In *The Fourteenth International Conference on Learning Representations (ICLR 2026)*, 2026. arXiv:2505.11409.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, May 2025.

Mengjiao Yang, Dale Schuurmans, Pieter Abbeel, and Ofir Nachum. Chain of thought imitation with procedure cloning. In *Advances in Neural Information Processing Systems 35 (NeurIPS 2022)*, pp. 36366–36381, 2022. doi: 10.52202/068431-2635. arXiv:2205.10816.

Reproducibility Statement

Appendices A, B, and C specify the operation schemas, algorithm invariants, and runtime that check every execution step. Appendix E gives the trace corpora, splits, and training recipes, and Appendix F the statistical procedures. The transfer analysis is detailed in Appendix H.

Appendix L.1 specifies the node-choice setting, whose held-out panels, DAgger ablation, and zero-shot baseline are reported in Appendices L.3.1, L.3.3, and L.3.4. Table 17 holds the run ledgers, and Table 18 indexes the pinned evidence paths. Code and evidence bundles will be provided as anonymous supplementary material.

Appendix roadmap. The appendices first specify the interface and the data, then report the trace-learning results in detail, and then document the audits and the separate choice-frontier study. Appendices A, B, and C define the operations, invariants, and runtime, and show that the runtime never chooses or repairs on the policy's behalf. Appendix D states what each observation exposes, Appendix E gives the corpora, splits, and training recipes, and Appendix F states what an interval licenses. Appendices G to I report the trace-study results: the primary matrix, the secondary panels with transfer and the second backbone, and the failure mechanisms. Appendix J shows illustrative search episodes. Appendix K states what the identity audit of the enumeration contract shows about oracle-assisted controls, Appendix L reports the choice-frontier study, and Appendix M audits two published interfaces. Appendices N and O trace each run to its receipt and ledger, and Appendix Q compares the two decision contracts.

A Operation Schemas

This appendix gives the operation vocabulary, validity checks, and rejection kinds of each declared algorithm, for readers who need the exact interface behind the Typed Search Operation definition.

The schemas enforce one rule: every operand that determines exploration must come from the policy (Section 3). Each declared algorithm defines a vocabulary of operation types τ, and a policy emission $u_t = (\tau_t, \theta_t)$ pairs a type from that vocabulary with an operand vector θ_t carrying every exploration-determining argument. The runtime rejects any emission whose type falls outside the declared vocabulary or whose operand vector omits a required field, so it never supplies a missing operand itself.

BFS. The primary operation is node expansion. Its operands name the source state (the current frontier head), the grounded successors selected for insertion, the action producing each successor, and the enqueue position for each. The runtime checks that the named source matches the FIFO frontier head, that each successor results from an applicable grounded action in the source state, that the successor is unvisited, and that each insertion targets the FIFO tail. A goal-test operation names the state to test. The BFS pilot corpus exposes the facts these checks use, namely exact grounded successor candidates and full visited membership at each step.

BFWS (complete unpruned). Expansion operands carry the novelty value and goal-count priority of the source state alongside the fields above. Insertion operands carry each candidate's duplicate membership, novelty, and priority. The runtime checks that the selected source has the declared best priority, that duplicate membership is correctly reported, and that insertions follow the declared ordering. The BFWS corpus exposes the facts these checks use, namely candidate duplicate, novelty, priority, insertion, and enqueue facts, at every training observation.

Additive best-first. Operands carry the source state's accumulated path cost g, heuristic estimate h_{add}, and composite value $f = g + w \cdot h_{\mathrm{add}}$ with $w = 3$ for the weighted variant, or $f = h_{\mathrm{add}}$ for the greedy variant. Insertion operands carry per-successor g, h_{add}, and f. The runtime checks that the expanded state has the minimum f in the frontier under the declared tie-breaking rule and that all arithmetic is consistent.

Validity preconditions. Every emission passes through the same ordered sequence of checks, and the first check it fails rejects it with that kind and ends the episode. Each recorded rejection kind therefore names the earliest check an emission failed. The checks run in the following order.

1. **Parse.** The raw output must parse into a well-formed Typed Search Operation with a recognized type and complete operand vector. A failure is recorded as `malformed_output`.
2. **Source identity.** The named source state must match the state the algorithm expects. A failure is recorded as `source_state_mismatch`.
3. **Action applicability.** Every grounded action in the operands must be applicable in the source state. A failure is recorded as `inapplicable_grounded_action`.
4. **Successor schema.** Any named successor state must satisfy the domain schema. A failure is recorded as `schema`.
5. **Successor effects.** The named successor must match the declared effects of the applied action. A failure is recorded as `effect`.
6. **Static context.** Referenced predicates, objects, and constants must be consistent with the task instance. A failure is recorded as `static_context`.
7. **Frontier discipline.** The frontier manipulation must respect the declared ordering (FIFO for BFS, priority for BFWS, f-order for additive best-first). A failure is recorded as `invalid_frontier_operation`.
8. **Algorithm invariant.** Any remaining algorithm-specific constraint not covered above. A failure is recorded as `other_invariant_violation`.

Rejections fall into two categories, format failures and search failures. *Malformed-output* rejections never reach the operation checker, whereas *post-parse rejections* parse but fail a runtime check. The oracle-assisted random-valid control emits only runtime-accepted operations, so its single failure

Table 3: **Base failures are mostly malformed output, and learned failures, pooled over learned arms, are mostly post-parse invariant violations.** The 512-record BFWS adapters are the exception and fail on format (Appendix I). Arm labels give failures/episodes where both counts exist; shares use failures within each class.

Arm class	Rejection kind	Failures	Share
Base (447/447)	malformed_output	379	84.8%
	invalid_frontier_operation	67	15.0%
	other_invariant_violation	1	0.2%
Learned (407/906)	other_invariant_violation	229	56.3%
	malformed_output	72	17.7%
	inapplicable_grounded_action	56	13.8%
	effect	27	6.6%
	schema	16	3.9%
	invalid_frontier_operation	5	1.2%
	source_state_mismatch	1	0.2%
	static_context	1	0.2%
Oracle (51 failures)	expansion_budget_exhausted	51	100.0%

kind is `expansion_budget_exhausted`, recorded when the frozen episode budget runs out before a goal is reached.

Rejection counts. The base model fails mostly on format, while learned policies, pooled over learned arms, fail mostly after parsing, on algorithm invariants (Table 3). The pooled profile does not hold for every algorithm. All 72 learned `malformed_output` failures come from the 512-record BFWS adapters, which fail on format (Appendix I). The counts come from the failure-mechanism calibration, which classified all 905 failures across 2,910 replay-verified episodes with zero unclassified. The `effect`, `schema`, and `static_context` kinds arise in successor-prediction episodes, where the model generates a dynamic-state successor.

B Algorithm Invariant Definitions

This appendix states the invariant the runtime enforces for each declared algorithm, which Section 3 defines only in outline.

The Trusted Search Runtime checks the following invariants on every operation. Because each invariant constrains a single step, stepwise validity and invariant compliance do not establish termination, completeness, optimality, or episode success (Section 3).

BFS. BFS keeps its frontier as a FIFO queue, so every expansion must select the queue head as its source and every insertion must target the queue tail. A state already in the visited set must not be re-inserted. This invariant is sufficient for completeness in finite state spaces with unit-cost transitions.

BFWS (complete unpruned). The frontier is a priority queue ordered first by novelty rank (lower is better), then by goal count among states with equal novelty. No generated successor is pruned. A successor whose state already appears in the frontier or visited set is not re-inserted.

Additive best-first, weighted ($w = 3$). The frontier is a priority queue ordered by $f(s) = g(s) + 3 \cdot h_{\mathrm{add}}(s)$, where $g(s)$ is accumulated path cost and $h_{\mathrm{add}}(s)$ is the additive heuristic. Among equal-f states the lower generation serial is expanded first. The expanded state is the heap head. A successor with lower g than an existing frontier entry for the same state reopens that entry with improved cost.

Additive best-first, greedy. The frontier is ordered by $h_{\mathrm{add}}(s)$ alone, with no path-cost component, then by ascending generation serial among ties. The expanded state is the heap head. Reopen and duplicate rules follow the weighted variant. The executed controller's serial assignment is what makes the enumeration contract insensitive to submission order. In both

additive settings, `BestFirstController.start_expansion` assigns serials before submissions, in the deterministic `PDDLStateAuthority.applicable_actions` order (sorted by `GroundedAction.serialize`). The `BestFirstController.apply_operation` heap orders entries by (priority, serial, state identifier, path cost), and `finish_expansion` requires the full candidate set. Submission permutations therefore leave frontier order unchanged on the enumeration contract, so the identity is a property of this runtime rather than of the enumeration contract in general.

C TRUSTED SEARCH RUNTIME

This appendix gives the runtime's guarantees and its step and episode pseudocode, for readers checking that the runtime never chooses or repairs on the policy's behalf.

The runtime provides three guarantees that hold across all declared algorithms.

1. **No-repair property.** The runtime validates, applies, persists to Search Memory, or rejects each emission. It never chooses omitted operands, reorders candidates, silently repairs an invalid emission, or substitutes a default search decision.
2. **Reject-and-terminate.** On the first rejected emission the episode terminates with the rejection kind recorded. There is no retry path.
3. **Replay determinism.** Replaying the raw emissions $u_{1:T}$ against the same task instance and the same initial Search Memory reproduces the explored trace. Any step whose outcome depends on runtime repair would violate this property.

The two procedures below make the guarantees concrete. STEP applies the checks of Appendix A in their fixed order, and EPISODE wraps it in a budgeted loop that ends at the first rejection or at the goal.

procedure STEP(u_t, M_{t-1}, $\mathcal{A}$, x)
  // Phase 1: Parse the raw emission
  if u_t does not parse as (τ, θ) with $\tau \in \text{vocab}(A)$ **then**
    return REJECT(`malformed_output`)
  // Phase 2: Validate source identity
  if θ.source $\neq \text{expected}(A, M_{t-1})$ **then**
    return REJECT(`source_state_mismatch`)
  // Phase 3: Validate operands against the task and state
  for each grounded action a named in θ **do**
    if a is not applicable in θ.source under $x.D$ **then**
      return REJECT(`inapplicable_grounded_action`)
  for each successor state s' named in θ **do**
    if s' violates the domain schema **then**
      return REJECT(`schema`)
    if $s' \neq \text{effects}(a, \theta.\text{source})$ **then**
      return REJECT(`effect`)
  if any static-context reference in θ is inconsistent with x **then**
    return REJECT(`static_context`)
  // Phase 4: Check Algorithm Invariant
  if frontier discipline of $\mathcal{A}$ is violated by (τ, θ, M_{t-1}) **then**
    return REJECT(`invalid_frontier_operation`)
  if any other invariant of $\mathcal{A}$ is violated **then**
    return REJECT(`other_invariant_violation`)
  // Phase 5: Apply and persist
  $M_t \leftarrow \text{apply}(\tau, \theta, M_{t-1})$
  return ACCEPT(M_t)

procedure EPISODE(x, $\mathcal{A}$, ϕ_m, B)
  $M_0 \leftarrow \text{InitSearchMemory}(A, x)$
  for $t = 1, 2, \ldots, B$ **do**
    $o_t \leftarrow \phi_m\big(c_t(x, M_{t-1})\big)$ *// build observation*
    $u_t \leftarrow \pi(o_t)$ *// policy emission*
    $r \leftarrow$ STEP(u_t, M_{t-1}, $\mathcal{A}$, x)

```
        if r is REJECT(kind) then
            return (t, FAIL, kind)
        M_t ← r.M
        if GoalTest(M_t, x.G) = TRUE then
            return (t, SUCCESS)
    return (B, FAIL, expansion_budget_exhausted)
```

Each call to STEP consumes one unit of the frozen episode budget B whether the emission is accepted or rejected, so invalid operations are charged to the budget (Section 4).

D MODALITY ADAPTERS AND OBSERVATION CONTRACT

This appendix defines the observation adapters and what each one exposes to the policy, which bounds any reading of the modality results.

Every adapter must be decision-sufficient: its observation must expose every fact the exact reference uses to resolve the current step under the declared algorithm. Three fixed Modality Observation adapters are declared. Each maps a decision context c_t (the canonical task context plus the decision-relevant view of Search Memory M_t) to an observation $o_t = \phi_m(c_t)$ presented to the policy, and the adapter stays fixed for an entire training or evaluation cell.

text-state. The state, goal, and search context are serialized as a compact relational representation. Object identities, predicate names, and all Search Memory contents (frontier, visited set, algorithm-specific state) appear as structured text.

visual-state. The state and goal are rendered as a pair of 128×128 pixel images, a rendered state image and a partial-goal constraint image. Object labels are not embedded in the rendered image. Search Memory contents and the grounded candidate list remain in the textual serialization, even in this adapter. This is the published frozen `visual-state` arm.

The figures show cached policy scenes, enlarged but never retouched. Panel (a) of Figure 1 uses the cached 128-pixel unlabelled scene of the initial ferry state that the policy received, with vector labels added beneath the image rather than drawn into it, and Figure 2 shows the same cached scene as a thumbnail. The trace figures in Appendix J enlarge cached 128-pixel unlabelled scenes of one blocksworld task by nearest-neighbour sampling, without re-rendering. The choice-frontier validation filmstrips in Appendix L.3.2 show label-free 256-pixel re-renders, used only after the same scenes at the policy's 128-pixel resolution matched the cached inputs pixel for pixel. The held-out puzzle shows the original cached 128-pixel frames instead, because re-rendering its initial state changes tile glyphs. Figure scripts retain the exact image paths and checks.

multimodal-state. The observation combines the text-state serialization and the visual-state rendered images. The policy receives the relational text and the image pair jointly.

E DATA AND SPLIT PROCEDURES

This appendix describes the corpora, splits, and evaluation manifests behind the reported panels, for readers checking that no evaluation task leaked into training.

Every training and evaluation split is by whole task instance, so no state, decision, or target from an evaluation task appears in any training row. Every released corpus also passed no-overlap, no-conflict, live/training-parity, and token-budget audits before use.

BFS pilot corpus. The BFS pilot corpus is split by semantic task identity, and its v8 evaluation panel draws only from the dev split, so no panel task contributes training rows. The v6 text-state process corpus holds 25,109 process rows split into 12,994 train and 12,115 dev rows, drawn from 90 replay-verified trusted traces covering 90 distinct tasks, with zero cross-split task overlap. The v8 evaluation panel selects its 15 tasks from the dev split. Corpus observations expose the canonical task context, exact grounded successor candidates, and full visited membership. Release audits recorded zero semantic, input, and input-target overlap, zero conflicting identical-input labels, zero

teacher-operation rejections, and zero live/corpus mismatches, all within an 8,192-token input cap and a 384-token output allowance.

BFWS corpus. The BFWS corpus covers the frozen 105-task panel, and its audit counters recorded zero overlap, mismatch, or rejection violations. It holds 69,019 replay-verified teacher decisions. Its release contains 69,019 process records, 67,215 operational records, and a process projection of 47,780 train and 21,239 dev rows. Each step's observation exposes candidate duplicate membership, novelty values, goal-count priorities, insertion positions, and enqueue facts.

Training recipes and budgets. Every adapter fine-tunes Qwen3-VL-8B-Instruct with LoRA, with the loss on the decision tokens only; the small-set grid pins the backbone at revision `0c351dd0`. The small-set execution adapters and the node-choice adapters use rank 64, α 128, dropout 0.05, learning rate 10^{-4} and batch size 32. Each small-set execution adapter trains for one epoch on 512 replay-verified records (16 optimizer updates, seed 17), and the larger BFS and BFWS runs train for three and two epochs on 12,994 and 47,780 records, so the recipes are not a matched-size comparison. Each node-choice adapter trains for 128 updates on 4,096 samples (2,048 records, each in two menu orders). Execution budgets count operations, including invalid ones, and the pretrained base is priced at one call because every base episode ends on its first, invalid operation. The program cap is 336 GPU-hours (Table 17). The larger visual execution runs predate the unlabelled scene recipe: they use an earlier 128-pixel Planimation render that keeps object names in 8 of the 12 main domains (15-puzzle, Blocksworld, Depot, Driverlog, Elevators, Ferry, Logistics, Towers of Hanoi) and in FreeCell, and its archived Sokoban initial frame is blank, so in Sokoban tasks those runs saw no informative initial scene.

Evaluation manifests. The BFS pilot uses a frozen 15-task outcome-blind panel. The BFWS structural gate froze a 105-task panel. The expanded baseline evaluates 24 tasks from 12 domains under the three declared observation types and four arms (1,152 logical bindings, 576 model episodes, all independently replayed).

F STATISTICS

This appendix states the interval, test, margin, and tiny-strata conventions behind every reported contrast, for readers judging what an interval licenses.

Default contrast procedure. Main-program intervals are descriptive bounds, not evidence of broad superiority, because they come from one training seed and panels of 24 or fewer instances. Contrasts are paired by whole task instance, and paired differences use a bootstrap with 10,000 resamples at seed 1729 and 95% percentile intervals.

The BFWS gate's reported lower bound is the 0.025 quantile, i.e., the lower endpoint of a two-sided 95% percentile interval. The gate follows the default procedure, with confidence level 0.95 and 10,000 paired whole-instance resamples (bootstrap seed 1729).

Tiny-strata rule. No intervals are computed over strata with fewer than 8 episodes, so such strata are reported as descriptive only. They are listed explicitly and never silently pooled or dropped.

Follow-up bootstrap conventions. The follow-up windows fix their own bootstrap seeds, separate from the main program's seed 1729 default. The choice-frontier ladder contrasts use a paired whole-problem bootstrap at seed 61813 with 10,000 resamples and 95% percentile intervals, and their adapter endpoints use a task-cluster bootstrap that draws tasks with replacement at seed 133 for 10,000 draws.

Materiality and non-equivalence. A follow-up contrast is material when its 95% interval excludes zero, and equivalence can be read only against a pre-registered margin. The choice-frontier protocols pre-register an equivalence margin of ± 0.05 for D (adapter minus random-valid M1) and for the DAgger change Δ, and none for the separation S from an exact-ϵ rung, so a non-separated S is not equivalence to the rung. Wherever the text reports a non-material contrast, an adjacent

Table 4: **Process SFT beats oracle-assisted random-valid on the BFWS panel.** Success over 15 tasks for one fixed training seed, alongside the pretrained base and the exact reference.

Condition	Success (15 tasks)
Pretrained base	0.0
Process SFT	0.933
Random-valid (oracle-assisted control)	0.467
Exact execution	1.0

Table 5: **At 512 records, the two run-ordered algorithms solve no task in any observation type.** Descriptive enumeration-contract baseline for BFS and BFWS by algorithm and modality. Successes, SFT invalid operations per decision, SFT expansions, and exact-reference decisions are separate columns. Random-valid (RV) and exact are oracle-assisted controls, not model abilities.

Algorithm	Modality	Base	SFT	RV	Exact	SFT inv/dec	SFT exp.	Exact dec.
BFS	text	0/24	0/24	15/24	24/24	24/66	15	1969
	visual	0/24	0/24	15/24	24/24	24/69	17	1969
	multimodal	0/24	0/24	15/24	24/24	24/63	15	1969
BFWS	text	0/24	0/24	17/24	24/24	24/24	0	974
	visual	0/24	0/24	17/24	24/24	24/24	0	974
	multimodal	0/24	0/24	17/24	24/24	24/24	0	974

sentence states that non-significance is not equivalence. The exact sign-flip p value is likewise descriptive. It enumerates all 2^n sign vectors of the n per-task differences, is two-sided, and tests a zero difference, so it never supports equivalence.

G Results Detail: Primary-Matrix Tables

On the expanded baseline, at 512 training records per cell, process SFT reaches no goal in any BFS or BFWS cell under any observation type (Table 5). The BFWS positive comes from a separate fixed-seed run on the frozen text-state BFWS panel with 47,780 training records, which beats random-valid with a paired bootstrap 95% lower bound of 0.2 (Table 4). The two tables keep what the model learned apart from what the controls achieve, and no measure is combined with another. Table 4 gives the BFWS panel, and Table 5 gives the expanded 24-task baseline by algorithm and modality. Section 5.1 summarizes both.

H Results Detail: Secondary Panels

This appendix reports the secondary enumeration-contract panels: the DAgger comparison, successor generation, the transfer detail behind Section 5.3, and the second backbone.

DAgger. DAgger (Ross et al., 2011) does not improve BFS execution over an exposure-matched continued-SFT control, and both stay far below random-valid (Table 6). The comparison ran two DAgger iterations, collecting corrections on 3 modality-task pairs and evaluating on 24 unseen ones. On the unseen panel, DAgger and continued SFT each achieve 1/72 invariant-valid successes, with one DAgger win, one loss, and 70 ties over whole problems. Random-valid reaches 45/72 and exact execution 72/72, while the original process-SFT checkpoint achieves 0/72. DAgger also has the higher unseen invalid-operation rate of the two iterated arms (0.316 versus 0.175). Final cumulative unique training corrections were text 201, visual 203, and multimodal 181, accumulated over 12 training cells of 512 records and 16 optimizer updates per cell-iteration.

Model-generated successors. The learned policy depends on the runtime's trusted successors. When the model generates dynamic-state successors itself, goal success collapses from 45/45 to 1/45 on the evaluated panel, and the single success is the one multimodal unseen cell. The trusted arm made zero trusted-state substitutions. The generated successors fail mostly on content rather than format. The trained successor model is exactly correct on 65/109 retained raw predictions, with 93 schema-valid but only 66 effect-valid, and downstream failure kinds are dominated by exact

Table 6: **DAgger does not improve BFS over exposure-matched continued SFT.** Success and unseen invalid-operation rate for the iterated arms, the original process-SFT checkpoint, and both references.

Arm	Dev success /9	Unseen success /72	Unseen invalid rate
DAgger iteration 2	3/9	1/72	0.316
Continued SFT iteration 2	2/9	1/72	0.175
Original process SFT	1/9	0/72	0.364
Random-valid	6/9	45/72	0.000
Exact reference	9/9	72/72	0.000

state-identity and effect failures (27 effect, 16 schema, 1 static-context). Collection-time verification of the untrained exact-successor accepted 119/1,536 predictions (text 43, visual 52, multimodal 24).

Transfer. The BFWS-trace adapters show the largest transfer gain, raising FOLIO accuracy from 0.600 to 0.645–0.660 under all three observations. The six BFS and BFWS adapters and the base were evaluated zero-shot on frozen subsets of FOLIO (Han et al., 2022) (200 items), GSM8K (Cobbe et al., 2021) (200), and HumanEval (Chen et al., 2021) (164), with per-adapter accuracies in Table 2. These are the 512-record small-set adapters, so the BFWS adapters that raise FOLIO solve no BFWS planning task at that budget (Table 5). Every adapter stays within 0.945 to 0.965 on GSM8K (base 0.955) and within 0.756 to 0.799 on HumanEval (base 0.780). A leakage screen over the frozen 8-gram window finds 7/564 benchmark items sharing only trivial numeric 8-grams with the planning corpus and zero content overlap.

Second backbone. A second backbone, InternVL3.5-8B (Wang et al., 2025), reproduces the small-set BFS result on a reduced-scope cell. The cell used a frozen 12-task key-cell panel selected by lowest reference decision count before any model outcome, with one seed-17 training run per modality cell, and neither arm reached a goal. The pretrained base scored 0/36 at 1 decision per episode (36 total), process SFT scored 0/36 at 1 to 4 decisions (59 total), and the reused BFS controls scored random-valid 54/72 (18/24 per modality, oracle-assisted) and exact 72/72. Coverage is complete, with 72/72 model episodes and 72/72 comparators replayed. InternVL3.5-8B shares the Qwen3-8B LLM family (Yang et al., 2025), so the comparison differs in the vision tower, connector, image tokenization, and multimodal recipe, and it is reported as an architecture-difference limitation.

I Results Detail: Failure-Mechanism Calibration

This appendix localizes where each arm class fails, for readers asking why learned success is absent under BFS and BFWS at the 512-record budget.

Each arm class fails in its own way. The base mostly cannot produce the output format, learned arms mostly emit well-formed operations that break the search invariant early, and the oracle control runs out of budget late. These profiles come from a CPU-only calibration, frozen before any outcome was computed, that mined 2,910 unique replay-verified episodes across the five executed model branches and classified all 905 failures with zero unclassified. The analysis is descriptive. The base policy fails in 447 of 447 episodes, always on the first decision. Of these, 379 emit a malformed output that never reaches the operation checker, and 67 issue rejected BFS frontier operations. Learned arms fail in 407 of 906 episodes. Their dominant failure kind is `other_invariant_violation` with 229 occurrences (56.3% of learned failures), namely parsed, source-correct, applicable operations that the runtime rejects because they violate the declared search invariant. The remaining learned failures are 72 malformed outputs, 56 inapplicable grounded actions, 27 effect failures, 16 schema failures, and smaller kinds. First failures cluster early, at decision indices 0, 1, and 2 with counts 137, 88, and 46. The oracle control fails 51 times, all late and all by expansion-budget exhaustion (buckets 5–8, 9–16, 17–32, and 33+ hold 9, 9, 3, and 30). Table 3 gives the rejection taxonomy.

The failures concentrate where the primary matrix shows no learned success (Table 5). Every baseline process-SFT episode under BFS or BFWS fails (144/144), and the DAgger-iteration BFS arms fail 155/162. Process supervision thus leaves mainly an invariant problem rather than a format problem, since most learned failures are rejected invariant violations and malformed outputs are a

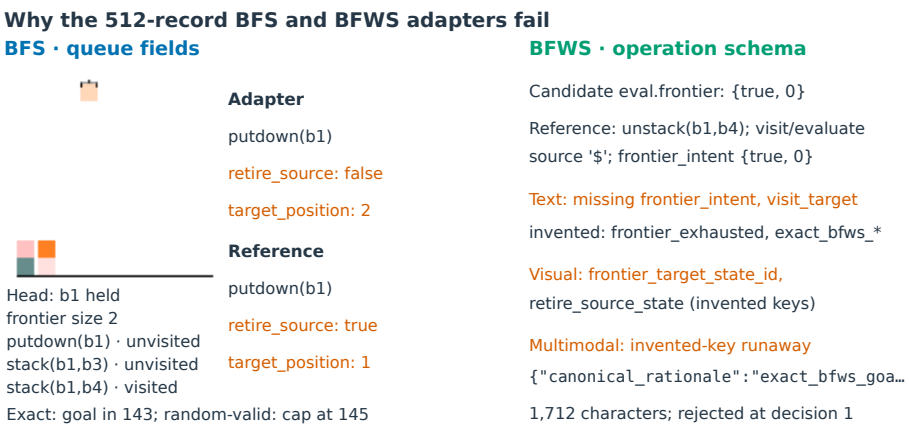


Figure 5: **At 512 records, BFS fails on queue bookkeeping and BFWS on the operation format.** Left: the BFS adapter names a valid successor but misstates whether the head is retired and the queue position (differences in orange); all three observation types give this error. Right: the BFWS adapter names the reference action but writes a malformed operation.

minority. Read with the horizon profile, the learned policy fails early on invariant grounds, whereas the oracle control, which always emits valid operations, fails only when its expansion budget runs out late in the episode. Like the rest of the calibration, this reading is descriptive.

At 512 records the two run-ordered algorithms fail in different ways, which the pooled profile hides. Under BFS the adapters emit well-formed search steps whose queue bookkeeping is wrong. Of the 72 main-grid BFS failures, 68 are successor operations with a wrong retire or position field and 4 retire the frontier prematurely. Under BFWS every one of the 72 main-grid failures is a malformed output at decision 0, so the adapter never emits a well-formed first operation. The BFS adapters have therefore learned the output format but not the queue discipline, whereas the BFWS adapters have not yet learned the format at this budget.

J Illustrative Search Episodes

This appendix shows one enumeration-contract episode whose scene comes from a blocksworld task. Figure 5 contrasts how the 512-record BFS and BFWS adapters fail: the BFS adapter emits well-formed steps with wrong queue fields, whereas the BFWS adapter fails on output format at its first decision (Appendix I). The episode is an illustration, selected without a pre-registered rule, and it does not estimate a task-level effect.

K IDENTITY AUDIT OF THE ENUMERATION CONTRACT

The identity audit compares random-valid with the exact reference decision by decision on the same tasks and seeds. Random-valid is an oracle-assisted control: at each decision it draws one of the runtime's listed valid operations uniformly at random, with a fixed seed and no retry or repair. Where its decisions never diverge from the reference's, its success measures operation validity, and it serves as a validity bound rather than as a competitor. On the enumeration contract the audit is a diagnostic, and on the choice-frontier contract it is a pre-registered pass condition (Appendix L).

In the enumeration contract's two additive cells, random-valid decides exactly as the exact reference on our runtime, on 48/48 task–algorithm pairs. The executed serial rule causes this. Each expansion submits its complete candidate set, and equal-priority ties break in sorted candidate order, so the runtime's priority heap fixes the order of expansion and the order of submission cannot change it. Random-valid therefore saturates the additive cells by construction, and success there measures whether the policy emits valid operations. The identity belongs to this controller rather than to additive best-first search in general (Appendix B), and the contract specification alone does not reveal it. In the BFS and BFWS cells random-valid's decisions differ from the reference's, and its success falls below the reference's.

L FRONTIER CHOICE FROM RENDERED SCENES

Under the enumeration contract the runtime's priority heap decides which state to expand next, so the trained policy never chooses among frontier states (Appendix K). This appendix reports a separate study of that choice. Under the choice-frontier contract the policy selects, from rendered scenes alone, which runtime-owned frontier state to expand next, and the runtime performs the expansion. Random-valid no longer saturates this contract, so a success measure has to be shown to rank choices correctly before any policy is scored against it. Box 1 gives the four-step procedure we used, and the subsections follow its steps. They cover the contract (Appendix L.1), the validation ladder (Appendix L.2), and the scaled adapter with its held-out panels (Appendix L.3). Appendix M applies steps 1 and 2 to published interfaces.

Under this contract random-valid chooses uniformly among the offered frontier states. Like the exact reference, it is an oracle-assisted control whose score is a bound, not an ability. Results come from freshly frozen and new-seed panels, each kind further removed from our design decisions. The ladder test and the adapter use the validation panel, 12 fresh tasks from 7 domains frozen before evaluation. The held-out panel (P2 in figures and tables) applies the same rules to new seeds on which no adapter had been evaluated, and their range was extended once before any evaluation episode (Appendix L.3.1). Held-out means this panel only.

The validation and held-out panels pass a random-valid screen that admits a task only if 1 to 9 of its 10 random-valid episodes reach the goal, which keeps tasks on which near-uniform choice sometimes succeeds. The unscreened panel (P2u), frozen with the held-out panel before evaluation, removes this selection by drawing kept candidates outside the held-out panel without the screen.

Each choice-frontier verdict follows a rule frozen before its outcome. The identity audit passes on any divergence over the frozen task–algorithm pairs, without re-runs. The ladder uses the pass rule in step 3 of Box 1. The adapter rule requires a positive difference from random-valid in M1, with a materiality margin of 0.05 and an equivalence margin of ± 0.05. Later frozen protocols add three training seeds, two fresh panels, a zero-shot policy, and a second external audit.

L.1 CONTRACT AND COMPARATOR SELECTORS

Under the choice-frontier contract the policy selects which runtime-owned frontier state to expand next, under a budget of twice the exact reference's expansion count. The contract runs only the two additive best-first variants. Its observation is scene-only. The policy sees rendered scenes of the initial state and of each frontier state, plus goal pages, and opaque labels are the only candidate text. No observation modality is therefore compared under this contract, and its results are not comparable with enumeration-contract results (Table 19). The paragraphs below specify the menu, output, budget, corpus derivation, metrics, and optimal costs.

Box 1: Validate before score. Four steps admit random-valid as a choice control and validate the success measure before any policy is scored.

Four-step validation

Inputs. The executed interface (per-decision candidate set, ordering, tie-break rule), the exact reference, and seeded random-valid with no retry or repair.

1. Identity audit. Compare random-valid with the exact reference decision by decision on the same tasks and seeds, reporting the divergent fraction and first divergence. Read success as choice only where decisions diverge and random-valid's success falls below the reference's. Otherwise success measures validity, which saturation alone cannot reveal (Appendix K).

2. Budget check. If a zero-token control exists, follow the compute-matched-baseline principle (Li & Talwalkar, 2020). Count the budget in the units that the control spends, decisions or expansions, and report its success at the reference's count and at twice it. The check fails if the control saturates in its native unit but not in decisions (Appendix M).

3. Ladder. If the reference computes a priority order, freeze a fresh panel before evaluation and pre-register the order exact reference, ϵ 0.25, 0.50, 0.75, random-valid. Each exact-ϵ rung replaces a fraction ϵ of reference decisions with random-valid choices. Score M1, the area under the solve-versus-budget curve up to twice the reference's expansion count. The ladder passes only if all four adjacent paired task-clustered 95% lower bounds are positive along the ϵ axis. The rungs are privileged, so a pass validates the measurement, not a policy (Appendix L.2).

4. Report. Give the policy's difference from random-valid with a paired task-clustered interval and a pre-registered margin (0.05 here), its ladder position, per-task signs, screened and unscreened panels, and training seeds. Non-separation from a rung is not equivalence (Appendix L.3).

Where a step was at risk (one audited instance each). Complete-set submission with a priority heap and sorted tie-breaking failed step 1 (our enumeration contract's additive cells, 48/48). Per-step choice among valid actions passed step 1 (ScienceWorld). Model-scored candidates on a frontier that keeps every alternative, under a token budget, failed step 2 by construction (LLM-First Search).

If a step fails, change the contract (for example, let the policy choose the frontier state, or break ties in submission order) and audit again.

Menu and observation. The choice-frontier observation withholds every score, position cue, and search counter, so a choice can rest only on the rendered scenes and the task pages. At each decision the runtime exposes the live frontier as an unscored menu of opaque labels $c_0, \ldots, c_K$. The labels are assigned after a seeded Fisher–Yates permutation of the generation-serial member order, with a per-decision seed derived from master seed 51131 and a hash of the member state identifiers, so a label carries no positional information. The observation holds static task-context pages, the unlabelled 128px initial-state scene, one unlabelled 128px scene per frontier state in menu order, and the partial-goal pages. Its text payload is limited to the algorithm name, the label list, the representation tag, the schema version, and the frozen legend, and a leak check fails closed on any forbidden key marker.

Output and validation. The output contract admits exactly one offered label per decision, and the first invalid emission ends the episode. The policy emits canonical JSON with the single key `expand_choice` and a label value. The runtime rejects malformed JSON, a wrong key set, a non-string label, a label not offered by the pending menu, and a label bound to a non-live member, and it records the rejection as `deterministic_invalid_operation`. An episode otherwise ends as `goal_reached`, `frontier_exhausted`, `decision_budget_exhausted`, or `expansion_budget_exhausted`.

Expansion and budget. The policy only chooses and the runtime does everything else, so one submitted choice is one decision. On acceptance the runtime removes the chosen member from the frontier and expands it atomically. Candidate generation, serial assignment, admission, duplicate detection, reopening, and expansion bookkeeping are inherited unchanged from the frozen additive best-first controller, which is not modified, and per-candidate admissions stay runtime-internal. Selecting a member that satisfies the goal solves the task without an expansion and without budget

charge. The exact episode therefore takes its reference expansion count plus one decisions. Decision and expansion caps both equal $2\times$ the reference expansion count.

Corpus derivation. Each training record teaches the exact reference's own choice, the label bound to the heap head. The training corpus was derived on CPU by replaying the frozen exact traces through the new contract and verified expansion-for-expansion against them (58 episodes bit-exact). Records are selected through the frozen matched-modalities membership, which indexes enumeration-contract decisions, and each such decision maps to the expansion whose candidate set it submitted into. Unique (task, expansion) pairs in first-occurrence order form the base set, and a round-robin over the frozen task order tops the set up to 512 training records per algorithm. Diagnostics are the unique expansions of the frozen diagnostic identifiers, with no top-up.

Training ran only after five fail-closed gates passed. The gates check expansion-for-expansion agreement with the stored exact traces, a token budget of measured input plus 384 within 32,768, the text-leak check, image resolution, and identity with the frozen membership. The pinned preparation report records outcome PASS over 29 tasks, 1,046 records (512 training and 11 diagnostic per algorithm), 3,544 scenes, and 58 episodes re-derived bit-exact. The smoke gate accepted 40/40 model calls.

Training and replay. Both adapters trained to completion under the frozen recipe, and their episodes are independently replayed. Each of the two cells trains a LoRA adapter (rank 64, α 128) for 16 updates on its 512 records at seed 17 with the frozen expanded-program recipe. Both jobs completed all 16 steps and saved final adapters, and only the later post-save summary construction failed. Their reports were therefore recovered from the primary training-state artifacts with provenance marked, no retraining was needed, and the training audit passes on the recovered reports. At finalize, independent replay re-derives every menu binding, payload, and trusted transition from the stored events and checks pixel bindings against the retained scenes.

Optimal costs. Each panel's C^* values come from one offline shortest-plan computation per task with replay-verified plans, and the native reference costs of all four declared algorithms are recorded separately.

Metric computation. The protocol fixes each metric's edge cases so that every arm is scored by the same rule. Under M1 an episode solves at multiplier m when its first goal selection occurs within $\lfloor mR_t \rfloor$ decisions with at most $\lfloor mR_t \rfloor$ preceding expansions, where R_t is the additive exact-reference expansion count. Unsolved $2\times$ runs count as failures at every m, the same prefix truncation applies to every arm, and task and algorithm weights are equal. M2 also has a censored form, in which failures are right-censored at the budget rather than assigned a fictitious expansion count. M3 counts transitions along the selected goal state's trusted predecessor chain and is undefined, never imputed, for unsolved episodes. The goal-at-selection rule fixes the reference's own M1. Goal selection is the $(R_t{+}1)$-th decision, so the exact reference fails at $m = 1$ and scores M1 $= 0.875$ by construction. An arm that reaches the goal in fewer decisions can therefore exceed the reference.

L.2 Validation Ladder

On the choice-frontier contract, M1 passes its pre-registered validity test. On the fresh validation panel it orders the exact reference, the three exact-ϵ rungs, and random-valid as pre-registered, with every adjacent paired task-clustered 95% lower bound positive (Figures 6 and 7). The reference's M1 of 0.875 is fixed by construction, because selecting the goal takes one decision beyond its expansion count (Appendix L.1). The ladder also passes on the held-out panel, frozen before any evaluation episode, so the ordering holds on tasks from seeds no adapter had been evaluated on. It passes descriptively on the unscreened panel as well (Appendix L.3.1).

A pass licenses one reading. Along the ladder's ϵ axis, a higher M1 means choices closer to the reference's (step 3 of Box 1). It does not validate M1 off this axis, because a privileged selector that reaches the goal in fewer decisions scores above the reference (Appendix L.1). The rungs read the reference's own priority order, which no policy sees, so the ladder validates the measurement, not a policy. Table 8 reports random-valid's success at M1's cap of twice the reference's expansion count.

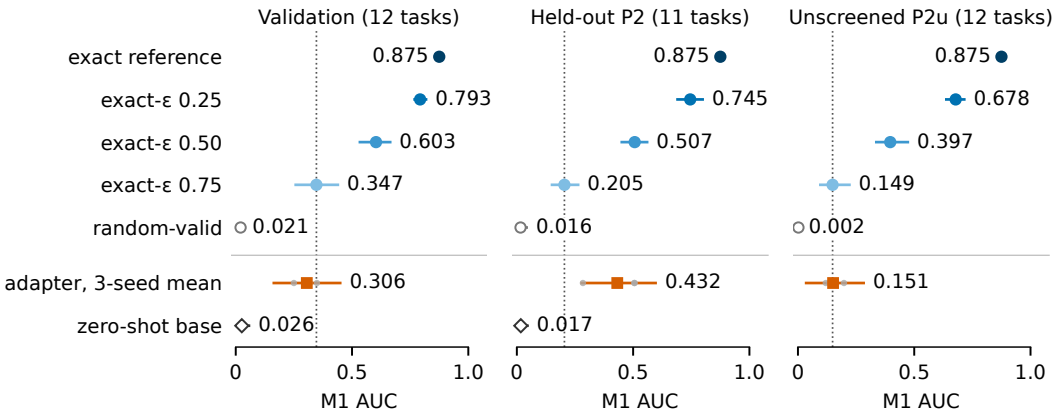


Figure 6: **M1 orders the privileged rungs on all three panels.** Points give M1, the area under the solve-versus-budget curve, with task-clustered 95% intervals, and faint points give the adapter's three seeds. Only on the held-out panel (P2) does the adapter's interval lie entirely right of the dotted ϵ=0.75 line. The rungs read privileged information, so the ladder validates the measurement, not a policy. The unscreened ladder (P2u) is descriptive.

Table 7: **Frozen validation panel membership.** Random-valid reached the goal in some but not all screening episodes on every task. R gives uncapped exact-reference expansions (greedy / w3), and Screen gives random-valid goals in the screening episodes.

Task	C^*	R	Screen
15puzzle-935006	8	46 / 12	1
blocksworld-935001	10	26 / 26	4
blocksworld-935004	10	13 / 13	2
depot-935005	10	14 / 15	1
depot-935011	9	13 / 13	1
elevators-935000	8	11 / 11	2
elevators-935013	8	11 / 11	2
grid-935016	8	9 / 9	4
storage-935000	10	20 / 17	1
storage-935001	8	18 / 15	2
towers_of_hanoi-935000	13	24 / 25	1
towers_of_hanoi-935002	15	24 / 26	2

The rest of this subsection shows how the validation panel was frozen and gives every arm's receipts on it.

The validation panel was fixed by rules committed in advance, the ladder protocol before task generation and the panel membership before evaluation. Of 240 generated candidates (12 snapshots, seeds 935000 to 935019), 127 met the cost filter ($8 \leq C^* \leq 20$), and 34 passed controls-only screening, which admits tasks on which random-valid is discriminative. The freeze rule sorts admitted tasks by domain and seed, keeps at most two per domain, and takes the first 12 (Table 7).

The ladder test passes on the validation panel because every adjacent gap has a positive lower bound. The pre-registered rule requires all four adjacent paired task-cluster lower bounds to be positive, with arms averaged over seeds within each task and algorithm, then over algorithms. The gaps widen toward the random end of the ladder: $+0.082$ $[+0.051, +0.111]$ for the exact reference minus the 0.25 rung, $+0.190$ $[+0.126, +0.252]$ for 0.25 minus 0.50, $+0.256$ $[+0.184, +0.339]$ for 0.50 minus 0.75, and $+0.326$ $[+0.233, +0.422]$ for 0.75 minus random-valid. All 696/696 CPU episodes and the 48 model episodes were independently replayed.

Three documented deviations affect how the panel was generated and rendered, not which tasks it holds. First, generation reuses the frozen candidate generator, which reproduces all 12 snapshots byte-for-byte although domain generators take the seed differently. Second, a first concurrent generation run corrupted the blocksworld outputs through a shared temporary file. That run was discarded, all candidates were regenerated serially and reproduced, and the deviation was disclosed before evaluation. Third, only the two additive native references seed the pre-rendered views, rather than four

Table 8: **Validation-panel arms.** M1 falls in order from the exact reference down the ϵ rungs to random-valid. M1 has task-clustered intervals, and M2 and M3 are solved-only.

Arm	M1 [95% CI]	2×	M2 ρ	M3 κ
Exact reference	0.875 [0.875, 0.875]	1	1.0	1.021
Exact-ϵ 0.25	0.793 [0.764, 0.824]	0.992	1.089	1.024
Exact-ϵ 0.50	0.603 [0.528, 0.670]	0.95	1.246	1.032
Exact-ϵ 0.75	0.347 [0.252, 0.445]	0.717	1.374	1.023
Random-valid	0.021 [0.007, 0.038]	0.083	1.692	1.042
Scaled adapter (seed 17)	0.349 [0.172, 0.531]	0.583	1.326	1.140
Pretrained base	0.000 [0.000, 0.000]	0	undefined	undefined

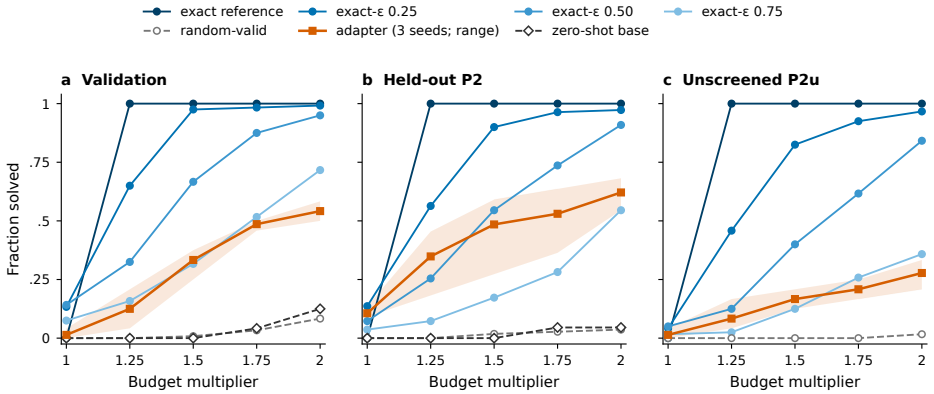


Figure 7: **Solved fraction against decision budget.** On all three panels the privileged rungs fall in order between the reference and random-valid, and the trained adapter's area exceeds random-valid's. Panels are validation, held-out (P2), and unscreened (P2u, descriptive), and the zero-shot base appears where measured. The adapter band is its three-seed range, not a confidence interval. Areas under these curves are M1, and the privileged rungs test only their own reference–random axis.

algorithms, because native BFS exceeds the 128-expansion reference ceiling on these tasks, and unseen states are rendered live.

L.3 Scaled Adapter

The validated measurement places the trained policy above random-valid and leaves the untrained base at random-valid's level (step 4 of Box 1). The trained adapter's D_3, the three-seed mean of its M1 minus random-valid's, is $+0.285$ on the validation panel and $+0.416$ on the held-out panel, and both intervals exclude zero (Table 9). On the validation panel every seed is positive, as the pre-registered rule requires.

The adapter's ladder position depends on the panel. It is separated above the 0.75 rung only on the held-out panel. On the other two panels and pooled, its difference from that rung is not separated from zero, which is not equivalence, because no equivalence margin was pre-registered for this contrast (Table 9). The held-out separation rests on 11 task clusters.

One other policy scores at the random-valid level, which shows that on these panels valid choices alone do not produce a gain. The pretrained base, prompted zero-shot under the same contract, observation, and cap, makes only valid choices. It is nonetheless equivalent to random-valid on the validation panel within the pre-registered ± 0.05 margin (one realisation), and its confirmatory held-out verdict is inconclusive. On both panels it falls below the 0.75 rung and below the adapter (Appendix L.3.4).

Without the screen, the gain survives but rests on 4 of 12 tasks. All 12 unscreened tasks happen to have no random-valid screening goals, and on the 8 tasks that do not carry the gain, neither the adapter nor random-valid solves (Table 9). This is descriptive evidence that the gain survives without the screen, not evidence about where choice discriminates.

Table 9: **The adapter beats random-valid on every panel but clears the 0.75 rung only on the held-out panel.** Three-seed adapter differences in M1 with paired task-clustered 95% intervals. Yellow rows are descriptive. ✓ denotes separation above zero and = denotes non-separation, not equivalence. The last column counts tasks with a positive, zero, or negative adapter-minus-random-valid difference (descriptive, not task-level tests).

Panel	Tasks	D_3 vs. random-valid		S vs. ϵ=0.75		Tasks $+/0/-$
Validation	12	$+0.285$ $[+0.140, +0.433]$	✓	-0.041 $[-0.187, +0.102]$	=	9/2/1
Held-out (P2)	11	$+0.416$ $[+0.250, +0.591]$	✓	$+0.227$ $[+0.053, +0.403]$	✓	9/2/0
Unscreened (P2u)	12	$+0.149$ $[+0.030, +0.283]$	✓	$+0.002$ $[-0.145, +0.157]$	=	4/8/0
Pooled	35	—		$+0.058$ $[-0.040, +0.156]$	=	22/12/1

Table 10: **Descriptive seed-17 per-task adapter gain over random-valid.** The gain is uneven across tasks, large on both storage and both elevators tasks and zero on both blocksworld tasks. The pre-registered three-seed endpoint is reported in Table 12.

Task	D	Task	D
15puzzle-935006	-0.0625	elevators-935013	0.6875
blocksworld-935001	0.0	grid-935016	0.3625
blocksworld-935004	0.0	storage-935000	0.7375
depot-935005	0.0625	storage-935001	0.7
depot-935011	0.7125	towers_of_hanoi-935000	0.2375
elevators-935000	0.5	towers_of_hanoi-935002	0.0

The gain is not carried by a single domain. Over all three panels, 22 of 35 per-task differences are positive and one is negative, and leaving out any one domain of the validation and held-out panels keeps every D_3 and its lower bound positive (descriptive, Figure 8, Table 13).

One on-policy DAgger round, run for one seed on half the planned corpus, does not separate the adapter from the 0.75 rung (Appendix L.3.3). Together these results support a choice-quality result relative to random-valid and the exact-ϵ ladder, on three panels with three training seeds. They carry no planning-ability or generality claim.

The rest of this subsection gives the adapter's frozen protocol, corpus, recipe, and per-seed results, followed by the held-out panels, illustrative episodes, the DAgger ablation, and the zero-shot policy.

Each part of the adapter protocol was frozen before the step it governs: the protocol before task generation, the corpus before training, and the schedule before launch. The endpoint D is the mean over training seeds of the adapter's M1 minus random-valid's M1, with a task-cluster interval (10,000 draws, seed 133). The verdict POSITIVE requires a positive lower bound, $D \geq 0.05$, and a positive estimate at every seed, and the equivalence margin is ± 0.05. Controls are reused from the ladder test.

The training corpus shares no task with any evaluation panel, and its menu augmentation places the target last no more often than chance. Of 2,400 serially generated training tasks, 1,182 remained eligible after excluding overlap with every evaluation panel and deduplicating. The seed-major walk reached 2048 records per algorithm from 236 tasks in 12 domains. Two reseeded menu orders per record give 8,192 samples, with the target last in 17.58% of samples against a chance rate of 17.58%.

The adapter changes the earlier small-corpus adapter's data, not its recipe. The recipe matches that adapter's except for corpus, augmentation, sample order, and seed (LoRA rank 64, alpha 128, 128 steps, seed 17). The smoke gate accepted 34/34 calls, and all 24/24 learned episodes, capped at $2R$, replayed with no mismatch and no invalid operation. For seed 17, the $2\times$ solve fraction is 0.583, M2 ρ 1.326, and M3 κ 1.140. Over 691 multi-option decisions heap-head agreement is 0.210 against chance 0.125, the last-label rate is 0.205, and the first-label rate is 0.090.

Four documented choices shaped this run, and only the first bears on what it can show. First, a scope edit to one training seed left seed variance unmeasured in that run. A later pre-registered replication closed the gap by training seeds 29 and 71 with the byte-identical recipe and the seed-17 sample order, so the seeds change only the LoRA initialisation and dropout. All 48 of its new episodes were independently replayed with no mismatch. Second, the step budget matches the earlier adapter's recipe. Third, the seed-major corpus walk resolves an ambiguity in the protocol text. Fourth, both

evaluation hooks first failed because the scheduler passes only the terminal path. After a fix that derives the attempt directory from that path, the audit re-ran on the same attempts and passed without affecting GPU work or episodes.

On the validation panel the adapter beats random-valid at every seed, but it sits near or below the exact-ϵ 0.75 rung. Per-seed D is $+0.328$ $[+0.155, +0.508]$ for seed 17, $+0.229$ $[+0.108, +0.354]$ for seed 29, and $+0.297$ $[+0.134, +0.466]$ for seed 71, so every lower bound is positive. Per-seed M1 is 0.349, 0.250, and 0.318, with a between-seed standard deviation of 0.051 (descriptive). The pre-registered difference from the 0.75 rung is $S = +0.002$, -0.097, and -0.029 per seed. Table 12 sets these values beside the held-out and unscreened panels.

The adapter also picks the last menu label less often than the earlier small-corpus adapter did. Its last-label rate per seed is 0.205, 0.192, and 0.171 (chance about 0.13, descriptive), against 0.300 (chance 0.144) for that adapter on the same panel.

L.3.1 Held-Out Panels

This subsection shows that both fresh panels were frozen before any evaluation episode and gives every arm's results on them. The held-out panel appears as P2 in tables and the unscreened panel as P2u.

The held-out panel repeats the validation panel's construction on fresh seeds. Both fresh panels come from 2,400 candidates generated from seeds 955000–955199. Removing overlap with earlier panels (1,168 candidates), duplicates (205), goals at the initial state (31), and C^* out of range (536) left 460. The held-out panel then applies the validation panel's generation, C^*, and screening rules, keeping a candidate when $8 \leq C^* \leq 20$ and random-valid reaches the goal in 1 to 9 of its 10 screening episodes. The freeze walk, sorted by domain and seed, takes at most 2 tasks per domain up to 12 tasks, with a minimum of 8 tasks from 5 domains. Only 11 admitted tasks were available under the per-domain limit, so the panel froze with 11 tasks from 8 domains. That meets the minimum, so every held-out test is confirmatory.

The unscreened panel drops the random-valid screen, so it shows descriptively whether the gain survives without it. It takes kept candidates outside the held-out panel on which the exact reference reaches the goal under both algorithms, at most 2 per domain, without consulting the screen. This gives 12 tasks from 7 domains, and all 12 happen to have 0/10 random-valid goals in screening. All 638/638 held-out-panel and 696/696 unscreened-panel episodes were independently replayed, with none missing.

The seed range was extended once, before any evaluation episode, with every other rule unchanged. The first freeze, from the original 40-seed pool 955000–955039, admitted 6 tasks from 5 domains and took the protocol's blocked exit. At that point only controls-only screening had run. The amendment extended the seeds to 955000–955199, kept the 480 earlier candidates unchanged, and generated the new seeds 955040–955199 serially after them. The generation mechanism, the exclusion sources, the duplicate check, the C^* rule, R_t, the screen, the freeze walk, and the unscreened-panel rule all stayed fixed, and both panels were frozen from the combined pool.

The ladder test passes on the held-out panel and, descriptively, on the unscreened panel. On the held-out panel the four pre-registered adjacent gaps, from the exact reference down to random-valid, are $+0.130$ $[+0.070, +0.190]$, $+0.239$ $[+0.183, +0.290]$, $+0.302$ $[+0.259, +0.340]$, and $+0.189$ $[+0.136, +0.248]$, so every lower bound is positive. On the unscreened panel the descriptive gaps are $+0.197$ $[+0.154, +0.243]$, $+0.281$ $[+0.191, +0.355]$, $+0.248$ $[+0.209, +0.290]$, and $+0.147$ $[+0.088, +0.226]$, and every lower bound is again positive. Table 11 gives each arm's M1 on all three panels.

Every seed's gain over random-valid has a positive lower bound on every panel, but the rung separations need care. Table 12 gives D per seed and D_3 with intervals on each panel and pooled over 35 tasks, together with the separation S from the exact-ϵ rungs. Seed 29 has the lowest per-seed D on every panel, and on the held-out panel its separation point from the 0.75 rung is $+0.080$ (descriptive). No equivalence margin was pre-registered for S, so a non-separated S is not equivalence to the rung. The held-out separation rests on 11 task clusters, with 8 positive and 3 negative per-task differences.

Table 11: **Descriptive M1 by panel and arm.** On every panel the adapter's three-seed mean lies above random-valid and below the exact reference. Values have task-clustered 95% intervals, and the exact-ϵ rungs use privileged information the policy never sees.

Panel	Arm	M1 [95% CI]
Validation (12)	Exact reference	0.875 [0.875, 0.875]
	Exact-ϵ 0.25	0.793 [0.764, 0.824]
	Exact-ϵ 0.50	0.603 [0.528, 0.670]
	Exact-ϵ 0.75	0.347 [0.252, 0.445]
	Random-valid	0.021 [0.007, 0.038]
	Pretrained base	0.000 [0.000, 0.000]
	Adapter, seed 17	0.349 [0.172, 0.531]
	Adapter, seed 29	0.250 [0.125, 0.375]
	Adapter, seed 71	0.318 [0.151, 0.490]
	Adapter, three-seed mean	0.306 [0.158, 0.455]
P2 (11)	Exact reference	0.875 [0.875, 0.875]
	Exact-ϵ 0.25	0.745 [0.685, 0.805]
	Exact-ϵ 0.50	0.507 [0.445, 0.566]
	Exact-ϵ 0.75	0.205 [0.145, 0.269]
	Random-valid	0.016 [0.000, 0.048]
	Pretrained base	0.000 [0.000, 0.000]
	Adapter, seed 17	0.506 [0.313, 0.699]
	Adapter, seed 29	0.284 [0.119, 0.489]
	Adapter, seed 71	0.506 [0.295, 0.705]
	Adapter, three-seed mean	0.432 [0.273, 0.602]
P2u (12)	Exact reference	0.875 [0.875, 0.875]
	Exact-ϵ 0.25	0.678 [0.632, 0.721]
	Exact-ϵ 0.50	0.397 [0.331, 0.478]
	Exact-ϵ 0.75	0.149 [0.091, 0.227]
	Random-valid	0.002 [0.000, 0.006]
	Pretrained base	0.000 [0.000, 0.000]
	Adapter, seed 17	0.198 [0.031, 0.396]
	Adapter, seed 29	0.120 [0.026, 0.224]
	Adapter, seed 71	0.135 [0.031, 0.250]
	Adapter, three-seed mean	0.151 [0.030, 0.288]

Over the validation and held-out panels no single domain carries the gain, while on the unscreened panel it comes from four of twelve tasks. Table 13 counts tasks by the sign of their three-seed difference and leaves out one domain at a time over the validation and held-out panels. Over the 35-task union 22 tasks are positive, 12 are zero, and 1 is negative. On the unscreened panel the eight zero tasks are both 15-puzzle, both blocksworld, both depot, and both driverlog tasks, on which neither the adapter nor random-valid solves within $2R_t$. Every leave-one-domain-out D_3 lies from $+0.305$ to $+0.395$, and every lower bound is at least $+0.185$. All of these results are descriptive.

L.3.2 Illustrative Choice-Frontier Episodes

These rule-selected episodes illustrate what the adapter saw and selected on one solved validation task and one unsolved held-out task. On the validation task the adapter follows the reference on 7/9 comparable decisions and reaches the goal while random-valid exhausts its budget, and on the held-out puzzle every adapter seed exhausts the cap while exact search solves (Figures 9, 10, and 11). They are illustrations, not an efficacy estimate or a diagnosed failure mechanism, and reference heuristics are used only for analysis, never as policy input.

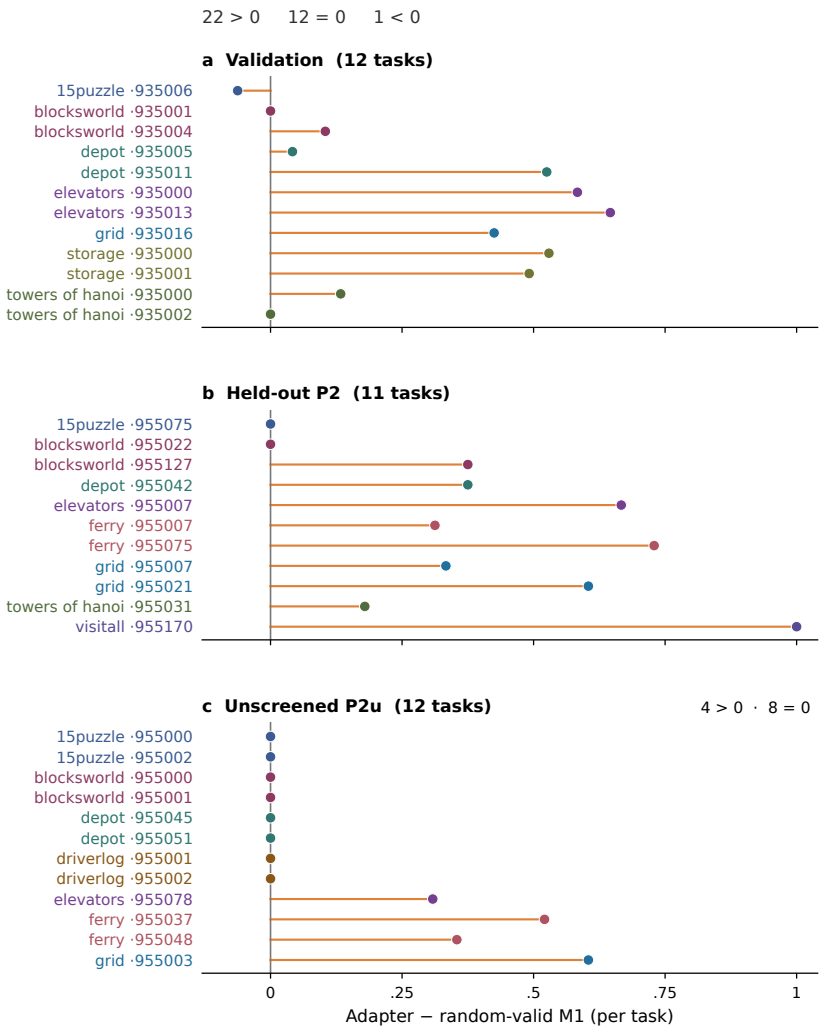


Figure 8: **Descriptive task-level choice-frontier gains.** Most tasks gain, and the unscreened panel's improvement is concentrated on four tasks. Each point is the three-seed adapter mean minus random-valid M1, averaged across the two additive algorithms for a task, and colours identify domains. These points are not task-level significance tests.

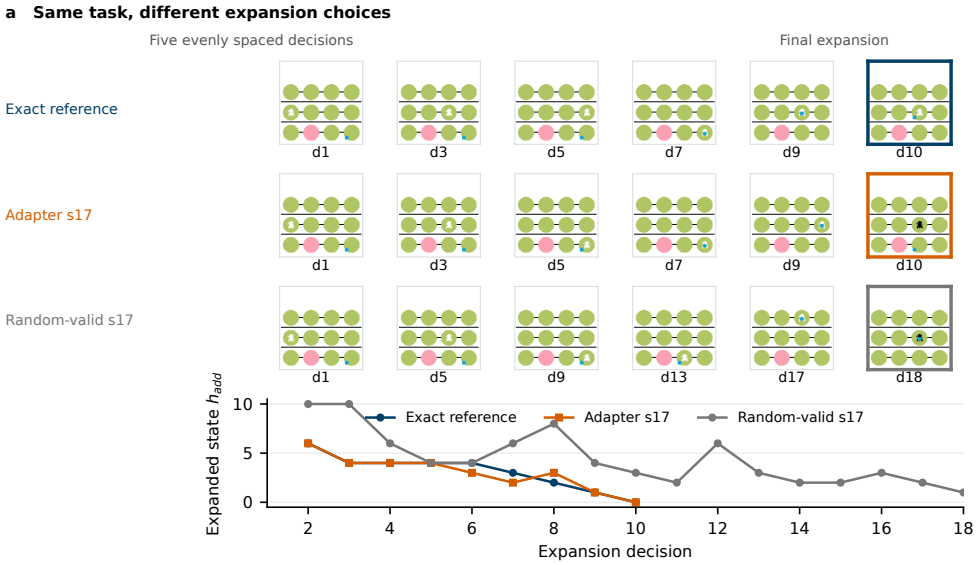


Figure 9: **On a solved validation task, the adapter follows the reference to the goal.** In this rule-selected weighted-A* episode, the seed-17 adapter agrees with the reference on 7/9 comparable decisions, and both reach the goal at decision 10, while random-valid exhausts its decision cap at 18. Rows show expanded scenes, and the plot traces their h_{add} after the first expansion. The example is illustrative, not an efficacy estimate.

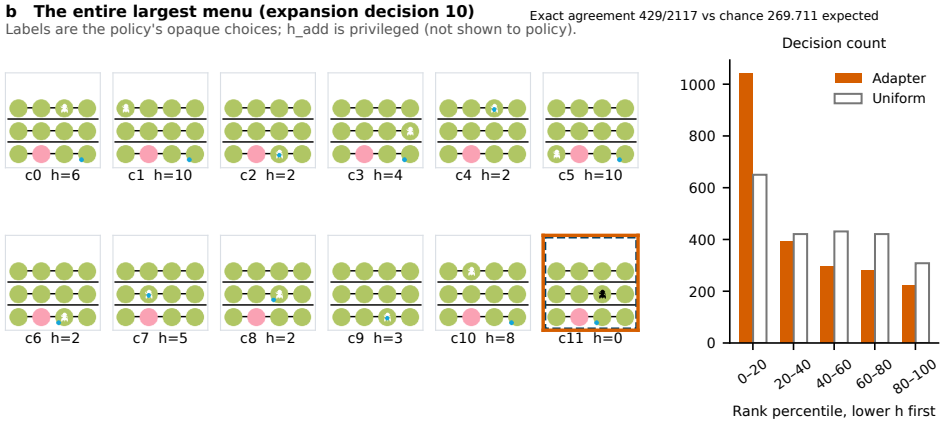


Figure 10: **The adapter picks the reference's state at the episode's largest menu and prefers low-h_{add} states overall.** At decision 10 of Figure 9, the adapter and the privileged reference both select c_{11} from 12 choices. The policy sees only opaque labels and scenes, not the heuristic values or the reference pick. The histogram describes the adapter's heuristic ranks across validation decisions against uniform choice on the same menus and is not an efficacy estimate.

Table 12: **Seed and rung contrasts on the three panels.** The adapter separates above random-valid on every panel, and above the 0.75 rung only on the held-out panel. Estimates are paired task-clustered 95% intervals for adapter minus random-valid (D) and minus a privileged ϵ rung (S), and yellow rows are descriptive. ✓ means separated above and = not separated.

Panel	Contrast	Estimate [95% CI]	Verdict
Validation (12)	D, seed 17	$+0.328$ $[+0.155, +0.508]$	
	D, seed 29	$+0.229$ $[+0.108, +0.354]$	
	D, seed 71	$+0.297$ $[+0.134, +0.466]$	
	D_3	$+0.285$ $[+0.140, +0.433]$	✓
	S vs. ϵ=0.75	-0.041 $[-0.187, +0.102]$	=
P2 (11)	D, seed 17	$+0.490$ $[+0.294, +0.688]$	
	D, seed 29	$+0.268$ $[+0.093, +0.477]$	
	D, seed 71	$+0.490$ $[+0.276, +0.694]$	
	D_3	$+0.416$ $[+0.250, +0.591]$	✓
	S vs. ϵ=0.75	$+0.227$ $[+0.053, +0.403]$	✓
P2u (12)	D, seed 17	$+0.196$ $[+0.031, +0.396]$	
	D, seed 29	$+0.118$ $[+0.026, +0.220]$	
	D, seed 71	$+0.133$ $[+0.031, +0.248]$	
	D_3	$+0.149$ $[+0.030, +0.283]$	✓
	S vs. ϵ=0.75	$+0.002$ $[-0.145, +0.157]$	=
Pooled (35)	D, seed 17	$+0.334$ $[+0.223, +0.450]$	
	D, seed 29	$+0.203$ $[+0.124, +0.290]$	
	D, seed 71	$+0.301$ $[+0.200, +0.411]$	
	D_3	$+0.279$ $[+0.191, +0.375]$	✓
	S vs. ϵ=0.75	$+0.058$ $[-0.040, +0.156]$	=

Table 13: **Descriptive concentration of three-seed gains.** Leaving out any one domain keeps D_3 positive with a positive lower bound. The upper rows count per-task signs, and the lower rows leave one domain out of the validation–P2 union.

Panel	Positive	Zero	Negative
Validation (12)	9	2	1
P2 (11)	9	2	0
P2u (12)	4	8	0
Union (35)	22	12	1
Domain left out	D_3	**95% interval**	**Tasks left**
15-puzzle	$+0.384$	$[+0.269, +0.501]$	21
blocksworld	$+0.395$	$[+0.269, +0.520]$	19
depot	$+0.353$	$[+0.224, +0.484]$	20
elevators	$+0.305$	$[+0.185, +0.430]$	20
ferry	$+0.331$	$[+0.212, +0.455]$	21
grid	$+0.331$	$[+0.204, +0.464]$	20
storage	$+0.332$	$[+0.209, +0.459]$	21
Tower of Hanoi	$+0.384$	$[+0.262, +0.507]$	20
visitall	$+0.318$	$[+0.211, +0.422]$	22

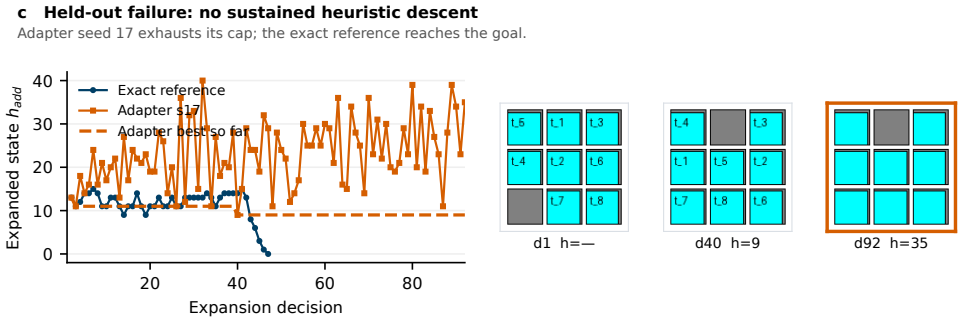


Figure 11: **On a held-out puzzle, the adapter stalls where the reference solves.** On this rule-selected puzzle, the reference reaches the goal at decision 47, while every adapter seed exhausts the cap at 92. The shown seed-17 adapter's best-so-far h_{add} stops improving at 9 after decision 40. The first cached scene lacks tile glyphs that later scenes show. The trace is illustrative and does not establish why the adapter fails.

L.3.3 DAgger Ablation

This subsection reports the pre-registered DAgger ablation, which finds no detectable change in the adapter's choice quality after one on-policy round at one seed.

The ablation asks whether one round of on-policy data lifts the adapter above the exact-ϵ 0.75 rung. It collects on-policy records with the seed-17 adapter under each additive algorithm, aggregates them with replay records from the frozen corpus, retrains, and evaluates on the validation and held-out panels. The primary endpoint is the paired separation S of the DAgger adapter from the 0.75 rung pooled over the 23 tasks, with SEPARATED_ABOVE requiring a positive lower bound. The co-primary is the change Δ in M1 from the pre-DAgger adapter, with equivalence margin ± 0.05.

The ablation ran at one seed on half the planned corpus, a reduction fixed before any DAgger training or evaluation outcome. Each algorithm's training set is the first 512 on-policy seed-17 records plus 512 replay records, which the menu augmentation turns into 2048 samples over 64 steps, with the recipe otherwise unchanged.

One DAgger round at one seed produces no detectable change in measured choice quality, so the ablation is inconclusive. The primary endpoint is NOT_SEPARATED, with $S = +0.115$ $[-0.026, +0.258]$ over 23 tasks. The co-primary is INCONCLUSIVE, with $\Delta = -0.030$ $[-0.114, +0.057]$ against the ± 0.05 margin. As descriptive secondary results, S is $+0.278$ $[+0.082, +0.473]$ on the held-out panel and -0.034 $[-0.197, +0.135]$ on the validation panel, and the last-label rate on the validation panel falls from 0.205 to 0.137. All 24/24 validation-panel and 22/22 held-out-panel DAgger episodes were independently replayed with no mismatch.

L.3.4 Zero-Shot Policy

This subsection shows that the untrained base model, prompted under the same contract, observation, and cap, is not separated from random-valid.

The zero-shot policy is the pretrained Qwen3-VL-8B-Instruct base with no adapter, prompted to choose by the goal. It receives the frozen scene-only choice-frontier observation and one appended system message, which states the JSON format and asks for the frontier state whose scene appears closest to satisfying the goal pages. The policy uses a fixed three-rule output extractor, greedy decoding at inference seed 17, and the adapter's cap of $2R_t$ calls. It ran on both additive algorithms over the validation and held-out panels, 46 episodes in all.

The zero-shot base is not separated from random-valid and falls below both the 0.75 rung and the trained adapter, as pre-registered (Table 14). Here D_3 names the pre-registered learned-minus-random-valid endpoint for both arms, a single realisation for the zero-shot arm. Held-out-panel verdicts are confirmatory. On the held-out panel the upper bound of D_3 exceeds the ± 0.05 margin by 0.001, so the frozen rule returns INCONCLUSIVE rather than EQUIVALENT. The arm's M1 is 0.026 [0.000, 0.063] on the validation panel, 0.017 [0.000, 0.051] on the held-out panel, and 0.022 [0.003, 0.046] pooled, below the exact-ϵ 0.75 rung on every panel. The pre-registered prediction, D_3 within the margin with S and the adapter contrast A separated below on both panels, held.

The base always emits a valid choice, but its choices agree with the reference's heap head at about the chance rate. All 1,592 model calls produced a valid choice through the first extractor rule. Heap-head agreement minus chance is $+0.003$ on the validation panel and $+0.001$ on the held-out panel, the last-label rate is 0.328 and 0.285 against a chance rate of about 0.13 to 0.14, and 4 of 46 episodes solve at $2R_t$ (descriptive). All 46/46 episodes were independently replayed with no mismatch.

M External Audits of Published Interfaces

The identity audit passes on ScienceWorld (Wang et al., 2022), a published sequential-choice interface, so it serves as a positive control for step 1 of Box 1. Random-valid diverges from the gold trajectory at the first decision of every audited pair and solves no task, against 0.915 for the reference, so success there measures choice. The audit covers 6 of ScienceWorld's 30 task types. None of the seven audited-candidate interfaces we surveyed uses an enumeration contract, so the 48/48 identity is shown only on our runtime (Table 15).

Table 14: **Zero-shot base contrasts by panel.** The base is not separated from random-valid and falls below both the 0.75 rung and the trained adapter on every panel. Estimates are paired task-clustered 95% intervals with descriptive exact sign-flip p values, and pooled rows are descriptive. $\equiv$ applies the pre-registered ± 0.05 margin, ? is inconclusive, ▼ separated below, and $|D_3| < 0.05$ is a descriptive point-only comparison.

Panel	Contrast	Estimate [95% CI]	Verdict	p
Validation (12)	D_3 vs. random-valid	$+0.005$ $[-0.025, +0.044]$	$\equiv$	0.906
	S vs. ϵ=0.75	-0.321 $[-0.422, -0.226]$	▼	4.88×10^{-4}
	A vs. adapter	-0.280 $[-0.425, -0.142]$	▼	0.004
P2 (11)	D_3 vs. random-valid	$+0.001$ $[-0.048, +0.051]$	?	1.000
	S vs. ϵ=0.75	-0.188 $[-0.239, -0.140]$	▼	9.77×10^{-4}
	A vs. adapter	-0.415 $[-0.589, -0.248]$	▼	0.004
Pooled (23)	D_3 vs. random-valid	$+0.003$ $[-0.024, +0.033]$	$\lvert D_3\rvert < 0.05$	0.896
	S vs. ϵ=0.75	-0.257 $[-0.324, -0.197]$	▼	2.38×10^{-7}
	A vs. adapter	-0.344 $[-0.459, -0.232]$	▼	7.63×10^{-6}

LLM-First Search (Herr et al., 2025) shows by construction why step 2 counts the budget in decisions. There the model scores candidates that a search then orders, and our audit replaces the paper's GPT-4o with a local open-weight substitute, Qwen3-30B-A3B-Instruct. Random-valid diverges from the reference by the third decision in every pair, yet under the paper's token caps it solves every task. A token cap never binds a chooser that spends no tokens, and because the frontier keeps every alternative, such a chooser is a complete randomized search. Counted in expansions, random-valid solves 0.0775 at the reference's own count and 0.19 at twice it (Appendix M.1).

The rest of this appendix shows how ScienceWorld was chosen from a survey of published decision interfaces, and the LLM-First Search audit follows as a subsection.

The survey found no published evaluation that uses our enumeration contract, so the external audits run on interfaces where the model picks or scores candidates. It admitted published evaluations from 2022 on in which the model chooses from a runtime-supplied candidate set inside a planning or search loop, with public tasks and code. Seven candidates passed, five of type B (the model picks one valid action per step), two of type C (the model scores candidates that a search orders), and none of type A, our enumeration contract (Table 15). The closest published analogue, LFG (Shah et al., 2023), has the model score candidates in the enumeration style, but it was screened out because no evaluation harness was released. Four candidates are not CPU-feasible under their papers' budgets. For WebShop (free-text search, no step limit) and LLM-First Search (an unreported token cap) this was a judgement call. LLM-First Search was the only candidate predicted to saturate, a prediction the later audit tested (Appendix M.1). ScienceWorld is the most cited feasible candidate, and the runner-up shares its prediction.

The surveyed interfaces differ in how they count budget, and only one counts a resource that random-valid does not spend. Six of the seven count budgets in steps, plan-length multiples, or iterations. Only LLM-First Search counts tokens, which a chooser that makes no model call never exhausts (Appendix M.1).

On ScienceWorld success measures choice, because the reference usually succeeds and random-valid never does. The frozen protocol takes the first 200 test variations under the paper's 100-step limit with five random-valid seeds (17, 5077, 6131, 7409, and 8527), and ten re-run episodes are byte-identical. The reference completes 183 of 200 (0.915), including 3/9 boil, 6/9 melt, 4/9 freeze, 6/9 change-state, 135/135 use-thermometer, and 29/29 measure-melting-point variations. It falls short only where 17 of the 36 Matter variations need more than 100 moves. Random-valid completes none, and 997 of its 1000 episodes end in a failure score.

The audit covers a narrow slice of ScienceWorld on a newer simulator version. The slice holds 6 of 30 task types (200 of 1,819 test variations), on all of which the paper's own Random-Valid score is 0.00. The paper reports normalized scores, so its DRRN score of 0.084 on these tasks is not a success rate. The audit uses ScienceWorld 1.3.0, whose notes warn that gold paths can change relative to 1.2.x, and the paper's simulator version was not re-run.

Table 15: **Surveyed decision interfaces and identity audits.** Both audited external interfaces register divergent choices, and random-valid matches the reference only on our runtime's enumeration contract, whose serial tie-break rule causes the identity. B selects an action, C scores search candidates, and A enumerates candidates. "div." means divergent and "sat." predicts saturation. = means identical choices, while ✓ marks an observed divergence.

Interface	Venue	Type	CPU	Pred.	Audit	Budget
AutoPlanBench (Stein et al., 2025)	ICAPS 2025	B	✓	div.	—	plan length ×
ScienceWorld (Wang et al., 2022)	EMNLP 2022	B	✓	div.	✓ 1,000/1,000	100 steps
GLAM (BabyAI-Text)	ICML 2023	B	✓	div.	—	steps (code)
WebShop	NeurIPS 2022	B	✗	div.	—	no step limit
RAP	EMNLP 2023	C	✗	div.	—	iterations
LLM-First Search (Herr et al., 2025)	arXiv 2025	C	✗[†]	sat.[†]	✓ 400/400	tokens
EmbodiedBench	ICML 2025	B	✗	div.	—	steps
Our enumeration (additive)	This paper	A	—	—	= 48/48	calls
Our choice frontier	This paper	—	—	—	✓ 18/18	$2R$ decisions

[†]Audited later with a GPU reference model; paper-budget saturation follows by construction (Appendix M.1). ✓/✗ in CPU denotes feasibility; "—" means no audit or no survey designation, not a negative result.

M.1 LLM-First Search Audit

A token budget cannot bind a chooser that spends no tokens. The LLM-First Search audit shows the consequence and why step 2 of Box 1 counts decisions instead.

LLM-First Search keeps every unexpanded alternative, and this makes a zero-token random chooser a complete search. It is a type-C interface. The model scores candidates, and the search expands the argmax child or pops the highest-valued frontier node, so every unexpanded alternative stays in the frontier. Random-valid answers every query with independent uniform draws, spends zero tokens, and makes no model call. It uses the five seeds of the first audit, and a decision is one node expansion after the root.

The audit runs the upstream code unchanged under the paper's own budgets and declares four deviations. The upstream code (commit `3025bdaa`, Apache-2.0) is imported unchanged and not vendored. The fixed rule audits every configuration whose published GPT-4o win rate exceeds 10%, which excludes Sudoku 6×6 and keeps Countdown 3, 5, and 7 and Sudoku 4×4, with games 0 to 19 of each, 80 tasks in all. Budgets are the paper's per-game token caps (1,000,000 for Countdown and 100,000 for Sudoku 4×4). The deviations are these. The paper's GPT-4o is replaced by the local Qwen3-30B-A3B-Instruct-2507, so reference success is that model's. The reference runs once per task. Sudoku 6×6 is excluded. Random-valid uses a faster frontier implementation that produced byte-identical logs on 57/57 probe episodes.

Random-valid diverges from the reference in every pair, and early. No pair of the 400 is identical, and the first divergence has median index 0 and maximum 2. All 80/80 reference and 400/400 random-valid episodes were independently replayed with no mismatch.

Under the paper's token budget random-valid nevertheless wins every episode, and this saturation follows by construction. Random-valid spends no tokens and the frontier keeps every alternative, so it is a complete randomized search, and a pre-audit probe had won 100/100 non-audited episodes. Its only way to lose was a 500,000-expansion safety guard, which never bound. Random-valid used a median of 201 expansions and a maximum of 102,606, against a median of 29.5 and a maximum of 1,300 for the reference. Random-valid wins at every seed, so the verdict is SATURATED, and every reference loss ends at the token cap (Table 16).

A budget counted in decisions lets success register choice again. Within the reference's own number of expansions per task, random-valid wins 0.0775 (per seed 0.0625, 0.075, 0.075, 0.0875, and 0.0875), so the verdict is CHOICE_REGISTERED. Doubling that budget raises random-valid's success only to 0.19. The reference's matched-budget success equals its token-budget success by con-

Table 16: **Descriptive LLM-First Search budget audit.** Random-valid saturates under the paper's token budget but falls far below the reference under a matched decision budget. Differences are substituted local reference minus random-valid success with task-cluster 95% intervals. The matched-budget reference success equals its paper-budget success by construction, and ✓ denotes registered choice and = saturation.

Configuration	Reference	Random-valid	Difference [95% CI]
Paper token budget (=, saturated)			
Countdown 3 (20)	1.000	1.000	$0.000\ [0.000, 0.000]$
Countdown 5 (20)	0.750	1.000	$-0.250\ [-0.450, -0.100]$
Countdown 7 (20)	0.650	1.000	$-0.350\ [-0.550, -0.150]$
Sudoku 4×4 (20)	0.900	1.000	$-0.100\ [-0.250, 0.000]$
All (80)	0.825	1.000	$-0.175\ [-0.263, -0.100]$
Matched decision budget (✓, choice registered)			
Countdown 3 (20)	1.000	0.120	$+0.880\ [+0.770, +0.970]$
Countdown 5 (20)	0.750	0.110	$+0.640\ [+0.370, +0.870]$
Countdown 7 (20)	0.650	0.020	$+0.630\ [+0.410, +0.840]$
Sudoku 4×4 (20)	0.900	0.060	$+0.840\ [+0.700, +0.950]$
All (80)	0.825	0.0775	$+0.747\ [+0.648, +0.838]$

struction, so the gap between the two arms in Table 16 is the evidence that choices matter. The matched-budget prediction held, and the paper-budget saturation follows by construction. Published GPT-4o success on these configurations is 0.768. One amendment, made before any audit episode, only reordered the smoke and audit start, and the ledger spent 2.89 of 12 GPU-h with no paid API use.

N Receipt Tables

This appendix holds the gate receipts, the per-window ledgers, and the artifact index, so that a reader can trace each reported run to its receipt.

Every branch of the program reconciles with its ledger and replay, with zero missing evidence (Table 17). Five branches completed at v1 scope, and two completed at reduced v2 scope under new versioned protocols. The table combines each branch's frozen scope, gate status, coverage, and compute from the synthesis and follow-up ledgers, and its Tickets column indexes internal ticket numbers.

The Gate column uses one code per kind of outcome, because the branches ended in different ways. C is completion at declared scope, $V \to C$ a frozen v1 scope reduction followed by reduced-scope v2 completion, and R the reduced recovery reserve. Q_s denotes choice sensitivity under the choice-frontier contract, Q_r registered choice in the external audit, P the instrument-validation pass, and ■ saturation under the LLM-First Search paper budget. ✓ marks a positive gate, = non-separation, $\equiv$ equivalence on the validation panel, and ? an inconclusive result on the held-out panel. In the GPU-hour column, the second-backbone cap reflects the prospective 12.11 GPU-h reserve transfer, and corruption and seed replication share the existing program caps. All follow-up windows have independent ledgers.

The program ledger charges failed attempts as well as successful ones, so its total does not hide unsuccessful runs. At the synthesis snapshot, the program recorded 135 attempts (96 succeeded, 36 failed, 3 cutoff), and failed and cutoff attempts retain their hours in the ledger. Every attempt ended before the absolute scheduler cutoff of 2026-09-21T11:55:19Z.

Two later addenda brought the program total to 68.52/336 GPU-h: the observation-corruption suite added 2.22 GPU-h to the generalization branch, and the training-seed replication addendum (closed 2026-09-21) added 12.79 GPU-h and 534 independently replayed episodes covering two cells.

Follow-up windows. Each follow-up window has its own cap and ledger, none is added to the program total, and every window finished under its cap. Failed attempts retain their hours in every window. The choice-frontier window spent 1.069 of 12 GPU-h (training 0.81, smoke 0.052, model evaluation 0.208), with corpus derivation and controls on CPU. It replayed its 144 panel episodes

Table 17: **Every branch reconciles with zero missing evidence, and each follow-up window keeps its own GPU-hour ledger.** Scope gives bindings or episodes and independently replayed counts. Follow-up windows are not summed into the program total. Gate codes are explained below.

Branch	Tickets	Gate	Scope / replay	GPU-h / cap
Expanded baseline	#116–#118	C	1,152 bindings; 576 model; 1,152/1,152	6.53 / 56
DAgger	#78–#84	C	405 episodes; 81 paired; 405/405	10.91 / 48
Successors	#85–#89, #99	C	90 episodes; 15 cells; 90/90	10.85 / 64
Curriculum	#119	C	243 model; 324 controls; 567/567	11.78 / 48
Transfer	#104–#107	C	39 cells (12 v1, 27 v2); 39/39	3.60 / 24
Backbone v2	#101–#103, #123	$V \to C$	72 model; 72 controls; 144/144	6.03 / 52.11
Generalization	#96, #98, #124	$V \to C$	1,200 (300 model, 900 controls); 1,200/1,200	6.04 / 32
Corruption	#126	C	360/360 new	shared
Seed replication	#127	C	534/534 new	12.79 / —
Recovery reserve	—	R	—	0.00 / 11.89
Program total				**68.52 / 336**
Follow-up windows (independent ledgers)				
Choice frontier	#132	Q_s	144/144 panel; 58 preparations	1.069 / 12
Validation	#135	P	696/696 CPU; 48/48 model	1.19 / 4
Scaled adapter	#136	✓	24/24 learned	4.84 / 30
External audit	#137	Q_r	1,000 pairs; 200 tasks	CPU
Replication / panels	#138, #139	✓	48/48 new; learned 88/88 P2, 96/96 P2u	15.22 / 24
DAgger ablation	#140	$=$	24/24 validation; 22/22 P2	9.06 / 40
Zero-shot	#141	$\equiv$ / ?	46/46 model	2.25 / 30
LFS audit	#142	■	80/80 reference; 400/400 random-valid	2.89 / 12

and re-derived its 58 preparation episodes bit-exact. The three-seed replication and the held-out and unscreened panels shared one window, which spent 15.22 of 24 GPU-h. The DAgger ablation window spent 9.06 of 40 GPU-h. The zero-shot window spent 2.25 of 30 GPU-h, and the LLM-First Search audit spent 2.89 of 12 GPU-h on its reference server with no paid API use.

Artifact index. Each follow-up claim class in the paper traces to a pinned evidence file, and Table 18 gives the mapping. Paths are relative to the evidence repository.

O Results Detail: Gate Receipts and Qualification Gates

This appendix gives the program's receipts, the follow-up ledgers, and the qualification gates that the accounting paragraph of Appendix N summarizes.

The expanded program stayed well inside its compute cap, and every executed branch reconciles with zero missing evidence. It ran on 2 A100 80GB GPUs under a 336 GPU-hour cap with an absolute scheduler cutoff, and spent 68.5244 of those hours in total, 53.52 (15.9%) at the synthesis snapshot and the remainder on the corruption and seed-replication addenda. Of the seven admitted branches, five completed at v1 scope and two completed at reduced v2 scope under new versioned protocols. Independent replay covers 1,152/1,152 baseline episodes, 405/405 DAgger episodes, and 90/90 successor-prediction episodes. Table 17 gives frozen scope, status, coverage, and ledger spend per branch.

The follow-up windows ran on their own ledgers and are never summed with the main program. Most used separate GPU ledgers, and the external audits ran on CPU. Each stayed within its cap: the choice-frontier window spent 1.069 of 12 GPU-hours, the three-seed replication and panel window 15.22 of 24, and the DAgger ablation window 9.06 of 40. Each follow-up closed with complete coverage and independent replay (Table 17).

Three earlier gates fixed which algorithms the active matrix contains. The bounded BFS pilot established executable BFS at scale: on a frozen outcome-blind 15-task panel, final checkpoints from

Table 18: **Each follow-up claim class traces to pinned evidence files.** Filenames are relative to each directory prefix and the evidence repository, and claim classes identify their use.

Relative file / directory	Claim class
`outputs/native-arms/v1/identity-audit.json`	Enumeration identity
`outputs/expanded-study/v1/second-backbone-v2/evaluation/`	Backbone BFS
`outputs/choice-frontier/v1/evaluation/` (evaluation, identity-audit, analysis, final-audit)	Gate / panel / replay
`v1/preparation/visual-choice-frontier/` (report, audit)	Corpus derivation
`v1/budget.json`	Window accounting
`v4/seeds/metrics/analysis.json`	Three-seed validation
`v4/panels/metrics/analysis.json`	P2 / P2u / concentration
`v5/metrics/analysis.json`	DAgger ablation
`v4/budget.json`; `v5/budget.json`	Window accounting
`v6/metrics/analysis.json`	Zero-shot policy
`v6/budget.json`	Window accounting
`outputs/external-audit/v2/lfs/identity-audit.json`	LFS identity
`budget.json`	External v2 ledger
`docs/experiments/choice-frontier/issue-132-closeout.md`	Binding erratum
`choice-frontier/metric-adjudication-2026-09-23.md`	Metric adjudication

five process-SFT seeds achieved 1.0 invariant-valid success with a 0.0 invalid-operation rate, and the base achieved 0.0. The BFWS structural gate then passed on its own frozen panel (Table 4).

The third gate retired the optimal A* arm over $h_{\max}$ with landmark counting. The two additive best-first variants supersede it. The five-seed BFS pilot above predates the budget rule, which authorizes one training run at seed 17 per enumeration-contract model and cell (Section 4).

P Retained-Infrastructure Feasibility Notes

This appendix defines the verified infrastructure components behind corpus generation, evaluation, and evidence production. They are feasibility evidence for the runtime and rendering pipeline and cannot support a Search Process Policy efficacy claim.

Plan pipeline. **Plan Submission** transmits a supplied plan from the project adapter to a planning or rendering backend. **Plan Interpretation** converts the submitted plan into the backend's ordered action representation. An **Action Sequence** is the canonical identity of a supplied plan as a normalized, ordered sequence of grounded action predicates. Casing and whitespace are not part of its identity.

Rendering pipeline. **Render Production** creates a visualisation output and its rendered image for a planning state. **Render Validation** confirms that the produced visualisation and image satisfy declared structural and semantic checks. **Plan Provenance** links a produced visualisation to the exact supplied plan that the backend interpreted.

Certification and evidence. **Integration Certification** is the finding that Plan Submission, Plan Interpretation, Plan Provenance, Render Production, Render Validation, and required execution-isolation checks all passed. An **Attempt** is one authorized, immutable execution with its own identity and retained outcome, whether successful or not. An **Evidence Bundle** is the self-contained records from an Attempt that allow an independent reproducer to verify its claims offline. The retained infrastructure supplies deterministic compact delta/event evidence with semantic replay while

Table 19: **Two non-comparable decision contracts.** The policy emits and the runtime checks different units under the enumeration and choice-frontier interfaces. The identity-audit row records the executed scope, not a cross-contract performance comparison.

Attribute	Enumeration	Choice frontier
Policy output	Grounded, unscored Typed Search Operation	Opaque frontier label
Frontier owner	Search Memory; visited / score	Runtime; seeded per-decision labels
Observation	Modality adapter	Visual: task / initial / 128px frontier / goal
Learned budget	$2\times$ reference decisions; base: 1 call	$2\times$ reference expansions and decisions; goal free
Identity audit	Diagnostic, additive cells	Pre-registered, 18 pairs

preserving search semantics, serving the replay-determinism requirement of the Trusted Search Runtime.

Historical terms. Planning Certificate, Joint Action-and-Certificate SFT, Adaptive Scaffolding, Support Route, Live Memory, Verified Joint Step, and Route Label are historical CGAS terms for retained infrastructure internals. They cannot support any current Search Process Policy efficacy claim and are not referenced elsewhere in this manuscript.

Q Decision Contracts

The two decision contracts differ in what the policy emits and what the runtime owns, so their results are not comparable. Table 19 sets them side by side. Section 3 defines the enumeration contract, and Appendix L.1 defines the choice-frontier contract.

R Use of LLM Statement

An LLM was used only for polishing language; the scientific content is entirely original.