ADAPTIVE SMALL WORLDS: QUERY-SUFFICIENT COMPILATION FOR EFFICIENT WORLD-MODEL PLANNING

Anonymous authors
Paper under double-blind review

ABSTRACT

World models enable planning by simulating actions, but a model broad enough to support many tasks may be unnecessarily large for any single decision. Moreover, a planning query may depend on only a few state factors such as object locations and resource counts. Thus, planning in the full model spends rollouts on irrelevant factors, whereas aggressive pruning can cause failures. We propose *Adaptive Small Worlds* (ASW), a test-time compiler that can reduce an existing world model to a query-specific *small world*, *i.e.*, a planning model of selected factors, action effects, and rewards. It selects factors from a supplied state description via agent-generated rules to build a small world, then uses dependency links built from training traces to add prerequisites and necessary missing state factors. ASW searches for this reduced model, decodes proposals into environment actions, and checks their initial steps in the original model within the same transition budget. Our analysis shows that query-specific representations can require exponentially fewer states than a shared representation, and bounds the expected cumulative reward loss relative to planning in the original model under stated assumptions. Across BabyAI, QueryStress, and symbolic Craftax, our proposed ASW improves the success–budget frontier over both static abstraction, query-based baselines, and conventional planners. These results support our claim: efficient planning should first compile the smallest executable world that preserves the present decision while retaining a well-formed route back to the full one.

1 INTRODUCTION

World models often act as a predictive model that can predict the consequences of actions based on states and a given planning query (Sutton, 1991; Hafner et al., 2019; 2020; Schrittwieser et al., 2020; Hansen et al., 2024; Hafner et al., 2025). The *state* describes the current situation through variables such as object locations, door status, and resource counts; we call these variables *state factors* (Boutilier et al., 2000). A *planning query* specifies a goal and constraints, with optional hypothetical changes such as a blocked passage. We call the available predictive model or simulator the *full model*. Although the full model must cover many goals and contingencies, one planning query may depend on only a small subset of its state factors and transitions. Planning over every factor wastes a finite rollout budget, but removing factors too aggressively is unsafe because an unmentioned prerequisite or hazard may determine whether a plan can be executed. We therefore ask: during test-time procedure, *how can a planner construct a smaller model for each query while retaining the information needed for executable, high-quality actions?*

Existing approaches reduce planning cost via compact latent world models, sparse imagination, and temporal abstraction reduce the cost of predictive search (Hafner et al., 2019; Hansen et al., 2024; Zhou et al., 2025; Chun et al., 2025; Zhang et al., 2026). State abstraction, bisimulation, value equivalence, and partial models remove distinctions that are unnecessary for control (Li et al., 2006; Ferns et al., 2011; Gelada et al., 2019; Grimm et al., 2020; Alver & Precup, 2023). Goal- and context-conditioned methods adapt an abstraction to a task or behavioral context (Chitnis et al., 2021; Hansen-Estruch et al., 2022). Minimal value-equivalent partial models retain only task-relevant aspects, while CAMPs learns context-specific abstractions in factored environments (Alver & Precup, 2023; Chitnis et al., 2021). However, these methods do not jointly provide a *test-time procedure* to implement an online reasoning under a shared budget.

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

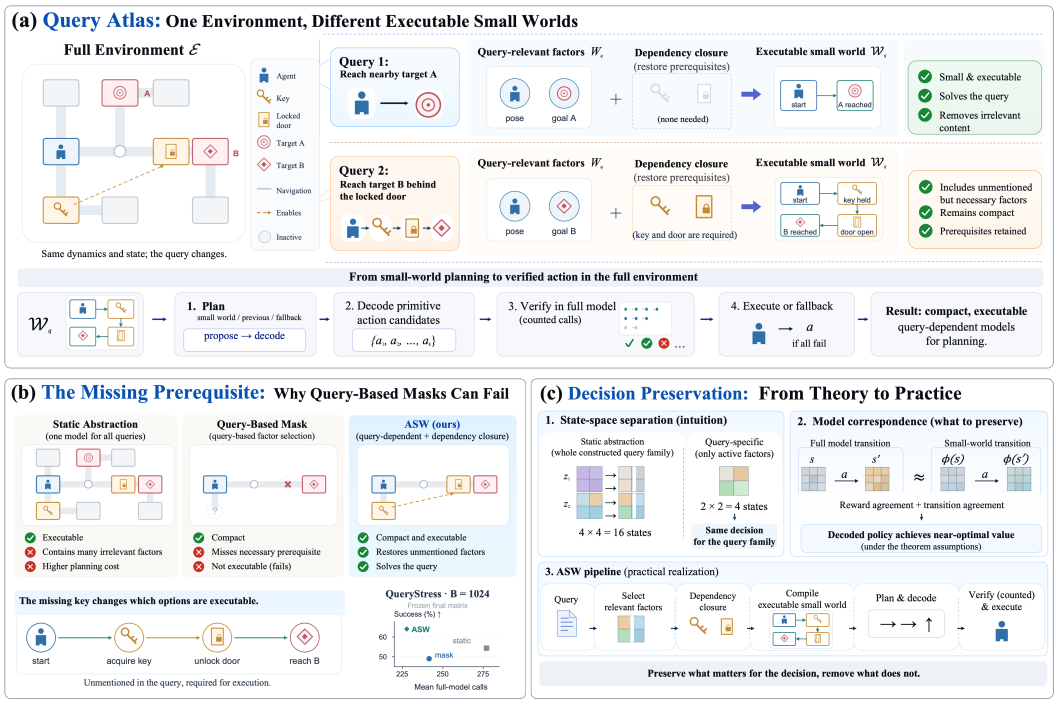

Figure 1: **Adaptive Small Worlds: one environment, different executable models.** **(a)** ASW compiles query-specific small worlds, plans within them, and verifies decoded action prefixes in the full model under a shared budget. **(b)** Dependency closure restores unmentioned prerequisites, *e.g.*, the key needed to open a door, that query-only masking can miss. **(c)** The theory characterizes state-space savings and bounds the decoded-policy value gap under the stated assumptions.

Recent adaptive world models modify other parts of the planning system. AdaWorld adapts latent-action visual dynamics to new environments and action spaces (Gao et al., 2025). The Test-Time Mixture of World Models (TMoW) instead adapts the routing over a bank of world models for new or evolving domains (Jang et al., 2026). Physically Viable World Models advocate composing the simplest physical model sufficient for an intervention query (Thorpe et al., 2026). However, adapting action representations or routing among predictors does not determine which state factors can be omitted when the goal changes within the same environment. Physically Viable World Models recognize this need for query-specific abstractions but leave their autonomous selection and composition as an open problem (Thorpe et al., 2026). Even an accurate predictor therefore leaves a planning trade-off: an overly broad model wastes search, but an overly narrow model omits execution prerequisites. This motivates selecting an executable model before searching for actions, then spending a limited full-model budget on the resulting proposals.

To address this problem, we introduce *Adaptive Small Worlds* (ASW), a test-time compiler that constructs a query-specific *small world*. A small world refers to the selected state factors with action-effect, reward predictions, and an executable action decoder. An offline multi-agent system generates task-interpreting programs from benchmark training tasks, rather than generating a new world model for each test instance. During planning, these frozen programs bind query references to observed objects, check unfinished requirements, and return the variable references that initialize selection. The backward traversal of stored action and training-transition dependencies adds necessary but missed prerequisites. For example, opening a door identifies a target state, but an executable plan may also require the matching key and its location. The compilation procedure therefore can produce different planning models from the same environment as the query changes.

Dependency expansion and verification address different errors. Expansion helps retain actions and prerequisites needed to represent a useful plan; verification checks whether a selected proposal executes as predicted. Checking only cannot recover an unproposed route, while expansion only cannot establish the transition accuracy. ASW therefore searches the reduced model and checks selected decoded action prefixes, *i.e.*, initial sequences considered for execution, without repeating

a full-model search. Every such transition counts against the same budget. A fixed gate selects an accepted prefix or a predefined fallback; exact-match caching reuses only compatible query-independent structure.

We validate two theoretical results. First, when different queries depend on different factors, a separate state abstraction for each query can avoid the exponential state-space growth required by one abstraction that must support them all. Second, a small world does not need to reproduce the full model everywhere; it only needs accurate rewards and transitions along the plans that could be selected. Additional results bound the loss caused by an incorrect verification decision and show how exact-match caching reduces repeated compilation. Together, these results formalize a simple principle: preserve the current decision, rather than the entire world.

We evaluate ASW on BabyAI (Chevalier-Boisvert et al., 2019), QueryStress (Chevalier-Boisvert et al., 2023), and symbolic Craftax (Matthews et al., 2024), together with a controlled exact-match cache ablation. We measure success across declared transition budgets using the normalized area under the success–budget curve and count every full-model transition used for proposal or verification. ASW improves this area over static abstraction by 0.1069 on BabyAI and 0.1005 on QueryStress. At the largest QueryStress budget, ASW improves success by 9.7 percentage points while using 17.8% fewer transitions. In the matched Craftax comparison, it improves success by 5.6 points while using 65% fewer transitions. ASW also retains large gains over best-first search, Monte Carlo tree search (MCTS), and a recurrent policy trained with proximal policy optimization.

Our contributions are:

- **Query-specific world compilation.** We formulate model scope as a test-time decision and compile a different small world for each planning query.
- **Executable and verifiable planning.** We instantiate this idea in a concrete method that combines dependency closure, small-world compilation, primitive-action decoding, budgeted full-model verification, fallback, and exact-match caching.
- **Theory of query-specific state abstraction.** We prove that query-specific state abstractions can be exponentially smaller than one static abstraction and show that decision preservation requires accuracy only along reachable states relevant to the compared plans.
- **Controlled evidence across planning settings.** Experiments on BabyAI, QueryStress, caching, and symbolic Craftax connect the proposed mechanisms to improved success–compute tradeoffs under explicit accounting rules.

See Appendix A for related work.

2 Planning Problem and Method Overview

Planning inputs. We plan from a current state z_0 for a query q, which specifies a goal, constraints, and optional hypothetical changes. The available predictor or simulator provides a *complete model* with states $z \in \mathcal{Z}$ and *primitive actions* $a \in \mathcal{A}$, the individual operations that can be executed in the environment. Its transition function P predicts their consequences. Together with the query reward r_q, it defines the planning problem $\mathcal{M}_q$. A supplied interface F reads the state as K variables or *state factors*, such as object locations, door status, and resource counts, *e.g.*, $F(z) = (x^1(z), \ldots, x^K(z))$.

What ASW constructs. A *small world* $\mathcal{W}_q$ is a reduced executable planning model, not just a list of relevant variables. Its selected index set $W_q \subseteq \{1, \ldots, K\}$ specifies which factors to represent. The *state-abstraction map* ϕ_q reads those factors' current values and retains the task-progress and controller memory needed to interpret them; Step 2 defines this operation explicitly in Equation 2. Action-effect and reward predictions describe how this reduced state changes. The *decoder* Γ_q executes stored action controllers to turn a small-world plan into primitive actions. Thus, W_q selects information, ϕ_q constructs its state representation, and $\mathcal{W}_q$ adds the predictive and executable structure needed for planning.

Decision quality and budget. We seek a cheaper planning model whose decoded decisions retain high reward in the full model. For a small-world policy $\hat{\pi}_q$, *i.e.*, a rule for choosing actions, let $\Gamma_q \hat{\pi}_q$ denote its primitive-action realization. The model and policy are *ε-query-sufficient at* z_0 when

$$V^{\star}_{\mathcal{M}_q}(z_0) - V^{\Gamma_q \hat{\pi}_q}_{\mathcal{M}_q}(z_0) \leq \varepsilon. \tag{1}$$

Here V is expected cumulative reward with discount $0 \leq \gamma < 1$, $V^\star$ is the full-model optimum, and $\varepsilon \geq 0$ is the tolerated loss. Every online call to P during compilation, planning, or verification enters a shared ledger $\mathcal{L}$, with $|\mathcal{L}| \leq B$ for transition-call budget B. Offline preparation is accounted for separately.

Overview of the method. *Step 1*: Frozen task programs identify query references and unfinished requirements; backward dependency traversal adds candidate prerequisites. *Step 2*: The compiler uses these references to construct ϕ_q, attaches action predictions and controllers, and searches the resulting small world. *Step 3*: It checks selected decoded action sequences in the full model, executes an accepted proposal or the fallback, and updates task progress from actual execution.

Exact-match caching reuses only compatible query-independent structure. Figure 1 and Algorithm 1 summarize this sequence; Appendix B.1 gives the formal specification.

3 Adaptive Small-World Compilation

3.1 Inputs, reusable components, and output

Alongside $(\mathcal{M}_q, F, z_0, q, B)$, ASW uses frozen task programs $\mathcal{R}$, a reusable action library, a dependency graph, and cache $\mathcal{C}$. The task programs were generated offline by LLM agents using benchmark training tasks; online, their code parses or looks up queries, checks progress, and returns variable references without LLM calls. Appendix C.2 describes this preparation. The action library supplies executable primitive actions or controllers, together with declared action requirements or training-derived outcome evidence. The dependency graph records relationships used to select these records; it is not a new world model generated for each query. The output is a primitive-action prefix τ_{exec}, an initial sequence to execute before replanning, and the transition ledger $\mathcal{L}$. All routines use the observations and diagnostics permitted by the benchmark interface, not evaluator-only success labels, hidden state, or reference actions.

3.2 Step 1: interpret the query and select factors

Query references and progress. The task programs interpret BabyAI missions with a fixed grammar and bind object descriptions by type, color, and relative location in the observable state. QueryStress and symbolic Craftax instead read registered query or achievement identifiers. Executable completion checks maintain a *progress record*: which requested operations are finished and which remain pending. For example, a matching toggle that opens the target door completes an opening clause; sequencing code then activates the next clause. The same programs insert known current-stage requirements, such as matching-key possession for a locked target door. The returned references, together with explicit goals, constraints, and hypothetical edits allowed, initialize the factor set $W_q^{(0)}$ by set union. This step executes the generated code; it does not ask an online agent or classifier to estimate relevance.

Adding indirect prerequisites. We call the subsequent backward graph traversal *dependency closure*: it expands the task and action records associated with the seeds and adds their bound state-variable references to W_q. For BabyAI and QueryStress, the links are supported by co-occurrences of starting conditions and observed effects in short training action sequences. For Craftax, they connect the required resources, tools, or stations to their production actions. These are dependencies between semantic records, not necessarily causal edges between independent state variables. The graph selects records; their bound variable references determine which state fields are retained. The expansion ends at a fixed point or a configured depth/size limit, and the resulting W_q is held fixed within that planning call. It can change at replanning, but the existing ledger is retained whenever planning resumes within the same budgeted query. The query-based mask omits the backward expansion. Appendix C.4 gives the initialization, traversal, and priority rules; bounded expansion does not certify that every prerequisite has been recovered.

3.3 Step 2: construct the small world and propose actions

Constructing the state map. Once W_q is selected, the compiler constructs ϕ_q by reading the corresponding fields of F:

$$\bar{z} = \phi_q(z) = \big((x^k(z))_{k \in W_q},\, m(z)\big), \qquad \bar{z}_0 = \phi_q(z_0). \tag{2}$$

Here, $m(z)$ records the progress of the task and the execution position of any active controller. It is maintained by task programs and controllers, not inferred by a new network; formally, z includes this memory. The set of possible reduced states is $\bar{\mathcal{Z}}_q$. Thus, ϕ_q is an indexed field read with retained memory, not a learned encoder or a second selection procedure. The selected indices remain fixed during search, while their values change across predicted states. A door-opening model can therefore track the target door, matching key, inventory, and navigation information without retaining unrelated objects.

Attaching predictions and executable actions. The compiler selects the associated action-library records to form the planning-action set $\mathcal{B}_q$. Their declared updates or empirical outcomes supply reduced transition predictions $\bar{P}_q$; permitted task rewards or training statistics supply $\bar{r}_q$. It also retains estimated primitive costs $\bar{c}_q$, evidence counts n_q, and uncertainty diagnostics U_q used to rank proposals. Selecting variables alone does not determine these quantities or make the reduced dynamics exact. A planning action $b \in \mathcal{B}_q$ carries a primitive action, a stored action sequence, or an existing controller that chooses primitive actions from permitted observations. Calling that attached implementation defines the decoder Γ_q; it does not synthesize a new controller at test time. The decoder converts action labels, not reduced states, and need not reconstruct omitted variables. The state map, these predictions, and the decoder together constitute $\mathcal{W}_q$. The static-abstraction baseline shares the action library and decoder but uses a query-independent state map and action graph. At a fixed query, the decoder accesses only the permitted interface; any information it uses beyond $\phi_q(z)$ remains part of the decoded model assessed by the local-error conditions.

Search and decoding. Search starts at $\bar{z}_0$ and uses the reduced action records to propose executable routes. BabyAI/QueryStress prioritize records by dependency depth and empirical support before proposing routes; Craftax traverses its ordered resource prerequisites. Appendix C.4 gives these priorities and the general reward–cost objective in Equation 40. The decoder turns each proposal into an initial primitive-action sequence, or an executable controller for producing that sequence as observations arrive. These decoded proposals, not abstract action labels, are passed to verification. The small-world search explores alternatives; the full model is reserved for selected checks and any explicitly charged proposal calls.

3.4 Step 3: Verify proposals and update execution

Candidates and evidence. The candidate set contains the decoded small-world proposal, any previous plan still available, the predefined fallback, and optional alternative proposals. The fallback is the configured primitive policy or controller used when no proposal is accepted; its identity and thresholds are part of the benchmark configuration. Candidates are de-duplicated by their action/controller identity and carry support, uncertainty, predicted verification cost, and observable progress information. A fixed *verification gate* ranks them and tests the checked execution status and configured support/progress requirements. These are programmatic comparisons over permitted diagnostics, not an additional LLM judgment; their formal interface is in Appendix C.4.

Budgeted checks. A verification reserve $B_{\mathrm{v}} \leq B$ is set before online compilation and proposal generation; any preceding full-model calls consume at most $B - B_{\mathrm{v}}$. Each selected prefix is simulated from a separate copy of the same root state and progress memory, calling the full-model transition function P. Every call enters $\mathcal{L}$, and actual counts, not predicted costs alone, enforce the reserve and total budget. Observed execution failures, constraint outcomes, and progress update the candidate's diagnostics without changing the real state. The structured BabyAI/QueryStress configuration considers at most 24 candidates and verifies at most the top two; Craftax uses its registered verification portfolio (Appendix C). This checks a small number of proposals rather than repeating the reduced-model search.

Execution and replanning. ASW executes the highest-ranked accepted proposal, or the fallback when none is accepted within budget. Only actual execution advances the real progress record; replanning then reads the updated state and may select a different W_q. A prefix interrupted by budget exhaustion is not marked fully verified. In the QueryStress repair configuration, a blockage discovered through counted simulation can instead trigger one recompile from the unchanged root, using only the remaining budget. Such evidence is not available before the counted interaction. Acceptance screens the checked prefix; it does not certify the complete plan or establish Equation 1.

Algorithm 1 Adaptive Small-World Compilation

Require: $(\mathcal{M}_q, F, z_0, q, B)$; frozen task programs $\mathcal{R}$, action library, dependency graph, cache $\mathcal{C}$
Ensure: executable primitive prefix τ_{exec} and ledger $\mathcal{L}$
Initialize $\mathcal{L} \leftarrow ()$ and reserve $B_{\text{v}} \leq B$ for verification.
Enforce a $B - B_{\text{v}}$ limit on all pre-verification P calls; log each as it occurs.
Load reusable structure only after exact cache-key validation.
Execute $\mathcal{R}$ to bind the query, read progress, and collect seed references.
Expand dependency records and collect the retained factor indices W_q (Step 1).
Construct ϕ_q by Equation 2 and set $\bar{z}_0 = \phi_q(z_0)$.
Attach the selected action predictions and controllers to form $\mathcal{W}_q$ (Step 2).
Search and decode proposals; include available previous-plan and fallback candidates.
for each selected candidate while the remaining verification budget permits **do**
    Simulate its prefix from a copy of z_0; append each P call to $\mathcal{L}$.
    Update that candidate's permitted diagnostics and acceptance status.
Choose the highest-ranked accepted prefix, or the predefined fallback (Step 3).
Execute it and update real progress; any budgeted execution calls remain charged.
Store only compatible query-independent structure in $\mathcal{C}$.
return $(\tau_{\text{exec}}, \mathcal{L})$.

3.5 Exact-match caching

The cache $\mathcal{C}$ reuses query-independent compilation structure only when the split, environment, full model, action library, task programs, compiler settings, and access regime have matching identifiers or hashes. It does not reuse a previous query's factor set, predicted trajectory, or acceptance decision. Current state values, progress, candidates, and the ledger are recomputed, even on a cache hit. A changed dependency artifact or task program invalidates the corresponding cached structure. Counterfactual evidence has stricter compatibility checks described in Appendix C.4; it is not transferable merely because two queries share a map.

4 Why Query-Specific Small Worlds Can Preserve Decisions

The theory mainly address two questions: how large a representation must be and how much reward its decoded policy can lose.

Query-specific state abstractions can be exponentially smaller. Consider two sets of queries that depend on disjoint binary factor sets S_1 and S_2. There are workloads for which one static state abstraction that supports every query must distinguish all $2^{|S_1 \cup S_2|}$ joint assignments. Given the query, a state abstraction retaining only the active factor set S_i needs $2^{|S_i|}$ states (Proposition 1). The comparison assumes decisions read the state through the abstraction. It concerns the number of abstract states, not an exponential runtime guarantee or minimum description length; the displayed projection is sufficient, not necessarily minimal.

A small world needs to be accurate only where relevant plans go. The small world need not match the full model at every state. Accordingly, we compare one-step rewards and transitions only at reachable states, after each small-world action has been decoded into primitive actions. For every reachable $z \in \mathcal{Z}$ and $b \in \mathcal{B}_q$, suppose

$$\left|r_q^{\Gamma}(z,b) - \bar{r}_q(\phi_q(z),b)\right| \leq \epsilon_r, \qquad \mathrm{TV}\big(\phi_{q\#}P_q^{\Gamma}(\cdot \mid z,b), \bar{P}_q(\cdot \mid \phi_q(z),b)\big) \leq \epsilon_p, \tag{3}$$

where r_q^{Γ} and P_q^{Γ} are the one-step reward and transition induced by Γ_q, and $\phi_{q\#}P_q^{\Gamma}$ expresses that transition in the reduced state space. Here, TV denotes total variation and $R_{\max}$ bounds the absolute one-step rewards in both models; the details are in Appendix B.1. The reachable support contains trajectories of both the returned decoded policy and a representable comparator within ϵ_Γ of the full-model optimum. Assume the returned policy is η-optimal for the unpenalized small-world reward; η therefore includes any bias from Equation 40 and finite search. Then the decoded policy satisfies

$$V^{\star}_{\mathcal{M}_q}(z_0) - V^{\Gamma_q\hat{\pi}_q}_{\mathcal{M}_q}(z_0) \leq \eta + \epsilon_\Gamma + \frac{2\epsilon_r}{1-\gamma} + \frac{4\gamma R_{\max}\epsilon_p}{(1-\gamma)^2}. \tag{4}$$

The bound separates planning and decoding error from local reward and transition error. Dependency expansion targets policy coverage, while transition evidence targets local prediction accuracy. The

guarantee is relative to the supplied full model and concerns a fixed decoded policy; it does not certify gated replanning from finite prefix checks. Appendix B gives the formal statements and proof.

Verification and reuse. Appendix B separately bounds loss relative to a fallback under an explicit false-accept condition and formalizes exact-match amortization. Neither result turns prefix validity into a certificate of near-optimality; caching amortizes only the query-independent construction cost.

5 Experiments

5.1 Benchmarks, baselines, and evaluation protocol

Benchmarks. We evaluate whether query-specific construction improves planning, whether the frozen result survives beyond configuration selection, and when exact-match reuse helps. The evidence covers three structured settings and one paired mechanism study. BabyAI provides compositional instruction-following tasks with explicit object, location, door, and key dependencies (Chevalier-Boisvert et al., 2019). We use seven levels spanning direct navigation, object manipulation, locked-door reasoning, and the composite `MiniBossLevel`. QueryStress is a controlled MiniGrid-based benchmark (Chevalier-Boisvert et al., 2023) in which each environment instance is paired with multiple goal, prerequisite, safety, shortest-path, distractor, and counterfactual queries. The relevant factors change across queries while the map remains fixed. The Craftax evaluation (Matthews et al., 2024) uses twelve achievement-directed tasks with observable resource, inventory, and goal information, testing longer prerequisite chains and reusable temporal actions; it is a goal-conditioned symbolic protocol. Finally, a pre-specified cache intervention repeats exact environment–compiler identities to isolate the effect of exact-match caching.

Baselines. We first compare against two baselines that search the full model: heuristic best-first graph search, following the priority-queue search paradigm of A* (Hart et al., 1968), and budgeted primitive-action Monte Carlo tree search (MCTS), for which upper confidence bounds applied to trees (UCT) is the canonical bandit-guided planning reference (Kocsis & Szepesvári, 2006). Craftax additionally includes a fixed goal-conditioned recurrent neural-network policy trained with proximal policy optimization (PPO-RNN) (Schulman et al., 2017). Two controlled baselines isolate the contribution of query-specific compilation. *Static abstraction* uses the same action library, primitive decoder, and transition source as ASW, but retains one query-independent factor–option graph. *Query-based factor mask* retains factors selected from the query and current observable task progress, but does not add factors for unmentioned dependencies or use ASW's verification and caching mechanisms. These controlled comparators are ablations, not renamed published algorithms; Appendix D.3 specifies their exact interfaces and the corresponding Craftax implementations.

We also conduct comparisons with three recent methods: the value-preserving abstract world model of Rodriguez-Sanchez & Konidaris (2024), which learns one abstract MDP over a fixed option set; VisualPredicator, which learns task-specific predicates and abstract operators (Liang et al., 2025); and SLAP, which learns shortcut options in an abstract planning graph (Liu et al., 2026). Each adaptation receives the same observable state, query or goal, primitive actions, transition source, and charged test-time budget as ASW; offline learning is outside the online transition ledger.

Protocol and statistics. BabyAI contains 175 environment instances and 2,625 method–budget records. The QueryStress final matrix contains 100 environment instances, ten query types, 1,000 paired query–environment instances per method, and 18,000 rows; separate configuration-selection matrices use 50 environment instances. Both use transition budgets $B \in \{64, 256, 1024\}$. The configuration-matched Craftax matrix evaluates 12 goals with 30 paired episode seeds per goal and $B \in \{256, 1024, 4096\}$; the original primitive-planning matrix is kept separate. Success is the terminal query-satisfaction indicator, and every transition used for proposal scoring or verification is charged. Paired effects use frozen instances, cluster bootstrap, paired randomization tests, and Holm correction across each predeclared set of comparisons. Appendix D gives matrix identities, access regimes, clustering units, complete operating points, and accounting rules.

5.2 Main success–compute results

Query-dependent navigation. On BabyAI, ASW reaches 0.869 success with 24.0 counted transitions at $B = 256$, versus $0.857/45.0$ for the query-based factor mask and $0.754/72.2$ for

Table 1: **Reported high-budget operating points.** Success and mean counted full-model transitions, including verification; BabyAI/QueryStress use $B = 1024$ and symbolic Craftax uses $B = 4096$. The upper block reports compiler and abstraction operating points; the lower block provides primitive-planning context. QueryStress compiler rows use the selection matrix, whereas primitive rows use the frozen final matrix and are not paired with them. The matched frozen compiler comparison is in Table 2. Bold denotes the highest success or lowest transition count within each block; recent-method rows support operating-point comparisons, not all-budget AUC tests.

Method	BabyAI		QueryStress		Craftax (symbolic)	
	Success	Trans.	Success	Trans.	Success	Trans.
ASW (ours)	**0.869**	113.0	**0.670**	240.8	**0.817**	**291.5**
SLAP (Liu et al., 2026)	0.832	112.1	0.626	245.4	0.795	468.3
VisualPredicator (Liang et al., 2025)	0.842	142.6	0.618	262.8	0.767	721.6
Abstract WM (Rodriguez-Sanchez & Konidaris, 2024)	0.808	119.3	0.610	296.5	0.745	648.0
Static abstraction	0.760	124.5	0.584	277.9	0.761	842.5
Query-based factor mask	0.857	**94.3**	0.532	**227.7**	0.733	642.6
Best-first search	0.669	429.0	0.178	927.3	–	–
Primitive-action MCTS	0.549	589.5	0.233	900.9	0.162	1563.2

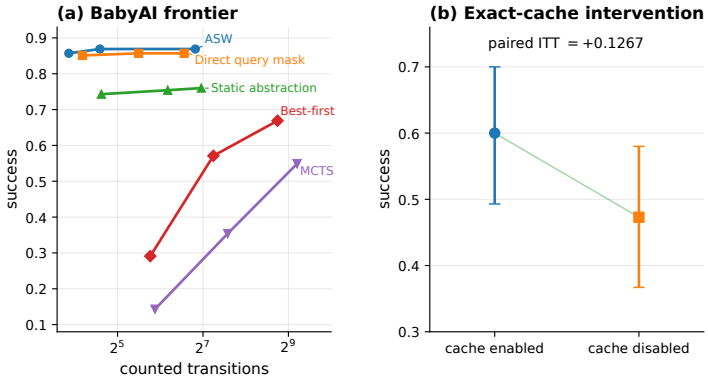

Figure 2: **Reduced-model planning and exact-match reuse. (a)** BabyAI success versus realized full-model transitions at the three registered budgets. **(b)** Cache-enabled and cache-disabled success in the paired intervention; error bars are 95% cluster-bootstrap confidence intervals, and the annotation is the paired difference. "Direct query mask" denotes the query-based factor mask. All registered verification calls are counted.

static abstraction. Its registered frontier-AUC gain over static abstraction is $+0.1069$ (95% CI $[0.0641, 0.1514]$, Holm $p = 0.0004$; Appendix Table 6). The gain over the query mask is much smaller in success, and that mask uses fewer calls at $B = 1024$. Thus, dependency-aware compilation is most useful when the retained execution structure compensates for its additional cost, rather than uniformly dominating direct selection.

On the 50-world QueryStress selection matrix, ASW reaches 0.670 success at $B = 1024$, versus 0.584 for static abstraction and 0.532 for the query mask (Table 9). The separate 100-world frozen evaluation supports the same direction without pooling the two matrices: success is 0.640 versus $0.543/0.490$. Against static abstraction, the frozen transition ratio is 0.822 (95% CI $[0.741, 0.927]$), including verification costs.

Long-horizon symbolic planning. The Craftax compiler matrix reaches 0.8167 success at both $B = 1024$ and $B = 4096$; static abstraction and the query mask reach 0.7611 and 0.7333 at the largest budget (Table 11). The declared-budget AUC gains are $+0.0382$ and $+0.0646$, respectively. At $B = 256$, however, ASW trails static abstraction (0.7417 versus 0.7556), exposing the cost of the more elaborate candidate and verification procedure when the budget is tight. The separate primitive-planning study reports frontier-AUC gains of $+0.59/+0.70$ over MCTS/PPO-RNN, but those comparisons also change access to reusable temporal actions and do not isolate factor selection.

Recent methods and cross-benchmark pattern. At the high-budget operating points in Table 1, ASW has the highest reported success on each benchmark. The unweighted mean success differ-

Table 2: **Frozen QueryStress: more simulation versus a different compiled model.** The same 100 worlds and ten query types are used throughout. Each budget column reports success / mean counted transitions. $\Delta\mathrm{AUC}_B$ is ASW minus the listed comparator over all three budgets, with a 95% cluster-bootstrap interval.

Method	$B = 64$	$B = 256$	$B = 1024$	$\Delta\mathrm{AUC}_B$ [95% CI]
ASW	**0.621** / 39.3	**0.640** / **100.3**	**0.640** / 227.8	–
Base ASW	0.526 / **35.1**	0.554 / 110.7	0.562 / **226.1**	$+0.0863$ $[0.0383, 0.1365]$
Static abstraction	0.510 / 42.7	0.543 / 113.9	0.543 / 277.0	$+0.1005$ $[0.0655, 0.1360]$
Query-based mask	0.472 / 37.0	0.490 / 121.9	0.490 / 241.6	$+0.1498$ $[0.1018, 0.2005]$

ences over Abstract WM, VisualPredicator, and SLAP are 6.43, 4.30, and 3.43 percentage points, respectively. Averaging the relative transition reductions separately across the three benchmarks gives 26.36%, 29.58%, and 12.94%. These are summaries of the operating points, not AUC differences or paired cross-benchmark significance tests. The trade-off is not uniform: on BabyAI, ASW uses 113.0 transitions versus SLAP's 112.1, while improving success from 0.832 to 0.869. The results support the evaluated compiler as a useful design alternative, without isolating factor selection from its decoding and verification mechanisms.

5.3 Frozen evidence beyond configuration selection

Changing the model is not equivalent to increasing the budget. At $B = 64$, ASW already exceeds static abstraction and the query mask at $B = 1024$ in reported success (0.621 versus $0.543/0.490$; Table 2). This cross-budget comparison is descriptive, while the paired all-budget effects provide the statistical evidence: the three compiler comparisons have positive intervals and Holm-adjusted $p \leq 0.00065$. The base ASW configuration already uses dependency closure; the additional $+0.0863$ AUC gain therefore supports the integrated query compiler and its execution checks beyond that base, not an attribution to closure alone.

More calls need not yield more success. In the frozen matrix, ASW's reported success remains 0.640 from $B = 256$ to 1024, while mean calls grow from 100.3 to 227.8. BabyAI and the Craftax compiler matrix show the same reported success plateau between their middle and largest budgets. The useful outcome is a success–cost operating point, not budget exhaustion; these aggregates do not validate an adaptive stopping rule.

5.4 Exact-match caching and experimental completeness

On 150 pre-specified paired cache keys spanning 50 environments at $B = 1024$, caching raises success from 0.473 to 0.600 (95% CI $[0.0667, 0.1933]$) and reduces mean counted transitions from 262.7 to 226.4. The intervention changes cache availability while retaining the frozen planning configuration, providing a more isolated mechanism test than the multi-component compiler comparisons. Query-specific records and decisions are recomputed; changes in the environment or explicit invalidation force a cold compilation (Figure 3).

6 Conclusion and Limitations

We validate that world-model planning does not need to use the same state abstraction for every query. ASW compiles a query-specific small world and plans within it under the same transition budget. The theory shows that accuracy on reachable states can be sufficient for near-optimal planning and that query-specific state abstractions can be exponentially smaller than one static state abstraction. The experiments show improved success–compute tradeoffs across structured navigation, multi-query stress tests, exact-match caching, and symbolic control. A current limitation is that ASW assumes an auditable factor and option interface; errors in query parsing or dependency specification can, therefore, remove information needed for planning. Future work will learn these interfaces from high-dimensional observations and extend the verification to stochastic, partially observed, and continually changing environments.

AI USE STATEMENT

Generative AI assisted with literature comparison, manuscript editing, experimental results checking, code validation, and editing of the scientific figure. An offline multi-agent system using the reported Qwen3.5-35B, GLM-4-32B, and DeepSeek-V3 model pool generated task-interpreting and progress-selection code from benchmark training tasks (Appendix C.2). Online ASW planning executes frozen programs without LLM calls. The authors take responsibility for the final claims, code, and reported evidence.

REPRODUCIBILITY STATEMENT

Appendices B–E specify assumptions, complete proofs, benchmark-specific compiler configurations, access regimes, baselines, budgets, clustering units, statistical procedures, complete result matrices, and boundary diagnostics. The corresponding experiment artifact is uploaded in the supplementary material, including the anonymized code, exact configurations, and frozen instance manifests.

ETHICS STATEMENT

This work studies planning efficiency in simulated and benchmark environments and does not involve human participants, personal data, or deployment in safety-critical systems.

REFERENCES

Safa Alver and Doina Precup. Minimal value-equivalent partial models for scalable and robust planning in lifelong reinforcement learning. In *Conference on Lifelong Learning Agents*, pp. 548–567. PMLR, 2023.

Craig Boutilier, Richard Dearden, and Moisés Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial intelligence*, 121(1-2):49–107, 2000.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. Babyai: A platform to study the sample efficiency of grounded language learning. In *International Conference on Learning Representations*, 2019.

Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo Perez-Vicente, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and J K Terry. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. *Advances in Neural Information Processing Systems*, 36:73383–73394, 2023.

Rohan Chitnis, Tom Silver, Beomjoon Kim, Leslie Kaelbling, and Tomas Lozano-Perez. Camps: Learning context-specific abstractions for efficient planning in factored mdps. In *Conference on robot learning*, pp. 64–79. PMLR, 2021.

Junha Chun, Youngjoon Jeong, and Taesup Kim. Sparse imagination for efficient visual world model planning. *arXiv preprint arXiv:2506.01392*, 2025.

Nicola Dainese, Matteo Merler, Minttu Alakuijala, and Pekka Marttinen. Generating code world models with large language models guided by monte carlo tree search. *Advances in Neural Information Processing Systems*, 37:60429–60474, 2024.

Ryan D'Orazio, Danilo Vucetic, Zichu Liu, Junhyung Lyle Kim, Ioannis Mitliagkas, and Gauthier Gidel. Solving hidden monotone variational inequalities with surrogate losses. In *International Conference on Learning Representations*, volume 2025, pp. 77721–77749, 2025.

Norm Ferns, Prakash Panangaden, and Doina Precup. Bisimulation metrics for continuous markov decision processes. *SIAM Journal on Computing*, 40(6):1662–1714, 2011.

Shenyuan Gao, Siyuan Zhou, Yilun Du, Jun Zhang, and Chuang Gan. Adaworld: Learning adaptable world models with latent actions. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=QQegZj99sk.

Carles Gelada, Saurabh Kumar, Jacob Buckman, Ofir Nachum, and Marc G Bellemare. Deepmdp: Learning continuous latent space models for representation learning. In *International conference on machine learning*, pp. 2170–2179. PMLR, 2019.

Christopher Grimm, André Barreto, Satinder Singh, and David Silver. The value equivalence principle for model-based reinforcement learning. *Advances in neural information processing systems*, 33: 5541–5552, 2020.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pp. 2555–2565. PMLR, 2019.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1lOTC4tDS.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. *Nature*, 640(8059):647–653, 2025.

Nick Hansen, Hao Su, and Xiaolong Wang. Td-mpc2: Scalable, robust world models for continuous control. In *International Conference on Learning Representations*, volume 2024, pp. 47376–47405, 2024.

Philippe Hansen-Estruch, Amy Zhang, Ashvin Nair, Patrick Yin, and Sergey Levine. Bisimulation makes analogies in goal-conditioned reinforcement learning. In *International Conference on Machine Learning*, pp. 8407–8426. PMLR, 2022.

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.

Dong Huang, Jie M Zhang, Michael Luck, Qingwen Bu, Yuhao Qing, and Heming Cui. Agentcoder: Multi-agent-based code generation with iterative testing and optimisation. *arXiv preprint arXiv:2312.13010*, 2023.

Jinwoo Jang, Minjong Yoo, Sihyung Yoon, and Honguk Woo. Test-time mixture of world models for embodied agents in dynamic environments. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=LQD1MrnbxH.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pp. 282–293. Springer, 2006.

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

Yichao Liang, Nishanth Kumar, Hao Tang, Adrian Weller, Joshua B Tenenbaum, Tom Silver, João F Henriques, and Kevin Ellis. Visualpredicator: Learning abstract world models with neuro-symbolic predicates for robot planning. In *International Conference on Learning Representations*, volume 2025, pp. 60416–60444, 2025.

Y Isabel Liu, Bowen Li, Benjamin Eysenbach, and Tom Silver. Slap: Shortcut learning for abstract planning. In *International Conference on Learning Representations*, volume 2026, pp. 68901–68926, 2026.

Zichu Liu and Lacra Pavel. Recursive reasoning in minimax games: A level k gradient play method. *Advances in Neural Information Processing Systems*, 35:16903–16917, 2022.

Hongbo Lu, Liang Yao, Chenghao He, Hao Han, Fan Liu, Wenlong Liao, Tao He, and Pai Peng. Rise: Adaptive imagination for world action models. *arXiv preprint arXiv:2608.20430*, 2026.

Michael Matthews, Michael Beukman, Benjamin Ellis, Mikayel Samvelyan, Matthew Thomas Jackson, Samuel Coward, and Jakob Nicolaus Foerster. Craftax: A lightning-fast benchmark for open-ended reinforcement learning. In *International Conference on Machine Learning*, pp. 35104–35137. PMLR, 2024.

Felix O'Mahony, Roberto Cipolla, and Ayush Tewari. Vdaworld: World modelling via vlm-directed abstraction and simulation. *arXiv preprint arXiv:2512.11061*, 2025.

Zhengqi Pei and Shuhui Wang. Dynamics-inspired neuromorphic visual representation learning. In *International Conference on Machine Learning*, pp. 27521–27541. PMLR, 2023.

Zhengqi Pei, Anran Zhang, Shuhui Wang, Xiangyang Ji, and Qingming Huang. Data-free neural representation compression with riemannian neural dynamics. In *International Conference on Machine Learning*, pp. 40129–40144. PMLR, 2024.

Zhengqi Pei, Qingming Huang, and Shuhui Wang. When LLMs develop languages: Symbolic communication for efficient multi-agent reasoning. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=ovpL0ujD6j.

Rafael Rodriguez-Sanchez and George Konidaris. Learning abstract world models for value-preserving planning with options. In *Reinforcement Learning Conference*, 2024. URL https://openreview.net/forum?id=h9IvopsMFS.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Richard S Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163, 1991.

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

Hao Tang, Darren Key, and Kevin Ellis. Worldcoder, a model-based llm agent: Building world models by writing code and interacting with the environment. *Advances in Neural Information Processing Systems*, 37:70148–70212, 2024.

Adam J Thorpe, Stepan Tretiakov, Cheng-Hsi Hsiao, Su Ann Low, Xingjian Li, Hassan Iqbal, Neel P Bhatt, Ufuk Topcu, and Krishna Kumar. Physically viable world models: A case for query-conditioned embodied ai. *arXiv preprint arXiv:2605.30542*, 2026.

Shoubin Yu, Yue Zhang, Zun Wang, Jaehong Yoon, Huaxiu Yao, Mingyu Ding, and Mohit Bansal. When and how much to imagine: Adaptive test-time scaling with world models for visual spatial reasoning. In *The 2nd Workshop on Foundation Models Meet Embodied Agents at CVPR 2026*, 2026. URL https://openreview.net/forum?id=JXTFsvh6Z7.

Wancong Zhang, Basile Terver, Artem Zholus, Soham Chitnis, Harsh Sutaria, Mido Assran, Randall Balestriero, Amir Bar, Adrien Bardes, Yann LeCun, et al. Hierarchical planning with latent world models. *arXiv preprint arXiv:2604.03208*, 2026.

Gaoyue Zhou, Hengkai Pan, Yann Lecun, and Lerrel Pinto. Dino-wm: World models on pre-trained visual features enable zero-shot planning. In *International Conference on Machine Learning*, pp. 79115–79135. PMLR, 2025.

A Related Work

World-model planning and adaptive computation. Learned world models support planning through imagined rollouts, tree search, and model-predictive control (Hafner et al., 2019; 2020; Schrittwieser et al., 2020; Hansen et al., 2024; Hafner et al., 2025). Recent optimization work uses surrogate losses for adaptive policy evaluation (D'Orazio et al., 2025) and recursive strategy prediction for planing-based learning (Liu & Pavel, 2022). Methods based on sparse imagination, temporal abstraction, and adaptive rollout allocation reduce the number, duration, or temporal resolution of these rollouts (Chun et al., 2025; Zhang et al., 2026; Yu et al., 2026; Lu et al., 2026). These methods alter the amount or temporal scale of predictive computation; ASW instead alters the state factors included in each small world.

Adaptable and induced world models. AdaWorld adapts latent-action visual dynamics to new environments and action spaces, whereas TMoW routes among multiple world models at test time (Gao et al., 2025; Jang et al., 2026). VDAWorld and WorldCoder construct structured or executable models from observations or interaction (O'Mahony et al., 2025; Tang et al., 2024). These methods change the learned dynamics, the action interface, the choice of models, or the model-construction process. ASW instead assumes that a full model is already available and compiles a query-specific small world from it.

State abstraction and task conditioning. State abstraction, bisimulation, and value-equivalent models remove distinctions that do not affect control (Li et al., 2006; Ferns et al., 2011; Gelada et al., 2019; Grimm et al., 2020). Efficient neuronal representations and neuro-symbolic framework provide an alternative to model the interactions (Pei & Wang, 2023; Pei et al., 2024; 2026). Recent learned variants construct value-preserving abstract MDPs over options or invent task-specific neuro-symbolic predicates and operators (Rodriguez-Sanchez & Konidaris, 2024; Liang et al., 2025); SLAP instead learns shortcut options within an abstract planning graph (Liu et al., 2026). Minimal value-equivalent partial models and CAMPs focus on task-relevant partial models or context-specific variable removal (Alver & Precup, 2023; Chitnis et al., 2021), while Physically Viable World Models seek the simplest physical abstraction that preserves an intervention query (Thorpe et al., 2026). ASW differs by compiling from an available full model at test time, adding factors for unmentioned dependencies, decoding plans to primitive actions, and verifying or rejecting them under the same transition budget.

B Formal Theory and Proofs

B.1 Full planning specification

This subsection supplies the formal types and decoder definitions underlying Section 2.

Full model and factor interface. For a query q in the query set $\mathcal{Q}$, the full planning problem is the discounted Markov decision process

$$\mathcal{M}_q = (\mathcal{Z}, \mathcal{A}, P, r_q, \gamma), \qquad P : \mathcal{Z} \times \mathcal{A} \to \Delta(\mathcal{Z}), \qquad r_q : \mathcal{Z} \times \mathcal{A} \to [-R_{\max}, R_{\max}], \quad (5)$$

where $\mathcal{Z}$ is either a finite structured state space or a subset of $\mathbb{R}^{d_z}$, $\mathcal{A}$ is a finite primitive-action set, $0 \le \gamma < 1$, and $\Delta(\mathcal{Z})$ denotes probability distributions over $\mathcal{Z}$. The current state is $z_0 \in \mathcal{Z}$ and the online transition-call budget is $B \in \mathbb{N}$. The typed factor interface $F : \mathcal{Z} \to \mathcal{X}_1 \times \cdots \times \mathcal{X}_K$ returns $F(z) = (x^1, \ldots, x^K)$ with $x^k \in \mathcal{X}_k$. Factor sets contain indices in $[K] = \{1, \ldots, K\}$ rather than factor values.

Query records. The frozen task programs parse supported instructions or read registered task identifiers and bind object references to the supplied state. Their goal, constraint, and edit records are summarized by

$$\mathrm{parse}(q) = (G_q, C_q, \Xi_q), \quad G_q, C_q \subseteq [K], \quad \Xi_q \subseteq \mathcal{E}, \quad \mathcal{E} := \bigsqcup_{k=1}^{K} \big(\{k\} \times \mathcal{X}_k\big), \quad (6)$$

where G_q and C_q contain goal and active-constraint indices. Each $(k, v) \in \Xi_q$ is a type-compatible hypothetical assignment to factor k; the disjoint union records which factor the assigned value belongs to. We write $\mathrm{vars}(\Xi_q) = \{k : \exists v, (k, v) \in \Xi_q\}$. This notation summarizes the factor references, not the complete task-progress record or its update procedure. Bindings to the current state are implicit in $\mathrm{parse}(q)$. Unknown counterfactual values remain unresolved placeholders until the allowed interaction supplies evidence; they are not treated as assignments in Ξ_q before they are observed.

Executable small world. For the current (q, z_0), the compiler constructs

$$\mathcal{W}_q = (\bar{\mathcal{Z}}_q, \mathcal{B}_q, \bar{P}_q, \bar{r}_q, \bar{c}_q, U_q, \phi_q, \Gamma_q), \tag{7}$$

where $\bar{\mathcal{Z}}_q$ is the reduced state space and $\mathcal{B}_q$ is a finite tagged union of primitive actions, fixed nonempty suffixes in $\mathcal{A}^+ := \bigcup_{h \geq 1} \mathcal{A}^h$, and closed-loop options. The maps $\bar{P}_q : \bar{\mathcal{Z}}_q \times \mathcal{B}_q \to \Delta(\bar{\mathcal{Z}}_q)$, $\bar{r}_q : \bar{\mathcal{Z}}_q \times \mathcal{B}_q \to [-R_{\max}, R_{\max}]$, $\bar{c}_q : \bar{\mathcal{Z}}_q \times \mathcal{B}_q \to \mathbb{R}_{\geq 0}$, and $U_q : \bar{\mathcal{Z}}_q \times \mathcal{B}_q \to \mathbb{R}_{\geq 0}$ specify transition predictions, query rewards, expected primitive-transition costs, and uncertainty estimates. The map ϕ_q is the field projection with retained memory defined in Equation 2. The full state z includes both the physical configuration and the finite task/controller memory $m(z)$, which is updated from observed or simulated transitions. For a fixed planning call, W_q and the field layout remain unchanged while factor values and memory evolve. The existence of this representation does not imply that projecting the full transition law yields an exact Markov model on reduced states; the local-error assumptions quantify the approximation.

Primitive-action decoding. The decoder

$$\Gamma_q(\cdot \mid z, b) \in \Delta(\mathcal{A}) \tag{8}$$

maps a small-world action $b \in \mathcal{B}_q$ at full-model state z to a primitive-action distribution. Variable-duration options are unrolled one primitive step at a time with controller memory, preserving the primitive-step reward and discount convention (Sutton et al., 1999). The induced one-step reward and transition are

$$\begin{aligned} r_q^{\Gamma}(z, b) &= \mathbb{E}_{a \sim \Gamma_q(\cdot \mid z, b)}[r_q(z, a)], \\ P_q^{\Gamma}(A \mid z, b) &= \mathbb{E}_{a \sim \Gamma_q(\cdot \mid z, b)}[P(A \mid z, a)] \quad \text{for measurable } A \subseteq \mathcal{Z}. \end{aligned} \tag{9}$$

A small-world policy $\hat{\pi}_q : \bar{\mathcal{Z}}_q \to \Delta(\mathcal{B}_q)$ induces the primitive policy in Equation 11. Query sufficiency is the value-gap criterion in Equation 1, evaluated at the current full-model state.

Ideal cost objective and accounting. The design target is

$$\begin{aligned} \min_{\mathcal{W}_q, \hat{\pi}_q} \quad & C_{\mathrm{compile}} + C_{\mathrm{plan}} + C_{\mathrm{verify}} \\ \text{s.t.} \quad & V^{\star}_{\mathcal{M}_q}(z_0) - V^{\Gamma_q \hat{\pi}_q}_{\mathcal{M}_q}(z_0) \leq \varepsilon, \qquad |\mathcal{L}| \leq B. \end{aligned} \tag{10}$$

Each cost is a nonnegative scalar, and $\mathcal{L} = ((z_i, a_i, z_i'))_{i=1}^{N_P} \in (\mathcal{Z} \times \mathcal{A} \times \mathcal{Z})^*$ records every call to P, so $|\mathcal{L}| = N_P \leq B$. The actual call count enforces the budget; the value-gap constraint expresses the desired decision quality and is not available as an online acceptance test. The offline task-program generation stage is distinct from these per-query planning costs.

B.2 Definitions and induced policies

Fix a query q and a discounted full-model MDP $\mathcal{M}_q = (\mathcal{Z}, \mathcal{A}, P, r_q, \gamma)$ with $0 \leq \gamma < 1$. Let the small-world MDP be $(\bar{\mathcal{Z}}_q, \mathcal{B}_q, \bar{P}_q, \bar{r}_q, \gamma)$, let $\phi_q : \mathcal{Z} \to \bar{\mathcal{Z}}_q$ be its measurable state-abstraction map, and let $\Gamma_q(\cdot \mid z, b) \in \Delta(\mathcal{A})$ be the Markov decoder introduced in Equation 8. For a small-world Markov policy $\pi : \bar{\mathcal{Z}}_q \to \Delta(\mathcal{B}_q)$, its decoded primitive policy is

$$(\Gamma_q \pi)(a \mid z) = \sum_{b \in \mathcal{B}_q} \pi(b \mid \phi_q(z)) \Gamma_q(a \mid z, b). \tag{11}$$

The corresponding one-step reward and transition kernel on full-model states are

$$r_q^{\Gamma,\pi}(z) = \sum_{b\in\mathcal{B}_q} \pi(b \mid \phi_q(z)) r_q^{\Gamma}(z,b),$$
$$P_q^{\Gamma,\pi}(\cdot \mid z) = \sum_{b\in\mathcal{B}_q} \pi(b \mid \phi_q(z)) P_q^{\Gamma}(\cdot \mid z,b). \tag{12}$$

The small-world policy induces $\bar{r}_q^{\pi}(s) = \sum_b \pi(b \mid s)\bar{r}_q(s,b)$ and $\bar{P}_q^{\pi}(\cdot \mid s) = \sum_b \pi(b \mid s)\bar{P}_q(\cdot \mid s,b)$. Assume throughout that $|r_q^{\Gamma}(z,b)|, |\bar{r}_q(s,b)| \le R_{\max}$, so all discounted value functions have sup norm at most $R_{\max}/(1-\gamma)$. For a probability measure μ on $\mathcal{Z}$, $\phi_{q\#}\mu$ denotes its push-forward through ϕ_q. We use $\mathrm{TV}(\mu,\nu) = \frac{1}{2}\|\mu-\nu\|_1$ for discrete spaces, with the usual measurable-space extension.

Variable-duration options are included by augmenting z with the option-controller state and unrolling the option into primitive transitions. This produces a Markov process to which the statements below apply directly. Equivalently, one may formulate a semi-Markov decision process with duration-dependent discount γ^{ℓ}; the primitive-unrolling presentation avoids introducing a second value convention.

Definition 1 (Reachable query sufficiency)**.** *Let $\mathcal{R}_q \subseteq \mathcal{Z}$ be a measurable support that contains z_0 and is closed under every decoded policy used in the comparison. The tuple $(\phi_q, \bar{P}_q, \bar{r}_q, \Gamma_q)$ is (ϵ_r, ϵ_p)-locally sufficient on $\mathcal{R}_q$ if, for every $z \in \mathcal{R}_q$ and every available $b \in \mathcal{B}_q$,*

$$|r_q^{\Gamma}(z,b) - \bar{r}_q(\phi_q(z),b)| \le \epsilon_r,$$
$$\mathrm{TV}\big(\phi_{q\#}P_q^{\Gamma}(\cdot \mid z,b), \bar{P}_q(\cdot \mid \phi_q(z),b)\big) \le \epsilon_p. \tag{13}$$

Because reward differences and total variation are convex under mixtures, Definition 1 also implies

$$|r_q^{\Gamma,\pi}(z) - \bar{r}_q^{\pi}(\phi_q(z))| \le \epsilon_r,$$
$$\mathrm{TV}\big(\phi_{q\#}P_q^{\Gamma,\pi}(\cdot \mid z), \bar{P}_q^{\pi}(\cdot \mid \phi_q(z))\big) \le \epsilon_p \tag{14}$$

for every small-world policy π and $z \in \mathcal{R}_q$.

Definition 2 (Decoder completeness)**.** *The small-world policy class is ϵ_Γ-complete at z_0 if there exists a small-world policy $\pi^\dagger$ whose decoded policy satisfies*

$$V_{\mathcal{M}_q}^{\star}(z_0) - V_{\mathcal{M}_q}^{\Gamma_q\pi^\dagger}(z_0) \le \epsilon_\Gamma. \tag{15}$$

This is a policy-class coverage condition: it does not require every primitive action to have a small-world representative, only that the compiled decoder class contains a near-optimal policy for the present query and initial state.

B.3 Auxiliary total-variation bound

Lemma 1 (Expectation difference under total variation)**.** *For probability measures μ, ν on a common measurable space and any bounded measurable function f,*

$$|\mathbb{E}_\mu[f] - \mathbb{E}_\nu[f]| \le 2\|f\|_\infty \mathrm{TV}(\mu,\nu). \tag{16}$$

Proof. Let $\sigma = \mu - \nu$ be the signed measure difference. By the definition of total variation norm, $\left|\int f\, d\sigma\right| \le \|f\|_\infty \|\sigma\|_{\mathrm{var}}$. Under the convention $\mathrm{TV}(\mu,\nu) = \frac{1}{2}\|\mu-\nu\|_{\mathrm{var}}$, the right-hand side is $2\|f\|_\infty \mathrm{TV}(\mu,\nu)$, proving Equation 16. □

B.4 Decoded value preservation

Lemma 2 (Fixed-policy simulation bound)**.** *Under Definition 1, for any small-world policy π whose decoded trajectories remain in $\mathcal{R}_q$,*

$$\left|V_{\mathcal{M}_q}^{\Gamma_q\pi}(z) - V_{\mathcal{W}_q}^{\pi}(\phi_q(z))\right| \le \Delta := \frac{\epsilon_r}{1-\gamma} + \frac{2\gamma R_{\max}\epsilon_p}{(1-\gamma)^2} \quad \forall z \in \mathcal{R}_q. \tag{17}$$

Proof. For compactness write $V_M(z) = V_{\mathcal{M}_q}^{\Gamma_q \pi}(z)$ and $V_S(s) = V_{\mathcal{W}_q}^{\pi}(s)$. Define the lifted small-world value $V_S^{\uparrow}(z) = V_S(\phi_q(z))$ and

$$d := \sup_{z \in \mathcal{R}_q} |V_M(z) - V_S^{\uparrow}(z)|. \tag{18}$$

The Bellman equations and Equation 14 give, for any $z \in \mathcal{R}_q$ with $s = \phi_q(z)$,

$$\begin{aligned} |V_M(z) - V_S(s)| &\le |r_q^{\Gamma,\pi}(z) - \bar{r}_q^{\pi}(s)| \\ &\quad + \gamma \left| \mathbb{E}_{z' \sim P_q^{\Gamma,\pi}(\cdot|z)} V_M(z') - \mathbb{E}_{s' \sim \bar{P}_q^{\pi}(\cdot|s)} V_S(s') \right| \\ &\le \epsilon_r + \gamma \mathbb{E}_{z' \sim P_q^{\Gamma,\pi}(\cdot|z)} |V_M(z') - V_S(\phi_q(z'))| \\ &\quad + \gamma \left| \mathbb{E}_{s' \sim \phi_{q\#} P_q^{\Gamma,\pi}(\cdot|z)} V_S(s') - \mathbb{E}_{s' \sim \bar{P}_q^{\pi}(\cdot|s)} V_S(s') \right|. \end{aligned} \tag{19}$$

The first continuation term is at most γd because $\mathcal{R}_q$ is closed under the decoded policy. For the second, Lemma 1, Equation 14, and $\|V_S\|_\infty \le R_{\max}/(1-\gamma)$ yield

$$\gamma \left| \mathbb{E}_{\phi_{q\#} P_q^{\Gamma,\pi}} V_S - \mathbb{E}_{\bar{P}_q^{\pi}} V_S \right| \le \frac{2\gamma R_{\max} \epsilon_p}{1-\gamma}. \tag{20}$$

Taking the supremum of Equation 19 over $z \in \mathcal{R}_q$ gives

$$d \le \epsilon_r + \gamma d + \frac{2\gamma R_{\max} \epsilon_p}{1-\gamma}. \tag{21}$$

Rearranging Equation 21 proves Equation 17. □

Theorem 1 (Query-sufficient decoded value preservation)**.** *Assume Definitions 1 and 2. Let the small-world planner return $\hat{\pi}$ satisfying*

$$V_{\mathcal{W}_q}^{\hat{\pi}}(\phi_q(z_0)) \ge V_{\mathcal{W}_q}^{\star}(\phi_q(z_0)) - \eta. \tag{22}$$

Then

$$V_{\mathcal{M}_q}^{\star}(z_0) - V_{\mathcal{M}_q}^{\Gamma_q \hat{\pi}}(z_0) \le \eta + \epsilon_\Gamma + \frac{2\epsilon_r}{1-\gamma} + \frac{4\gamma R_{\max} \epsilon_p}{(1-\gamma)^2}. \tag{23}$$

Proof. Let $s_0 = \phi_q(z_0)$, let $\pi^\dagger$ be the policy guaranteed by Definition 2, and let Δ be the fixed-policy discrepancy in Lemma 2. Insert the two small-world values $V_{\mathcal{W}_q}^{\pi^\dagger}(s_0)$ and $V_{\mathcal{W}_q}^{\hat{\pi}}(s_0)$ between the full-model values:

$$\begin{aligned} V_{\mathcal{M}_q}^{\star}(z_0) - V_{\mathcal{M}_q}^{\Gamma_q \hat{\pi}}(z_0) = &\underbrace{V_{\mathcal{M}_q}^{\star}(z_0) - V_{\mathcal{M}_q}^{\Gamma_q \pi^\dagger}(z_0)}_{\text{decoder coverage}} \\ &+ \underbrace{V_{\mathcal{M}_q}^{\Gamma_q \pi^\dagger}(z_0) - V_{\mathcal{W}_q}^{\pi^\dagger}(s_0)}_{\text{model discrepancy for } \pi^\dagger} \\ &+ \underbrace{V_{\mathcal{W}_q}^{\pi^\dagger}(s_0) - V_{\mathcal{W}_q}^{\hat{\pi}}(s_0)}_{\text{small-world planning gap}} \\ &+ \underbrace{V_{\mathcal{W}_q}^{\hat{\pi}}(s_0) - V_{\mathcal{M}_q}^{\Gamma_q \hat{\pi}}(z_0)}_{\text{model discrepancy for } \hat{\pi}}. \end{aligned} \tag{24}$$

The first term is at most ϵ_Γ by Equation 15. The second and fourth terms are each at most Δ by Lemma 2. For the third term, $V_{\mathcal{W}_q}^{\pi^\dagger}(s_0) \le V_{\mathcal{W}_q}^{\star}(s_0) \le V_{\mathcal{W}_q}^{\hat{\pi}}(s_0) + \eta$ by Equation 22; hence it is at most η. Therefore

$$V_{\mathcal{M}_q}^{\star}(z_0) - V_{\mathcal{M}_q}^{\Gamma_q \hat{\pi}}(z_0) \le \epsilon_\Gamma + 2\Delta + \eta. \tag{25}$$

Substituting Equation 17 into Equation 25 yields Equation 23. □

Corollary 1 (Finite-horizon form). *Consider an undiscounted horizon-H process with per-step rewards bounded by $R_{\max}$ and the same local errors ϵ_r, ϵ_p at each time. If the small-world policy class is ϵ_Γ-complete and the planner is η-optimal, then*

$$V^\star_{\mathcal{M}_q,H}(z_0) - V^{\Gamma_q \hat{\pi}}_{\mathcal{M}_q,H}(z_0) \le \eta + \epsilon_\Gamma + 2H\epsilon_r + 2R_{\max}H(H-1)\epsilon_p. \tag{26}$$

Proof. Let d_t be the worst full-versus-small-world value discrepancy with $H-t$ steps remaining and set $d_H = 0$. At time t, the small-world continuation value is bounded by $(H-t-1)R_{\max}$. Repeating the Bellman decomposition from Lemma 2 without discounting gives

$$d_t \le \epsilon_r + d_{t+1} + 2(H-t-1)R_{\max}\epsilon_p. \tag{27}$$

Summing Equation 27 for $t = 0, \dots, H-1$ yields

$$\begin{aligned} d_0 &\le H\epsilon_r + 2R_{\max}\epsilon_p \sum_{t=0}^{H-1}(H-t-1) \\ &= H\epsilon_r + R_{\max}H(H-1)\epsilon_p. \end{aligned} \tag{28}$$

Applying the four-term decomposition in Equation 24 contributes the fixed-policy discrepancy twice, plus $\eta + \epsilon_\Gamma$, which proves Equation 26. □

B.5 Static versus query-specific state-space separation

Proposition 1 (Static versus query-specific separation). *For any disjoint binary factor sets S_1 and S_2, there exists a workload partitioned into query sets $\mathcal{Q}_1$ and $\mathcal{Q}_2$ such that every deterministic state abstraction sufficient for all queries in $\mathcal{Q}_1 \cup \mathcal{Q}_2$ has at least $2^{|S_1 \cup S_2|}$ states. For each query $q \in \mathcal{Q}_i$, a query-specific state abstraction retaining only S_i has $2^{|S_i|}$ states. Consequently, the static-to-query-specific state ratio for queries in $\mathcal{Q}_i$ is $2^{|S_{3-i}|}$, and both ratios are at least $2^{\min(|S_1|,|S_2|)}$.*

Proof. Let the full state space be $\mathcal{Z} = \{0,1\}^{S_1 \cup S_2}$ and let the action set be $\{a_0, a_1\}$. Transitions are self-loops, so each query is a one-step decision problem and sufficiency reduces to preserving its optimal action. For every $i \in \{1,2\}$ and assignment $u \in \{0,1\}^{S_i}$, define a query $q_{i,u}$ with reward

$$r_{i,u}(z,a) = \mathbb{I}[z_{S_i} = u]\mathbb{I}[a = a_1] + \mathbb{I}[z_{S_i} \ne u]\mathbb{I}[a = a_0]. \tag{29}$$

Let $\mathcal{Q}_i = \{q_{i,u} : u \in \{0,1\}^{S_i}\}$. For query $q_{i,u}$, the unique optimal action is a_1 exactly when $z_{S_i} = u$, and it is a_0 otherwise.

Call a deterministic state abstraction ϕ sufficient for a set of queries if every query in the set admits an optimal policy that is constant on the fibers of ϕ. Take two distinct states z, z' that differ on at least one factor in $S_1 \cup S_2$. They differ on S_i for at least one i; choose $u = z_{S_i}$. Under $q_{i,u}$, the unique optimal action at z is a_1, whereas the unique optimal action at z' is a_0. If $\phi(z) = \phi(z')$, no policy that is constant on this fiber can choose both actions, contradicting sufficiency. Thus, a static abstraction sufficient for all queries in $\mathcal{Q}_1 \cup \mathcal{Q}_2$ must distinguish all $2^{|S_1 \cup S_2|}$ assignments.

For any $q \in \mathcal{Q}_i$, the query-specific state-abstraction map $\phi_q(z) = z_{S_i}$ has exactly $2^{|S_i|}$ states and is sufficient because Equation 29 depends only on z_{S_i}. Since $S_1 \cap S_2 = \varnothing$,

$$\frac{2^{|S_1 \cup S_2|}}{2^{|S_i|}} = 2^{|S_{3-i}|} \ge 2^{\min(|S_1|,|S_2|)}, \tag{30}$$

which proves the claim. □

B.6 Exact-match cache amortization

Proposition 2 (Exact-match cache amortization). *Suppose $M \ge 1$ queries have the same exact-match cache key and share query-independent compilation structure of construction cost $C_0 \ge 0$. Let query j have query-specific compilation and planning cost $C_j \ge 0$. Without caching, the average cost is $C_0 + \frac{1}{M}\sum_{j=1}^{M} C_j$. With exact-match caching, it is $\frac{C_0}{M} + \frac{1}{M}\sum_{j=1}^{M} C_j$. The average saving is $\frac{M-1}{M}C_0$. Any cache-key mismatch eliminates this saving and requires a cold compilation.*

Proof. Without caching, each query independently constructs the reusable structure, so the total cost is

$$C_{\text{cold}} = \sum_{j=1}^{M}(C_0 + C_j) = MC_0 + \sum_{j=1}^{M} C_j. \tag{31}$$

With exact-match caching, the reusable structure is constructed once and each query still pays its own query-specific cost:

$$C_{\text{reuse}} = C_0 + \sum_{j=1}^{M} C_j. \tag{32}$$

Dividing Equations 31 and 32 by M gives the two average costs. Their difference is $(1 - 1/M)C_0 = (M-1)C_0/M$. The exact-match condition is essential: if the data split, environment, full model, action library, compiler configuration, or access regime changes, the cached object is not valid and Equation 32 no longer describes the execution. □

B.7 Counted verification and confidence-gated fallback

Proposition 3 (Candidate-set verification monotonicity)**.** *Let candidates be ordered by a fixed proposal rule and let $Y_i \in \{0,1\}$ indicate whether candidate i succeeds when evaluated by the counted full-model verifier. For $A_m = \{\max_{1\le i\le m} Y_i = 1\}$, $\Pr(A_m)$ is nondecreasing in m without any independence assumption. Let $\mathcal{G}$ be the sigma-algebra generated by the fixed proposal diagnostics. If the Y_i are conditionally independent given $\mathcal{G}$ and $p_i := \Pr(Y_i = 1 \mid \mathcal{G})$, then, almost surely,*

$$\Pr(A_m \mid \mathcal{G}) = 1 - \prod_{i=1}^{m}(1 - p_i), \tag{33}$$

and the conditional marginal success probability contributed by candidate $m+1$ is

$$\Pr(A_{m+1} \mid \mathcal{G}) - \Pr(A_m \mid \mathcal{G}) = p_{m+1}\prod_{i=1}^{m}(1 - p_i). \tag{34}$$

Consequently, $\Pr(A_m) = 1 - \mathbb{E}[\prod_{i=1}^{m}(1 - p_i)]$.

Proof. The events are nested: $A_m \subseteq A_{m+1}$ because success among the first m candidates remains success among the first $m+1$. Probability monotonicity follows immediately, without invoking any distributional relation among candidates. Conditioning on $\mathcal{G}$,

$$\Pr(A_m \mid \mathcal{G}) = 1 - \Pr(Y_1 = 0, \ldots, Y_m = 0 \mid \mathcal{G}). \tag{35}$$

Conditional independence factors the joint failure probability as $\prod_{i=1}^{m}(1 - p_i)$, proving Equation 33. Subtracting the conditional expressions for $m+1$ and m proves Equation 34. Taking expectations of Equation 33 gives the unconditional identity by the tower property. Thus increasing m trades a precisely counted verification cost against a nonnegative, generally diminishing conditional probability of discovering an additional successful candidate. □

Proposition 4 (Confidence-gated fallback)**.** *Let the predefined fallback have value $V_0 \in [0, V_{\max}]$. Suppose the acceptance rule accepts a candidate only when that candidate has value at least V_0, except on a false-accept event of probability at most δ. Assume all candidate values lie in $[0, V_{\max}]$. Then the policy that executes the candidate on acceptance and the fallback otherwise has expected value at least $V_0 - \delta V_{\max}$.*

Proof. Let A denote acceptance and $F \subseteq A$ the false-accept event, with $\Pr(F) \le \delta$. On $A \setminus F$, the executed candidate has value at least V_0. On A^c, the policy executes the fallback and has value exactly V_0. On F, the candidate value is nonnegative. Therefore

$$\begin{aligned}
\mathbb{E}[V_{\text{exec}}] &\ge V_0 \Pr(A \setminus F) + V_0 \Pr(A^c) + 0 \cdot \Pr(F) \\
&= V_0\big(1 - \Pr(F)\big) \\
&\ge V_0 - \delta V_0 \\
&\ge V_0 - \delta V_{\max}.
\end{aligned} \tag{36}$$

This establishes the stated guarantee without assuming that candidate values are independent of the acceptance event. □

C BENCHMARK-ORIENTED METHOD DETAILS

C.1 SHARED IMPLEMENTATION CONTRACT

Every benchmark implementation follows Algorithm 1 and exposes the same six records: (i) the parsed query and seed factors; (ii) the factor set after dependency closure and any unresolved dependencies; (iii) the small-world state–action table and primitive decoder; (iv) the de-duplicated candidate set with explicit previous-plan and fallback identifiers; (v) the counted verification ledger; and (vi) the exact-match cache key and invalidation decision. The interface rejects evaluator-only fields, including oracle paths, hidden simulator state, ground-truth actions, online evaluator success, and reference policies. Every candidate is represented as a primitive action sequence or a closed-loop primitive controller with a stable hash and provenance record. The benchmark-specific differences below concern only how available observations are factorized, which dependency relations are available, and which executable small-world actions are admitted.

C.2 OFFLINE MULTI-AGENT CONSTRUCTION OF TASK PROGRAMS

Scope and inputs. The offline stage generates executable task programs, not a solution for each test query or a replacement for the full world model. Its inputs are the permitted state/action interface, supported task descriptions, and queries and transitions from the designated training worlds. The interface supplies typed observation fields and action identifiers; public task semantics specify what observable events count as completion. These inputs are explicit domain knowledge rather than task solutions discovered by the generator. Construction, repair-development, and final-check subsets are disjoint by world, so queries from the same world cannot cross these boundaries. Only construction examples and repair-development feedback enter generation; final-check worlds remain unseen until the bank is frozen. Official evaluation instances, hidden state, and evaluator-only test outputs are excluded. This restriction concerns supplied data and feedback, not an assertion that pretrained models have never encountered related benchmark descriptions.

Generated interface. A program bank $\mathcal{R}$ implements three operations, shown through a normalized interface:

```
parse_query(query, initial_state) -> task_record
update_progress(task_record, memory, state, action, next_state)
    -> updated_memory
select_references(task_record, memory, state) -> variable_refs
```

The parser emits action/object records, constraints, and instruction ordering. Completion routines evaluate observed action–state events and update the pending requirements; selection routines emit their explicit state-variable references and known current-stage prerequisites. The supplied state schema resolves these references to the compiler's factor vocabulary. Unmatched references or unsupported task forms return an unsupported status rather than a fabricated interpretation. The runtime implementations may therefore be grammar parsers, query-registry lookups, and conditional functions even though LLMs author their code offline.

Agent roles and execution feedback. The construction uses the Qwen3.5-35B, GLM-4-32B, and DeepSeek-V3 model pool. Exact checkpoints, provider aliases, decoding settings, and call limits are experiment identifiers to recover or specify for a new run, not inferred from these family names. A proposer writes or patches the three routines from the interface contract and construction examples. A semantic reviewer checks object binding, instruction ordering, allowed field access, and unsupported prerequisite claims. A counterexample designer proposes challenging training cases, initially from the task specification rather than the candidate code. The three roles rotate across models under a fixed schedule; this rotation is part of the reconstruction, not a claim about the unrecovered historical assignment. A separate Python test runner, not an LLM vote, executes the candidate in a resource-limited process with only the permitted interface. It returns failing inputs, observed outputs, and grounded expected behavior for the next patch. This proposal–execution–repair pattern draws on executable-model synthesis and multi-agent code testing (Tang et al., 2024; Dainese et al., 2024; Huang et al., 2023).

Grounded checks and acceptance. Expected completion labels come from permitted task semantics and observed transition events, checked independently of the proposed program. Agent-authored

tests without such a reference remain hypotheses rather than acceptance evidence. The runner checks syntax, output types, field permissions, termination, object binding, and progress updates, including failed actions and actions on non-target objects. Semantics-preserving renaming tests change object identifiers and their bindings consistently; compositional tests vary supported conjunction and ordering without changing the meaning of atomic actions. Candidate state changes proposed by an agent are used only after a training-environment execution or a validated state constructor establishes that they are admissible. Generation cannot rewrite the reference checker, task objective, or protected constraints. A candidate must pass the required semantic and access tests; among eligible programs, selection uses task completion on repair-development worlds under a fixed ASW planning budget, with fewer full-model calls and shorter code as tie-breakers. The planner, action library, dependency artifact, and verification settings remain fixed during this comparison. Every repair reruns the accumulated regression suite, and a failed patch leaves the incumbent unchanged. A predetermined call limit ends generation; unsupported coverage is reported rather than filled by untested rules. The selected bank is frozen before the separate final check, whose failures are reported rather than fed back into the same run.

Illustrative rule and its tests. For a query requesting that door d be opened, a readable rule record is:

```
operation: open(d)
complete: action == toggle and front_before == d
          and not open_before(d) and open_after(d)
extra_refs: if locked(d) and no_matching_key_is_carried
            then {matching_key_location, carrying}
```

This record illustrates the intended interface, not recovered source code. The program binds d to the queried object and reads the corresponding observable fields; it does not call an LLM during execution. Tests reject completion when another door opens or a toggle fails, and distinguish a matching key from a wrong key. Sequence tests ensure that a later action is not credited before its branch becomes active. The extra references identify missing planning information, not a prescribed route for obtaining the key. The rule is parameterized by the target object rather than a fixed map coordinate or test-instance identifier.

Connection to the audited dependency bank. Program generation and numerical dependency estimation are distinct operations. In the audited BabyAI/QueryStress configurations, a separate builder counts initiation/effect associations in two-to-four-action windows of training trajectories; ASW traverses the supported links after query/progress seeding. Those traces use `BabyAIBot` and an exact A* reference solver offline, whose contribution is retained in the provenance record. The resulting links are observational support, not causal facts certified by agent agreement. Craftax instead obtains its main prerequisite graph from declared resource, tool, and station requirements. LLM authorship of the task routines does not change these evidence sources or establish execution of an additional learned relevance estimator.

Deployment, provenance, and scope. Online planning loads the frozen programs and dependency artifacts without generator calls; executing the programs initializes the factors used by the existing compiler, decoder, verification gate, and fallback. Code and configuration hashes identify the bank and invalidate incompatible caches. The reconstruction records prompts, responses, model identifiers, split manifests, patches, human edits, test outcomes, and the final source hash. Generation tokens, wall time, and training-simulator calls are reported separately from online transition counts and runtime. A regenerated bank with a different implementation is a new configuration and cannot inherit the historical result tables without corresponding evaluation. The common generation protocol targets reusable rules within a declared task interface; transfer beyond that vocabulary, sensitivity to the generator, and advantages of multiple agents require separate evidence. Finite testing does not certify universal rule correctness or establish the assumptions of the value-gap theorem.

C.3 Dependency representation and structured priorities

The BabyAI/QueryStress runtime graph contains token-expanded semantic tags from task and option records; these tags are not independent physical state variables. Formal factor sets refer to the state fields represented by selected records, rather than treating each token as a separate coordinate of F. Training support, confidence, and lift filter the edges; incoming links are followed for at most three hops, with at most six predecessors per target. A retained record at depth d has priority $1/(1+d)$.

This score orders supported options; it is not a learned estimate of a factor's influence on future value. Craftax instead traverses declared producer–requirement relations and ranks by reverse topological route position. Its Wilson-bound uncertainty is a separate execution diagnostic, not a per-factor relevance term.

C.4 Compiler records, priorities, and verification

From program outputs to factor indices. The functions `parse_query`, `update_progress`, and `select_references` in Appendix C.2 describe the logical task-program interface. Their records contain object bindings and references to fields in the supplied state schema; resolving these references gives indices in $[K] = \{1, \dots, K\}$. Let G_q, C_q denote goal and constraint indices, Ξ_q the permitted hypothetical assignments, and $E_q(z_0)$ the references returned for unfinished requirements and current-stage prerequisites. Then

$$W_q^{(0)} = G_q \cup C_q \cup \mathrm{vars}(\Xi_q) \cup E_q(z_0). \tag{37}$$

The notation expresses the compiler interface; it does not assert that each implementation serializes a separate array under each symbol. For ordered tasks, the progress memory is part of z_0 rather than reconstructed from its physical fields alone.

Record-level dependency traversal. Let $S_q^{(0)}$ be the task/action descriptors associated with the initial selection. The runtime graph connects such descriptors, not necessarily the K physical coordinates. Writing D_{uv} for retained evidence on an edge $u \to v$, and $\mathrm{Req}_q(S)$ for explicitly declared prerequisite, hazard, constraint, and edit descriptors associated with S, its abstract traversal is

$$S_q^{(d+1)} = S_q^{(d)} \cup \{u : \exists v \in S_q^{(d)},\ D_{uv} > 0,\ D_{uv} \geq \tau_D\} \cup \mathrm{Req}_q(S_q^{(d)}). \tag{38}$$

After bounded traversal, W_q includes the state-field references of the retained bound records, in addition to the protected seed fields. A descriptor can correspond to multiple fields, and a token has no independent state coordinate merely because it is a graph node. The projection in Equation 2 acts on these fields, not on the words used to retrieve records. This distinguishes the formal state representation from the semantic index used to select actions.

For BabyAI/QueryStress, the graph is built from starting-condition, action-tag, and effect associations in two-to-four-action windows of training traces. The traces use the offline reference planners documented in Appendix C.2. Retained options require at least ten occurrences, five worlds, five trajectories, and a public-state-change rate of at least 0.60; this rate is not terminal task success. Serialized edges require joint count at least ten, five worlds, and conditional support at least 0.20. Additional token-expanded edges require lift at least 1.20; at most six predecessors per target are followed to depth three. For a retained descriptor v, the priority is

$$\rho_q(v) = \frac{1}{1 + d_q(v)}, \tag{39}$$

where $d_q(v)$ is the shortest predecessor distance from a protected seed. Option selection uses depth, empirical public-change rate, distinct-world support, occurrence count, action length, and a stable identifier in that order. Craftax instead traverses declared producer–requirement edges and uses reverse topological route position. Neither implementation fits per-factor transition or value sensitivities for this selection.

Predictions, evidence, and the planning objective. The small-world symbols describe the selected action records' prediction and execution interface; they do not imply that a dense table over all reduced states is materialized. Declared action updates or empirical outcome records provide successor predictions, while public task semantics or permitted reward statistics provide query rewards. Evidence counts measure support for those records, not the probability of solving the current query. For BabyAI/QueryStress, candidate uncertainty is $1/(2+n)$ for candidate support n; working-set uncertainty is $1/(1+N_+)$ for the number N_+ of positive-support candidates. Craftax uses one minus the goal-effect Wilson lower bound as a working-set uncertainty diagnostic. These diagnostics need not coincide with the generic per-state/action quantity U_q in every implementation.

The general finite-search objective is

$$\widehat{J}_q(\bar{\tau}) = \sum_{t=0}^{H-1} \gamma^t \big[\bar{r}_q(\bar{z}_t, b_t) - \lambda_U U_q(\bar{z}_t, b_t) - \lambda_C \bar{c}_q(\bar{z}_t, b_t)\big], \tag{40}$$

with fixed nonnegative weights and $\bar{\tau} = (\bar{z}_0, b_0, \ldots, \bar{z}_H)$. Here H counts primitive steps after unrolling controllers, as in the value analysis; options grouped into edges instead require duration-aware rewards and discounting (Sutton et al., 1999). The structured priorities above are the documented implementation rules; Equation 40 is the general reward–cost–uncertainty objective, not evidence that every frozen configuration evaluates an identical numerical score. The planning-error term in the value theorem concerns unpenalized reward and includes any loss introduced by search heuristics or cost/uncertainty penalties.

Candidate and gate interface. A primitive prefix is a finite nonempty action sequence; a controller can generate such a prefix during simulation and execution. The logical candidate set is

$$\mathcal{P}_q = \{\tau_q^{\mathrm{sw}}, \tau_q^{\mathrm{fb}}\} \cup \mathcal{P}_q^{\mathrm{prev}} \cup \mathcal{P}_q^{\mathrm{div}}, \tag{41}$$

where $\mathcal{P}_q^{\mathrm{prev}}$ is empty when no previous plan is available, and $\mathcal{P}_q^{\mathrm{div}}$ contains optional alternatives. Candidates refer to the same planning root and carry their action/controller identity and evidence. The generic diagnostic record is

$$d_q(\tau) = \big(n_{\min,q}(\tau), u_{\max,q}(\tau), \mathrm{margin}_q(\tau), \mathrm{prog}_q(\tau), \widehat{c}_{\mathrm{v},q}(\tau)\big). \tag{42}$$

Its fields summarize support, uncertainty, proposal-score separation, progress measured by the task programs, and estimated verification calls; implementations retain the applicable fields rather than treating an absent diagnostic as measured evidence. The fixed ranking function s_θ and acceptance predicate h_θ act on this record and the execution-status observations. The benchmark configuration supplies their numerical thresholds and fallback controller; these are not learned by online LLM calls. Verification starts from a copy of the root and updates a separate candidate-local task memory. A failed simulation or budget-truncated prefix is not presented as a fully checked proposal. This acceptance decision does not evaluate the full-model optimal value and does not by itself establish the value-relative premise of Proposition 4.

Cache identity and online evidence. The query-independent structure uses the key

$$\begin{aligned}\kappa_{\mathcal{C}} = (&\mathrm{split}, \mathrm{environment}, \mathrm{full\ model}, \mathrm{action\ library},\\ &\mathrm{compiler\ config}, \mathrm{access\ regime}),\end{aligned} \tag{43}$$

whose library/configuration hashes include the frozen task programs and dependency artifact. State-dependent values, plans, acceptance decisions, and ledgers are not reusable query-independent structure. QueryStress counterfactual landmarks require additional same-world, query-order, and enforced-constraint compatibility; otherwise they must be rediscovered through counted interaction. The repair route can recompile once from the unchanged root using only unused budget, and both passes are charged. This evidence is distinct from the frozen training dependency graph.

C.5 Instantiation map

Table 3: Benchmark-oriented realizations. The final column exists only to connect the manuscript to code and configuration files; these identifiers are not method names or additional baselines.

Benchmark	Query seeds	Factors added by dependency closure	Executable planning and verification	Configuration ID
BabyAI	mission predicates and current observable task progress	object–location, key–door, containment, ordering, and mission preconditions	state–option graph; at most 24 candidates; top-two primitive-prefix verification; exact-match cache	`structured-u3`
QueryStress	query predicates, hazards, counterfactual edits, and current observable task progress	prerequisites, safety constraints, path costs, distractor dependencies, and key–door order	state–option graph; counted recompile after observed counterfactual blockage; previous plan included in verification	`asw_u3_cc_v2_cf_repair`
Craftax	predeclared goal, inventory, resource phase, and achievement progress	resource prerequisites, phase-dependent action eligibility, and goal-specific fallback conditions	phase-aware small-world plan and goal-specific fallback verified under one transition ledger	`resource_u45_mvp_routed2`

For the statistical tables below, *base ASW configuration* denotes QueryStress configuration `structured-u3`, which applies dependency closure without the additional hazard, path-cost, key–door-order, and previous-plan features. The *resource-only ASW configuration* denotes Craftax configuration `resource-u4.3`, which uses the shared task graph and an executable macro without resource-phase state or rollback-aware termination. These descriptions distinguish controlled configurations within ASW; they are not additional reader-facing methods.

C.6 BabyAI compiler

The BabyAI factor interface records the agent pose, visible object identities and locations, carried object, door state, key possession, and mission progress. The mission parser maps compositional instructions to goal predicates and explicit ordering constraints. Dependency closure restores factors that may not be stated in the mission but are required by an executable route, such as a key and the state of its corresponding door. All structured methods use the same training-only action library, induced precondition graph, shared factor-to-landmark decoder, and counted transition interface. The seven evaluated levels are `GoToObj`, `GoToLocal`, `PickupLoc`, `PutNextLocal`, `OpenDoor`, `Unlock`, and `MiniBossLevel`. Frozen compiler limits are: minimum dependency count 10, confidence 0.20, lift 1.20, at most six predecessors, closure depth three, 48 active options, four options per schema, 24 candidates, top-two verification, route-bank limit 64, route confidence 0.75, replan interval eight, and primitive horizon 256.

C.7 QueryStress compiler

QueryStress is a controlled benchmark derived from MiniGrid-style object and navigation semantics (Chevalier-Boisvert et al., 2023). Each environment instance contains walls, doors, keys, hazards, goals, and distractor objects, while multiple queries alter which factors determine success. The five map types are four-room, door–key maze, lava–key maze, locked multi-room, and obstructed corridor. The ten query types cover direct goals, key-required execution, safety constraints, counterfactual blockage, distractor targets, distractor invariance, shortest-path objectives, ordered door opening, and composite combinations. This construction separates query adaptivity from environment variation: methods receive the same observable structured environment for each paired instance but must compile a different small world when the query changes.

The query-based factor mask retains factors specified by the query and current observable task progress. The static abstraction retains one query-independent factor–option graph. The final ASW configuration retains hazards, path costs, explicit key–door order, and a conservative previous plan. For counterfactual queries, an observed blocked cell may trigger one recompile from the same initial state using only the unused transition budget. Both passes are fully charged, and candidate selection uses the same observable support criteria.

C.8 Craftax symbolic compiler

The Craftax protocol uses observable resource counts, inventory, achievement progress, and goal identity; it does not expose hidden evaluator state or oracle action sequences. The 12 goals are: collect wood, place table, make wood pickaxe, collect stone, make stone pickaxe, place furnace, collect coal, collect iron, make iron pickaxe, collect diamond, make iron sword, and defeat skeleton. The primitive decoder realizes small-world resource and crafting actions through executable action suffixes or closed-loop controllers. The static abstraction and query-based factor mask use the same observable resource graph and decoder. The former retains one query-independent graph, whereas the latter retains only factors selected directly from the goal and current observable task progress. The final ASW configuration evaluates a phase-aware small-world plan and, for predeclared goals, a complementary fallback plan from the same initial state. All candidate rollouts share one transition ledger, are verified independently, and are selected using only observable completion score, verification status, decoded length, and a deterministic tie-break. This protocol is designed to test prerequisite composition under Craftax mechanics; it is not presented as a score on the standard open-ended Craftax leaderboard.

C.9 Exact-match cache keys and invalidation

Reuse is restricted to query-independent compilation structure. The exact-match key binds the data split, environment identity, full-model or checkpoint hash, action-library hash, compiler source and configuration hash, and access regime. A query-specific factor set, candidate set, verification result, or selected action is never reused. A cold start, explicit invalidation, environment change, or any key mismatch forces a cache miss. Thus, cache-key matching is enforced for correctness rather than treated as an optional performance heuristic.

D Experimental Protocol and Access Regimes

D.1 Benchmark and access summary

Table 4: Planner-facing access, evaluation size, budgets, and paired statistical units.

Evaluation	Permitted online access	Instances	Budgets	Cluster unit
BabyAI	structured mission and observable state; counted generative transitions	7 levels, 175 environment instances, 2,625 method–budget records	64, 256, 1024	environment instance
QueryStress	structured map, observable state, and query; counted generative transitions	frozen final: 5 map types, 10 query types, 100 environment instances, 18,000 rows; selection matrices: 50 instances	64, 256, 1024	environment instance
Exact-match cache intervention	paired structured state and frozen cache key	150 paired cache keys in 50 environment instances	1024	environment instance
Craftax symbolic	observable resource/inventory state and goal identity	12 goals, 30 paired seeds per goal in the configuration matrix	256, 1024, 4096	environment seed

All registered supplemental matrices have complete paired coverage and zero recorded execution failures; the QueryStress final export is hash-matched to its frozen 18,000-cell manifest. The four displayed Craftax compiler configurations each aggregate 360 episodes per budget, corresponding to $4 \times 3 \times 360 = 4{,}320$ displayed method–budget–episode records. The primitive-planning reference rows come from a separate study and are not assigned this denominator. Access regimes are analyzed separately and are never pooled into one numerical benchmark score. This distinction is essential because structured state access, observable symbolic access, and learned-model rollouts expose different planning interfaces.

D.2 Metrics, transition accounting, and inference

For a registered compiler matrix, let $(B_k, s_{m,k})$ denote declared transition cap B_k and success $s_{m,k} \in [0,1]$ for method m. The predeclared endpoint linearly interpolates success over $\ell_k = \log_2 B_k$ and computes

$$\mathrm{AUC}_B(m) = \frac{1}{\ell_K - \ell_1} \int_{\ell_1}^{\ell_K} \widetilde{s}_m(\ell)\, d\ell. \tag{44}$$

This declared-budget AUC preserves exact paired support even when methods terminate early; mean realized counted transitions are reported separately. The original BabyAI and primitive-baseline matrices retain their previously declared frontier analyses, and no endpoint is pooled across matrices. The inferential unit is the paired per-cluster difference within one frozen matrix. The transition ledger charges every full-model transition used for tree expansion, rollout scoring when required by the method, candidate verification, and fallback evaluation. Compilation time and wall-clock time are reported only when measured under a common systems protocol and are not substituted for transition accounting.

For the equally spaced log-budget grids used here,

$$\mathrm{AUC}_B(m) = \tfrac{1}{4} s_{m,1} + \tfrac{1}{2} s_{m,2} + \tfrac{1}{4} s_{m,3}. \tag{45}$$

The recent-method operating-point summaries use

$$\overline{\Delta s}_m = \frac{100}{3} \sum_{d=1}^{3} (s_{\mathrm{ASW},d} - s_{m,d}), \qquad \overline{\rho}_m = \frac{100}{3} \sum_{d=1}^{3} \left(1 - \frac{t_{\mathrm{ASW},d}}{t_{m,d}}\right), \tag{46}$$

Table 5: Definitions of every baseline appearing in the main paper. External planning algorithms and controlled ASW ablations are distinguished explicitly.

Baseline	Class	Exact role in this paper	Reference
Best-first search	external search	Expands full-model states in ascending $f = g + h_q$ priority until success or budget exhaustion; it uses primitive actions, heuristic weight 1.0, and the same transition source as ASW.	Hart et al. (1968)
MCTS	external search	Allocates the declared transition budget to tree simulations over primitive actions using the same full-model transition source; it receives no compiled options, cache entries, or policy prior.	Kocsis & Szepesvári (2006)
Primitive-action MCTS	external search	The Craftax realization of the same comparator class; every tree edge is a primitive environment action rather than a compiled option or macro.	Kocsis & Szepesvári (2006)
Goal-conditioned PPO-RNN	learned policy	A fixed recurrent policy trained with PPO; at evaluation it receives the same observable symbolic state and goal, executes primitive actions, and performs no test-time world compilation.	Schulman et al. (2017)
Abstract WM	learned abstraction	Access-matched adaptation of the value-preserving abstract-MDP method: learn one abstract model over the shared option set, then plan without query-time factor recompilation. Offline learning cost is reported separately.	Rodriguez-Sanchez & Konidaris (2024)
VisualPredicator	learned abstraction	Structured-access adaptation that retains predicate and operator learning while restricting observations and simulator access to the same interface used by ASW.	Liang et al. (2025)
SLAP	option learning	Learns shortcut options over the same base abstract planning graph; training interactions are separated from evaluation, and every test-time model transition is charged.	Liu et al. (2026)
Static abstraction	controlled ablation	Uses the same action library, primitive decoder, and transition source as ASW, but retains one query-independent factor–option graph for every query.	related principles: Ferns et al. (2011); Grimm et al. (2020)
Query-based factor mask	controlled ablation	Retains factors specified by the query and current observable task progress, without dependency closure, alternative candidate generation, full-model verification, or exact-match caching.	paper-defined control

where d indexes the three benchmark columns of Table 1 and t is mean counted transitions. The first quantity is in percentage points and the second is a percentage; both are descriptive summaries of the displayed rounded operating points. Confidence intervals and paired tests are not reconstructed from those rounded means.

Structured comparisons are paired by query within each environment instance. Craftax comparisons are paired by goal and episode seed, with inference clustered by environment seed. Confidence intervals use 10,000 cluster-bootstrap resamples, and paired tests use the frozen draw count for each matrix (100,000 sign flips in the supplemental selections). Holm adjustment is applied within each predeclared set of comparisons.

D.3 Baseline definitions and fairness controls

Table D.3 presents the complete definitions of the baselines mentioned in the paper. External baselines answer whether compilation outperforms conventional primitive planning under the registered transition interface. For BabyAI and QueryStress, the best-first search and MCTS receive the same declared structured state and counted generative transition source as ASW; neither receives the ASW option graph, verification diagnostics, or cache. For Craftax, primitive-action MCTS and PPO-RNN receive the same observable symbolic state and goal identity, and each executed or simulated primitive transition is charged under the benchmark-specific ledger. The two controlled baselines test whether selecting factors separately for each query and restoring their dependencies improves over one static abstraction or a query-based factor mask when the full model and primitive decoder are held constant. The Craftax implementations are benchmark-specific versions of these two controls, not additional methods.

E Complete Results and Statistical Tests

E.1 Paired frontier and intervention effects

Table 6: Measured paired effects used for inferential claims. The QueryStress rows report five predeclared comparisons on the frozen 100-instance final matrix; the Craftax rows retain the separate primitive-baseline matrix. Complete operating points are reported in Tables 8 and 11.

Evaluation	Comparison	Endpoint	Effect	95% CI	Holm p
BabyAI	vs. best-first search	frontier AUC	$+0.3703$	$[0.3258, 0.4146]$	0.0004
	vs. MCTS	frontier AUC	$+0.3840$	$[0.3355, 0.4335]$	0.0004
	vs. static abstraction	frontier AUC	$+0.1069$	$[0.0641, 0.1514]$	0.0004
QueryStress frozen final	vs. base ASW configuration	declared-budget AUC	$+0.0863$	$[0.0383, 0.1365]$	0.00065
	vs. query-based factor mask	declared-budget AUC	$+0.1498$	$[0.1018, 0.2005]$	< 0.0001
	vs. static abstraction	declared-budget AUC	$+0.1005$	$[0.0655, 0.1360]$	< 0.0001
	vs. best-first search	declared-budget AUC	$+0.5658$	$[0.5048, 0.6253]$	< 0.0001
	vs. MCTS	declared-budget AUC†	$+0.5468$	–	–
Exact-match cache intervention	cache enabled − disabled	success ITT	$+0.1267$	$[0.0667, 0.1933]$	0.0016
Craftax original matrix	vs. primitive-action MCTS	frontier AUC	$+0.5958$	$[0.5385, 0.6563]$	0.0009
	vs. goal-conditioned PPO-RNN	frontier AUC	$+0.7083$	$[0.6583, 0.7583]$	0.0009

E.2 BabyAI budget curves

Table 7: BabyAI success and mean counted transitions across all registered budgets.

	$B = 64$		$B = 256$		$B = 1024$	
Method	Succ.	Trans.	Succ.	Trans.	Succ.	Trans.
ASW	**0.857**	**14.5**	**0.869**	**24.0**	**0.869**	113.0
Query-based factor mask	0.851	18.1	0.857	45.0	0.857	**94.3**
Static abstraction	0.743	24.6	0.754	72.2	0.760	124.5
Best-first search	0.291	54.4	0.571	151.5	0.669	429.0
MCTS	0.143	58.7	0.354	191.1	0.549	589.5

E.3 Frozen QueryStress final matrix

Table 8: Frozen QueryStress final-matrix results across registered budgets (1,000 paired query–environment instances per method and budget; 100 environment instances and ten query types). The primary endpoint is declared-budget AUC; mean counted transitions include proposal and verification calls.

	$B = 64$		$B = 256$		$B = 1024$	
Method	Succ.	Trans.	Succ.	Trans.	Succ.	Trans.
ASW	**0.621**	39.3	**0.640**	**100.3**	**0.640**	227.8
Base ASW configuration	0.526	**35.1**	0.554	110.7	0.562	**226.1**
Query-based factor mask	0.472	37.0	0.490	121.9	0.490	241.6
Static abstraction	0.510	42.7	0.543	113.9	0.543	277.0
Best-first search	0.000	64.0	0.050	250.8	0.178	927.3
MCTS	0.005	63.9	0.058	249.9	0.233	900.9

The final matrix was created from a frozen manifest before outcome aggregation and contains all six registered methods, all three budgets, and all ten query types with zero execution failures. Relative to the base ASW configuration, the query-based factor mask, static abstraction, best-first search, and MCTS, ASW improves declared-budget AUC by $+0.0863$, $+0.1498$, $+0.1005$, $+0.5658$, and $+0.5443$, respectively. The corresponding 95% cluster-bootstrap confidence intervals and five-comparison Holm-adjusted tests appear in Table 6. At $B = 1024$, ASW improves success by $+0.097$

over static abstraction while reducing counted transitions by 17.8% (transition ratio 0.822, 95% CI $[0.741, 0.927]$).

E.4 QUERYSTRESS CONFIGURATION SELECTION

Table 9: Configuration-matched QueryStress results on 500 paired query–environment instances per method and budget. The final ASW configuration is compared with the base ASW configuration and the two controlled abstractions under one counted transition ledger.

	$B = 64$		$B = 256$		$B = 1024$	
Method	Succ.	Trans.	Succ.	Trans.	Succ.	Trans.
ASW	**0.630**	34.9	**0.670**	**91.9**	**0.670**	240.8
Base ASW configuration	0.548	**34.0**	0.580	102.3	0.598	**200.9**
Query-based factor mask	0.496	36.4	0.532	115.1	0.532	227.7
Static abstraction	0.550	41.8	0.584	110.8	0.584	277.9

The final ASW configuration improves declared-budget AUC by $+0.0835/+0.1370/+0.0845$ relative to the base ASW configuration, the query-based factor mask, and static abstraction. At $B = 1024$, the corresponding success gains are $+0.072/+0.138/+0.086$, and the transition ratio to static abstraction is 0.8668.

E.5 EXACT-MATCH CACHE INTERVENTION

Table 10: Pre-specified cache intervention at $B = 1024$ (150 paired keys in 50 environment instances).

Method	Success	95% CI	Mean transitions
ASW, cache enabled	0.600	$[0.493, 0.700]$	226.4
ASW, cache disabled	0.473	$[0.367, 0.580]$	262.7
Static abstraction	0.527	$[0.420, 0.633]$	305.6

The paired cache-enabled minus cache-disabled intention-to-treat effect is $+0.1267$, with 95% CI $[0.0667, 0.1933]$ and Holm-adjusted $p = 0.0016$.

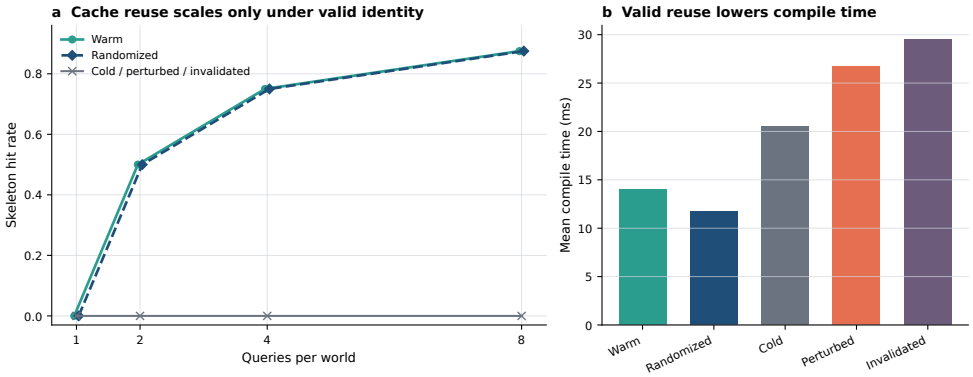

Figure 3: **Exact-match cache keys determine when reuse is valid.** With one, two, four, or eight queries sharing the same environment/model/compiler key, the query-independent compilation structure is built once and reused, giving hit rates 0, $1/2$, $3/4$, and $7/8$. Randomizing query order preserves the rate, whereas changing the environment identity or explicitly invalidating the key forces zero hits. Panel (b) reports compilation time under the same controls. The intervention verifies cache-key matching and amortization; it does not reuse query-specific decisions or transfer state across environment instances.

E.6 CRAFTAX SUCCESS–BUDGET CURVES AND COMPILER COMPARISONS

Table 11: Goal-conditioned symbolic Craftax operating points. The upper block is the configuration-matched compiler comparison, with 360 episodes per method and budget. The lower block retains primitive-planning reference results from a separate study; those rows are not part of the matched compiler analysis. Bold entries compare only the compiler configurations.

	$B = 256$		$B = 1024$		$B = 4096$	
Method	Succ.	Trans.	Succ.	Trans.	Succ.	Trans.
ASW	0.7417	112.6	**0.8167**	**217.0**	**0.8167**	**291.5**
Resource-only ASW configuration	**0.7611**	101.1	0.7639	271.6	0.7639	825.8
Query-based factor mask	0.7333	**96.4**	0.7333	256.1	0.7333	642.6
Static abstraction	0.7556	103.3	0.7611	275.3	0.7611	842.5
Primitive-action MCTS	0.0708	249.1	0.1167	940.6	0.1621	1563.2
Goal-conditioned PPO-RNN	0.0083	155.4	0.0083	161.0	0.0083	161.0

The final ASW configuration improves the declared-budget AUC by $+0.0347/+0.0646/+0.0382$ relative to the resource-only ASW configuration, the query-based factor mask, and static abstraction. At $B = 4096$, the corresponding success gains are $+0.0528/+0.0833/+0.0556$; the transition ratio to static abstraction is 0.3460. The primitive-action MCTS and PPO-RNN rows come from the original primitive-baseline matrix and are included for benchmark-level context rather than pooled compiler statistics.