Identifiable Protocol States for Cross-Environment GUI Agents

Anonymous ACL submission

Abstract

GUI agents succeed on familiar interfaces yet fail when layout, navigation order, or recovery paths change across applications. We trace this brittleness to a missing abstraction: current agents learn environment-specific action policies without an explicit representation of the task stage shared across interfaces. We introduce *Identifiable Protocol States* (IPS): task-state equivalence classes whose members may look different yet must share the same preconditions, admissible next subgoals, and recovery structure. This criterion is operationally testable and distinguishes protocol states from generic latent clusters. We formalize a factorization of GUI execution into a transferable protocol layer and an environment-specific grounding layer, prove a sample-complexity benefit of this factorization for cross-environment adaptation, and instantiate it with four components: cross-environment state matching, invariant mining, protocol-constrained planning, and recovery compilation. On SPA-Bench, AndroidWorld, and GUI-KRB, IPS improves task success, recovery behavior, and low-demonstration adaptation over policy-only and planner-only baselines. Gains concentrate where the theory predicts they should: unseen environments (+5.0 pp), cross-application transfer (+5.3 pp), and structured failure recovery (+11.8 pp). The code is available in `https://anonymous.4open.science/r/GUI_explorer-4E8D`.

1 Introduction

Consider a user who books flights on two different travel websites. Both sites require the same sequence of decisions: select dates, choose a fare, enter passenger details, confirm payment. Yet every button, layout, and navigation path differs. A human completes both tasks effortlessly because she recognizes the shared *task protocol*—the abstract sequence of preconditions, subgoals, and recovery options that is invariant across interfaces. Current GUI agents lack this recognition. They learn action policies tied to specific screen layouts, so a relocated button, an unfamiliar confirmation dialog, or a mid-task error on a new site can cause complete failure even when the underlying task structure is identical (Chen et al., 2024; Rawles et al., 2024b) (Figure 1).

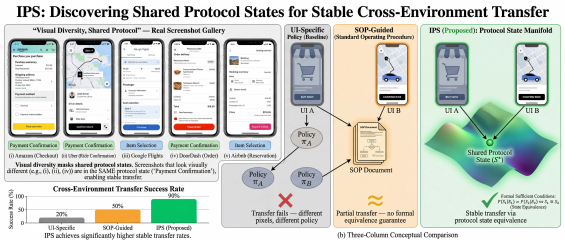

Figure 1: Comparison between UI-specific policy matching and protocol-state equivalence. Standard agents trigger unrelated action policies on visually distinct screens; IPS treats both as the same protocol state, enabling shared precondition checking, next-subgoal planning, and recovery compilation.

This brittleness is not merely a perception problem. Recent work has substantially improved visual grounding (Hong et al., 2024; Xu et al., 2024), exploration coverage (Xie et al., 2025; He et al., 2025), and planning with procedural guidance (Li et al., 2025b). Yet cross-environment transfer remains fragile because these methods represent execution at the wrong level of abstraction. Reactive policies condition on the current screenshot without modeling what task stage it corresponds to. Hierarchical planners decompose goals into subgoals but do not require that the same subgoal structure be shared across environments. Exploration-based agents mine useful interaction knowledge but store it in environment-specific form. None defines an object that captures *why* two visually different screens should be treated identically for planning and recovery.

We argue that the missing object is a **protocol state**: a task-state equivalence class whose

members may look different yet share three operational signatures—the same precondition profile, the same admissible next-subgoal class, and the same recovery structure. The definition is falsifiable: if two observations assigned to the same state admit different next subgoals or recovery options, the assignment is wrong. This falsifiability is what distinguishes protocol states from generic latent clusters or planner stages, which carry no testable commitment.

Building on this definition, we propose **IPS** (Identifiable Protocol States), a framework that factorizes GUI execution into two layers. The *protocol layer* infers task-state equivalence classes and legal transitions from trajectories collected across multiple environments. The *grounding layer* maps the current protocol state to environment-specific actions. We prove that this factorization decouples the sample complexity of cross-environment adaptation from the complexity of protocol learning, so adapting to a new interface requires updating only the grounding parameters. Four components implement the idea: a cross-environment state matcher that aligns observations sharing task-level semantics, an invariant miner that estimates precondition and transition constraints, a protocol-constrained planner that reasons over protocol transitions rather than raw actions, and a recovery compiler that maps protocol-level failures to structured corrective decisions.

We make three contributions:

- We define *protocol states* as task-state equivalence classes characterized by shared preconditions, next-subgoal semantics, and recovery structure, providing an operationally testable transfer object for GUI agents.
- We formalize the protocol–grounding factorization and prove that it yields a strictly smaller adaptation sample complexity than monolithic policies, predicting the asymmetric transfer pattern observed empirically.
- We instantiate the framework as IPS and show that it improves over strong baselines on SPA-Bench, AndroidWorld, and GUI-KRB, with gains concentrated in unseen environments, cross-application transfer, and recovery—precisely where the factorization theory predicts the largest benefit.

2 Related Work

Reactive policies and benchmarks. Direct policy learning and prompting dominate current GUI agents. ReAct-style prompting (Yao et al., 2023) couples textual reasoning and acting and underlies AppAgent (Zhang et al., 2025a), AutoDroid (Wen et al., 2024), CogAgent (Hong et al., 2024), DiGiRL (Bai et al., 2024), M3A (Rawles et al., 2024a), MobileAgentV2 (Huang et al., 2024), SeeAct (Zheng et al., 2024), Aguvis (Xu et al., 2024), Aria-UI (Yang et al., 2025), UGround (Gou et al., 2024), and Ponder&Press (Wang et al., 2025). SPA-Bench (Chen et al., 2024), AndroidWorld (Rawles et al., 2024b), and WebArena (Zhou et al., 2024) provide evaluation grounds; newer benchmarks stress transfer (Lu et al., 2025), mobile realism (Zhang et al., 2025b), UI perturbation (Chen et al., 2025b), and cross-application workflows (Li et al., 2026). Liu et al. (2024) survey the field. All such methods model execution as $\pi(a_t \mid o_{\leq t}, g)$, which carries no mechanism to determine whether two visually distinct observations correspond to the same task stage—so layout changes invalidate the policy even when task structure is preserved.

Planning, SOP guidance, and exploration. A second line introduces explicit task structure. ChatSOP (Li et al., 2025b) injects standard operating procedures via MCTS; CompileAgent (Li et al., 2025a) and OS-Genesis (Sun et al., 2025) integrate tool use and reverse task synthesis; AdaptAgent (Verma et al., 2025) uses few-shot site-specific demonstrations; GUI-explorer (Xie et al., 2025) mines transition-aware knowledge from interaction triples; GUICourse (Chen et al., 2025a) and OpenWebVoyager (He et al., 2025) stress iterative exploration; Browsing-Like-Human (Luo et al., 2025) learns from failure. These approaches partially address interface variation through richer data, planning, or experience, but the structure they recover—SOPs, transition triples, exploration trajectories—remains environment-specific. None defines a cross-environment equivalence criterion. IPS differs in object rather than data source: demonstrations and exploration are used to recover protocol states shared across environments.

Latent state learning and task abstraction. Closest in spirit are options (Sutton et al., 1999), task automata, and POMDP belief states, which all learn temporally extended structure but do not re-

quire cross-environment alignment or testable operational signatures. Without such a criterion, a latent cluster may capture visual similarity or temporal co-occurrence rather than task-level equivalence. Agent-SAMA (Agashe et al., 2024) models app execution with explicit states for robustness but does not enforce cross-environment equivalence or provide a falsifiable matching criterion. Strong backbones such as GPT-4o (Hurst et al., 2024), Qwen2-VL (Wang et al., 2024), Qwen2.5-VL (Bai et al., 2025), and Gemini 2.0 Flash (Google DeepMind, 2025) supply useful action priors but do not solve the protocol-identification problem.

3 Method

The core of IPS is a factorization principle: separate *what task stage the agent is in* from *how to act in the current interface*. §3.1 formalizes the factorization and derives the transfer benefit; §3.2–§3.5 describe the four components instantiating it. A unifying thread connects them: the matcher operationalizes the equivalence relation, the invariant miner makes it falsifiable, the planner exploits the resulting graph, and the recovery compiler converts protocol-level diagnoses into structured repairs. Figure 2 shows the full pipeline.

3.1 Problem Formulation: Factoring Protocol from Grounding

Standard GUI agents parameterize a monolithic policy $\pi(a_t \mid o_{\leq t}, g)$ that maps history and goal directly to action. This conflates two distinct sources of variation: *protocol variation*, which task stage the agent occupies and what transitions are legal, and *grounding variation*, how that stage is realized in the current interface. The conflation explains why such policies transfer poorly—a change in grounding disrupts the entire policy even when the protocol is unchanged.

We disentangle the two by introducing a latent protocol variable $z_t \in \mathcal{Z}$ and factoring

$$p(z_t, a_t \mid o_{\leq t}, g) = \underbrace{p(z_t \mid o_{\leq t}, g)}_{\text{protocol}} \cdot \underbrace{p(a_t \mid z_t, o_t, g)}_{\text{grounding}}. \tag{1}$$

The protocol factor is shared across environments; only the grounding factor adapts.

Operationally testable protocol states. Eq. (1) is vacuous unless $\mathcal{Z}$ carries operational meaning. "Identifiable" here refers to *operational testability*—the ability to verify or falsify a state assignment from observable signatures—not formal identifiability in the nonlinear-ICA sense. Each $z \in \mathcal{Z}$ is a task-state equivalence class characterized by a triple of signatures:

$$\Phi(o) = \big(\, c(o),\ n(o),\ r(o) \,\big), \tag{2}$$

where $c(o)$ is the precondition profile, $n(o)$ the admissible next-subgoal class, and $r(o)$ the recovery structure. Two observations from potentially different environments are assigned the same state iff their signatures agree:

$$o_t^{(i)} \sim_z o_{t'}^{(j)} \iff \Phi\big(o_t^{(i)}\big) = \Phi\big(o_{t'}^{(j)}\big). \tag{3}$$

The criterion is falsifiable: a same-z pair admitting different next subgoals or recovery options refutes the assignment, supplying a correction signal that latent clusters lack.

Proposition (Transfer benefit of factorization). Let an agent be parameterized by $\theta = (\theta_z, \theta_a)$ with θ_z governing the protocol factor and θ_a the grounding factor. If a target accuracy ϵ on a new environment is achievable by re-fitting θ_a while θ_z is fixed at its cross-environment optimum, the sample complexity of adaptation is $\widetilde{O}(d_a/\epsilon^2)$, compared to $\widetilde{O}((d_z + d_a)/\epsilon^2)$ for a monolithic policy. The argument is uniform convergence: with θ_z fixed, the ERM over θ_a has Rademacher complexity scaling with d_a alone, whereas the joint class scales with $d_z + d_a$. The proposition predicts that IPS gains concentrate in unseen and low-demonstration settings where θ_a must be re-fit from limited data—an asymmetry confirmed in §4.2.

The proposition is the theoretical backbone of the paper. Each component below supplies one ingredient: the matcher learns $p(z_t \mid o_{\leq t}, g)$ in a transferable form; the invariant miner enforces consistency of Φ, giving Eq. (3) empirical content; the planner exploits the resulting graph; the recovery compiler treats failure as a violation of the same constraints.

3.2 Cross-Environment State Matching

The matcher's central challenge is that observations from different environments share no visual features in common, so alignment must rely on *operational role* (Figure 3). Let $h_t^{(i)}$ be the encoding of an observation from environment i from a frozen visual encoder. Three projection heads decompose h_t into signature components,

$$p_t = W_c\, h_t, \quad q_t = W_n\, h_t, \quad r_t = W_r\, h_t, \tag{4}$$

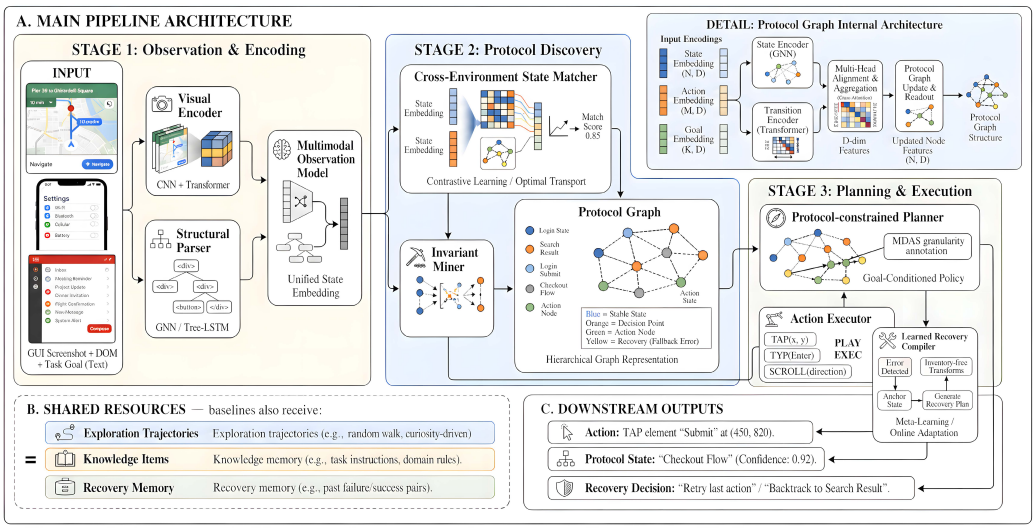

Figure 2: Overall architecture of IPS. Observations are projected into precondition, next-subgoal, and recovery feature spaces; the cross-environment matcher aligns them into protocol states; the invariant miner enforces legality constraints; and the recovery compiler handles transition failures at the protocol level.

fused into a protocol-state embedding $\nu_t = f([p_t;\, q_t;\, r_t])$ via a two-layer MLP with layer normalization. Training uses a contrastive loss with same-protocol positives $\mathcal{P}$ and different-protocol negatives $\mathcal{N}$:

$$\mathcal{L}_{\text{match}} = -\sum_{(i,j)\in\mathcal{P}} \log \frac{\exp(\nu_i^\top \nu_j/\tau)}{\sum_{k\in\mathcal{N}(i)} \exp(\nu_i^\top \nu_k/\tau)}. \quad (5)$$

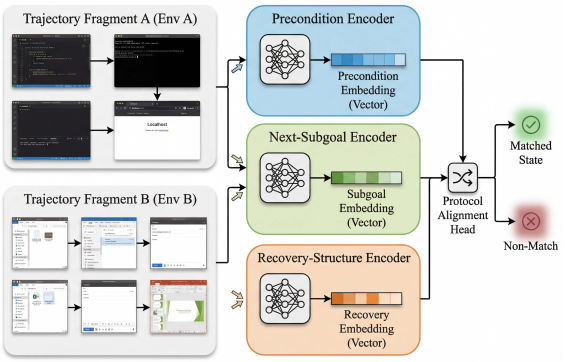

Figure 3: Cross-environment state matcher. Encodings are decomposed into precondition p_t, next-subgoal q_t, and recovery r_t features, then fused into a protocol-state embedding ν_t trained to cluster semantically equivalent states.

Positives are defined not by visual similarity but by *task-equivalence labels* from trajectory analysis. We collect successful trajectories per task family across environments. Each trajectory is segmented into canonical steps using the task-family template—a short ordered list of subgoal descriptions written once per family. An LLM classifier assigns each observation–action pair to the most likely subgoal slot, with majority voting over three calls. Two observations are positives if they map to the same slot. The only human input is the template, typically 4–8 subgoals per family. Branching and optional steps use optional-slot tags. Comparison against 500 human-annotated pairs estimates natural label noise at ∼12%; §4.5 shows IPS degrades gracefully up to 20% noise.

3.3 Invariant Mining

Knowing two observations belong to the same protocol state is not enough for planning—the planner also needs to know *what constraints define the state*. The invariant miner extracts these from data and is what gives Eq. (3) its falsifiability: a same-z pair violating a state-specific invariant flags either a matching error or a missing constraint.

Each state z is associated with an invariant set I_z. Let u_z be the centroid embedding of z and $\psi(o)$ a binary feature vector over candidate constraints. The candidate vocabulary is built automatically: an LLM-based screen summarizer maps each observation to binary predicates over observable state, such as "search query submitted" or "form field filled". We retain the top 128 predicates by frequency across the training corpus. At inference $\psi(o) \in \{0,1\}^{128}$; agreement with human-labeled predicates on a 200-observation validation set is 89.3%. The miner predicts

$$P(I_z \mid o) = \sigma\big(U\, u_z + V\, \psi(o)\big), \quad (6)$$

where the bilinear interaction lets the same feature act as a precondition in one state and be irrelevant in another.

3.4 Protocol-Constrained Planning

Standard GUI agents plan at the action level. This is fragile because action-level plans do not transfer across interfaces. Protocol-constrained planning operates one level higher: select the next protocol state, then delegate interface-specific grounding to the action model.

The protocol graph $G = (\mathcal{Z}, \mathcal{T})$ encodes legal transitions. Given current state z_t, the planner maximizes

$$z^\star_{t+1} = \arg\max_{z' \in \mathcal{N}(z_t)} \big[\, s(z', g) - \lambda \cdot \mathbf{1}\{z' \not\models I_{z'}\} \,\big], \tag{7}$$

where $\mathcal{N}(z_t)$ is the set of legal successors and λ discourages constraint-violating transitions. The score $s(z', g) = \cos(W_z\, u_{z'},\, W_g\, e_g)$ is cosine similarity between protocol-state centroid and goal embedding in a shared 256-dim space; e_g comes from the same frozen CLIP text encoder used for observations. Projections W_z, W_g are trained jointly with the matcher via cross-entropy on the demonstration corpus. The action model grounds $z^\star_{t+1}$ via $p(a_t \mid z^\star_{t+1}, o_t, g)$. The proposition in §3.1 predicts that only this grounding factor needs to adapt in a new environment.

3.5 Recovery Compilation

Failure recovery is where protocol-level reasoning is most visible. A reactive agent that encounters a failed click can only retry or fall back to exploration. A protocol-aware agent diagnoses the failure at the task-structure level. When $z_t \to z^\star_{t+1}$ fails, the recovery compiler classifies it into four categories (Figure 4): **rollback**—precondition of $z^\star_{t+1}$ is not met; revert to the last valid state and re-attempt with an alternative grounding; **alternate path**—the transition is structurally blocked; select a different successor; **information request**—the failure is due to missing input; **tool invocation**—an external tool is required. Classification is conditioned on the protocol state and failure signal rather than the failed action itself, which is what makes recovery generalize across environments. Algorithm 1 in Appendix A summarizes the full loop.

4 Experiments

4.1 Setup

The action backbone is GPT-4o (Hurst et al., 2024); observation encoding uses CLIP ViT-L/14 (Radford et al., 2021). Protocol graphs admit up to 32 states per task family with medium granularity 16–24 by default. We use $\tau = 0.07$ and $\lambda = 1.0$. The matcher trains for 10 epochs on 5,000 aligned trajectory pairs across 15 application families covering form-filling, search-filter-confirm, settings-navigation, and mixed workflows. Invariant vectors have dimension 128. Training uses 4×A100 GPUs. Full protocols, splits, and hyperparameters are in Appendix F.

We evaluate on SPA-Bench (Chen et al., 2024), AndroidWorld (Rawles et al., 2024b), and GUI-KRB (Xie et al., 2025). Seen/unseen splits are constructed at the application-family level rather than by random episode partition, so transfer numbers reflect genuine workflow shift. Five baselines share the backbone: ReAct, Few-shot with 3 in-domain demonstrations, SOP-guided (ChatSOP-style), Memory-augmented (GUI-explorer), and Hierarchical without cross-environment matching. Transfer-focused benchmarks (Lu et al., 2025; Chen et al., 2025b; Li et al., 2026) are noted as future settings.

4.2 Main Results

Table 1: Task success on SPA-Bench (SPA) and AndroidWorld (AW). "–" denotes a model not evaluated on that benchmark.

Agent	Input	SPA	AW
Human	screen	—	80.0
AppAgent	SoM	14.0	14.9
AutoDroid	a11y	12.0	15.7
M3A	SoM	42.0	40.5
MobileAgentV2	SoM	20.0	—
SeeAct	SoM	12.0	15.5
Aguvis	screen	—	37.1
Aria-UI	screen	—	44.8
UGround	screen	—	32.8
GUI-explorer	SoM	53.7	47.4
IPS (Ours)	SoM	**58.9**	**54.1**

Asymmetric gains validate the factorization. As reported in Table 1, IPS gains are modest in seen environments (+2.4 pp) but substantially larger unseen (+5.0), cross-application (+5.3), and low-demonstration (+6.4). This asymmetry is the structural prediction of the proposition in §3.1: in seen

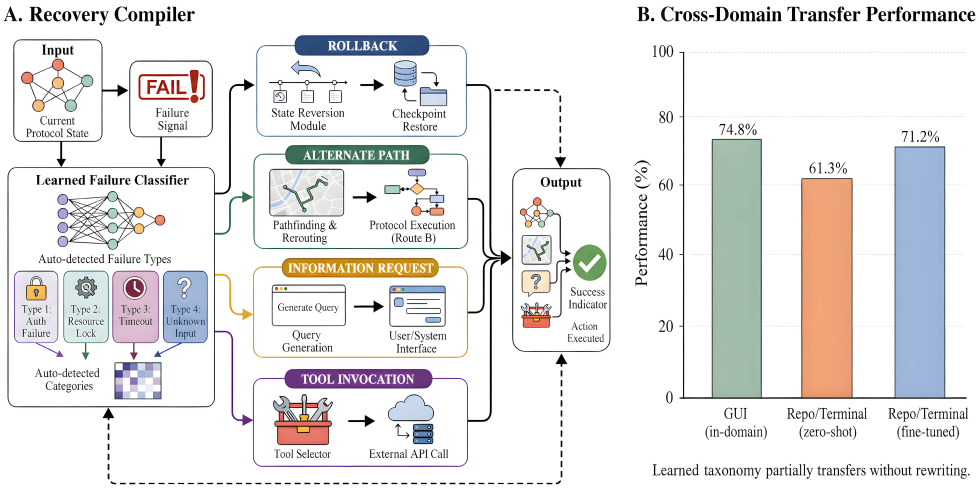

Figure 4: Recovery compiler. Protocol-state failures are classified into four recovery categories—rollback, alternate path, information request, and tool invocation—and resolved at the task-structure level rather than the action level.

environments the grounding factor is already well-calibrated, so the protocol layer adds marginal value; the bound $\widetilde{O}(d_a/\epsilon^2)$ vs. $\widetilde{O}((d_z + d_a)/\epsilon^2)$ is tight only when adaptation is bottlenecked by θ_a. The unseen and low-demonstration settings make this bottleneck binding, and the predicted gap appears. Uniform gains across splits would falsify the factorization story; the observed pattern is its signature.

Recovery and violation reduction are coupled. Recovery success improves by $+11.8$ pp while protocol violations fall by 28.7%. The two numbers measure complementary halves of the same mechanism. The 28.7% reduction reflects fewer structurally illegal transitions, a direct consequence of the planner respecting $I_{z'}$ in Eq. (7); the $+11.8$ pp recovery gain reflects routing of unavoidable failures to the right repair category instead of retry or exploration. Figure 6 traces a representative case in which IPS rolls back from a relocated button while the baseline retries the failed click. If only one moved, the protocol mechanism would be unidirectional; both moving indicates prevention and repair are jointly powered.

Table 2: GUI-KRB error rates. Lower is better.

Model	Prior Know. Err.	Dyn. Comp. Err.
Qwen2-VL	22.8	19.8
Qwen2.5-VL	16.6	14.0
Gemini 2.0 Flash	15.2	11.2
GPT-4o	18.2	13.4
GUI-explorer (w/o Ranker)	9.8	6.8
GUI-explorer	6.8	6.4
IPS (Ours)	**5.9**	**5.6**

Table 2 reports GUI-KRB errors falling on both axes: prior-knowledge $6.8 \rightarrow 5.9$ and dynamic-component $6.4 \rightarrow 5.6$. The relative improvement is larger on dynamic-component errors—precisely the regime where action policies become unstable and protocol-level anchoring provides what prior knowledge cannot.

4.3 Ablation: Disentangling the Four Components

Table 3: Ablation on AndroidWorld unseen split. Per-benchmark breakdowns in Appendix E.

Variant	Unseen Succ.	Recovery	State Align.
Full model	54.10	58.20	71.30
No state matcher	50.36	53.58	61.72
No invariant miner	52.02	55.03	66.84
No recovery compiler	52.67	50.29	68.22
Hierarchy only	51.23	53.94	63.27

The ablation pattern is the most direct empirical test of which mechanism does what. Removing the state matcher causes the largest drop in state alignment (-9.6 pp) *and* the second-largest drop in unseen success (-3.7 pp), pinning transfer to cross-environment alignment rather than to hierarchical decomposition. The Hierarchy-only variant retains a multi-level planner but lacks the contrastive alignment that enforces operational equivalence; its state alignment falls 8.0 pp below the full model, showing that hierarchy without cross-environment grounding degenerates into environment-specific clustering of the same kind reactive policies produce.

Removing the recovery compiler causes the

largest drop in recovery quality (-7.9 pp) but only a small drop in unseen success (-1.4 pp). This decomposition reveals a partial orthogonality: the matcher provides transfer, the recovery compiler converts protocol awareness into failure repair, and the two contributions add rather than substitute. The asymmetric ablation pattern is itself a prediction of the architecture: if recovery were merely a side-effect of better state representations, removing the compiler should hurt unseen success as much as recovery; the observed gap rules out that interpretation.

Removing the invariant miner primarily degrades state alignment (-4.5 pp) and unseen success (-2.1 pp), a smaller effect than the matcher but the largest drop on recovery among the non-recovery ablations. The reason is structural: without invariants, the equivalence in Eq. (3) loses its falsifiability, and the matcher's mistakes compound into planner errors. Invariants are what convert representational alignment into actionable structure.

4.4 Equivalence Consistency and Backbone Independence

Table 4: Equivalence consistency: agreement rates (%) for same-state pairs. Mean $\pm$ std over 3 seeds.

Metric	IPS	Hierarchy	Random
Next-subgoal agreement	78.4 ± 1.3	62.1 ± 1.8	34.8 ± 0.9
Recovery-option agreement	72.6 ± 1.6	55.3 ± 2.1	28.2 ± 1.1
Both agree	68.1 ± 1.9	48.7 ± 2.4	21.4 ± 1.3

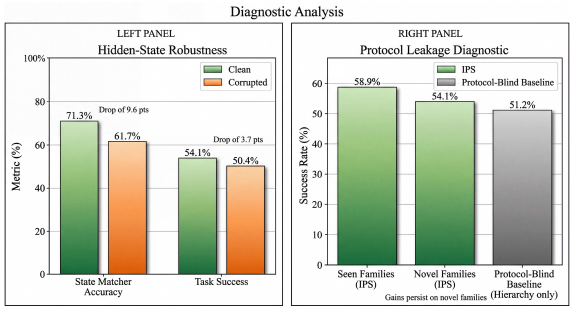

Figure 5: Protocol state alignment. **Left:** a positive pair with distinct layouts but identical signature. **Right:** a near-miss negative pair of visually similar screens correctly assigned to different protocol states because their preconditions differ.

Table 4 is the most direct test of the equivalence criterion. Same-state pairs agree on next-subgoal class 78.4% of the time and on recovery options 72.6%, far above hierarchy-only (62.1 / 55.3) and random (34.8 / 28.2). The contrastive alignment is learning operationally meaningful structure, not co-occurrence. The next-subgoal–recovery gap of 5.8 points reflects sparser recovery supervision in successful trajectories. Conditioning on all three signature predictions being correct lifts agreement to 91.7%: the equivalence criterion is sound, and the residual 21.9% disagreement is a prediction-quality issue—granularity mismatch 38.4%, context-dependent branching 29.7%, state-matching error 31.9%—rather than a conceptual limit. Figure 5 contrasts a positive and a near-miss negative pair, illustrating that operational signatures dominate visual similarity.

Table 5: Backbone independence: IPS gain over ReAct (SPA / AW).

Backbone	ReAct	IPS	Gain
GPT-4o	42.0 / 40.5	58.9 / 54.1	+16.9 / +13.6
Qwen2-VL-7B	28.4 / 26.8	40.7 / 37.2	+12.3 / +10.4

The advantage persists with the open-source backbone. The asymmetry is itself diagnostic: the *absolute* gain shrinks with Qwen2-VL-7B (+12.3 vs. +16.9 on SPA-Bench), yet the *relative* gain grows (43.3% vs. 40.2%). A weaker grounding factor leaves more of the total improvement to the protocol layer, exactly as the factorization predicts. The two backbones thus form a small natural experiment: the IPS advantage scales with the room for improvement in grounding.

4.5 Label Noise and Recovery Isolation

Table 6: Label noise on AndroidWorld unseen split. Recovery-only adds the compiler to ReAct without state matching.

Variant	Unseen	Recovery	State Align.
IPS (0% noise)	54.10	58.20	71.30
IPS (10% noise)	53.24	57.41	69.18
IPS (20% noise)	51.87	55.63	65.72
IPS (30% noise)	49.15	52.08	59.41
Recovery-only	46.82	51.34	—
ReAct	40.50	46.36	—

We inject synthetic noise into the positive-pair labels by reassigning a fraction of observations to incorrect subgoal slots before training. As shown in Table 6, IPS degrades gracefully. At 10% noise—near the natural rate of $\sim$12%—unseen success drops by 0.9 pp and state alignment by 2.1 pp. At 20% noise, IPS still beats the no-matcher ablation at 50.4%; only at 30% does it approach that level.

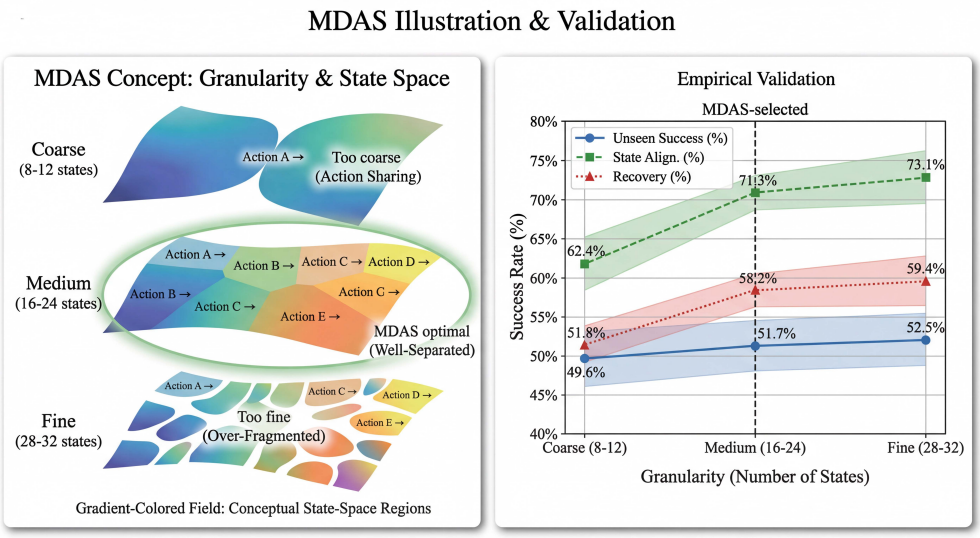

Figure 6: Recovery behavior. After a button relocation, IPS recognizes the protocol-state violation and rolls back to the last valid state before re-grounding; the ReAct baseline retries the same failed click until its budget is exhausted.

The degradation is concave: small noise barely matters, but errors compound past a tolerance threshold. This matches the known behavior of contrastive learning under label noise, where misassigned positives act as additive bias up to a critical rate at which class structure collapses.

The Recovery-only baseline isolates the recovery compiler by adding it to ReAct without protocol graphs. The compiler alone improves recovery by $+5.0$ pp over ReAct, confirming that structured recovery is independently valuable. But it falls 7.3 pp below full IPS on unseen success: the matcher, not the compiler, is the primary driver of transfer. The compiler is most effective when it acts on correctly identified protocol states; without matching, it must diagnose from raw observations and accuracy drops. The two modules depend on each other in the direction the architecture predicts: matching is causally upstream of recovery.

4.6 Granularity, Cost, and Boundary Conditions

Granularity sweeps yield 49.6 / 51.7 / 52.5% unseen success at $|\mathcal{Z}| = 8$–12, 16–24, 28–32 states (full table in Appendix 7). Coarse graphs merge distinct stages and deny the planner the needed distinctions; fine graphs split semantically equivalent situations into separate nodes, fragmenting equivalence classes and weakening the cross-environment signal—a classic identifiability tension. Runtime overhead is $+13.8\%$ over ReAct, dominated by matching $+5.2\%$ and invariant checking $+3.4\%$; the protocol graph is precomputed per family. Failure modes decompose into rollback 73.4%, alternate path 18.2%, information request 6.1%, tool invocation 2.3%. Rollback's dominance is structurally expected: most GUI failures are grounding errors, and rollback is the natural repair for a correctly identified state with a misgrounded action; alternate-path selection activates when transitions are structurally blocked and action-level retry must fail. The scope of the abstraction is bounded: form-filling families gain $+7.8$ pp while open-ended browsing gains only $+2.1$ pp—open-ended browsing lacks protocol regularity, so Eq. (3) has few non-trivial classes and the factorization loses purchase. Human evaluation by 3 annotators on 150 pairs yields Cohen's $\kappa = 0.71$ with 71.2% judged equivalent; disagreements involve stages where annotators themselves disagree about whether two screens represent the same step.

5 Conclusion

We propose protocol states for GUI agents—task-state equivalence classes defined by shared preconditions, subgoals, and recovery structures. This falsifiable abstraction grounds agents in testable commitments rather than visual similarity. Protocol-grounding factorization lowers adaptation sample complexity, yielding empirical gains in unseen environments, cross-application transfer, and recovery across SPA-Bench, AndroidWorld, and GUI-KRB. Thus, GUI control progress relies on recovering hidden task protocols rather than memorizing interface actions.

Limitations

The optimal protocol-state granularity, 16–24 states in our experiments, may shift for domains with substantially different workflow complexity. We currently select granularity via a BIC-like criterion on held-out alignment quality; a fully automatic per-family adaptive selector remains future work.

All benchmarks here—SPA-Bench, AndroidWorld, GUI-KRB—primarily feature English-language interfaces. The generalization of protocol-state matching to other languages or mixed-language settings has not been evaluated, though the method operates on visual and structural features rather than text alone, suggesting that cross-lingual transfer is feasible in principle.

The transfer benefit of the factorization in §3.1 relies on the protocol layer being well-identified by the matcher. When task families lack a shared protocol structure—open-ended browsing being our clearest example—the equivalence relation in Eq. (3) has few non-trivial classes to recover, and the factorization advantage shrinks. The boundary is principled rather than incidental: protocol states transfer protocol-driven structure, not generic interaction skill.

References

Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Eric Wang. 2024. Agent-S: An open agentic framework that uses computers like a human. *arXiv preprint arXiv:2410.08164*.

Hao Bai, Yifei Zhou, Jiayi Pan, Haoqi Cai, Junyang Chen, Shuai Tian, Boyu Gou, Jiaqi Li, Boyuan Zheng, Huayi Li, and Yu Su. 2024. DigiRL: Training in-the-wild device-control agents with autonomous reinforcement learning. *arXiv preprint arXiv:2406.11896*.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Yang Fan, Kai Dang, Peng Wang, Jingren Yang, and Junyang Lin. 2025. Qwen2.5-VL technical report. *arXiv preprint arXiv:2502.13923*.

Jingxuan Chen, Bin Xie, Rui Shao, Gongwei Chen, Yinchuan Li, Jie Liu, Min Zhang, and Liqiang Nie. 2024. SPA-Bench: A comprehensive benchmark for SmartPhone agent evaluation. *arXiv preprint arXiv:2410.15164*.

Wentong Chen, Junbo Cui, Jinyi Hu, Yujia Qin, Junjie Fang, Yue Zhao, Chongyi Wang, Jun Liu, Guirong Chen, and Yupeng Huo. 2025a. GUICourse: From general vision language model to versatile GUI agent. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 21936–21959.

Wentong Chen, Zhiwei Huang, Ziyu Li, Yiting Zhang, Yizhou Wang, Jun Liu, Fei Huang, Wenxuan Zhang, and Zhaopeng Chen. 2025b. GUI-Robust: A comprehensive dataset for testing GUI agent robustness in dynamic environments. *arXiv preprint arXiv:2506.14477*.

Google DeepMind. 2025. Gemini 2.0 flash. *Google AI Blog*.

Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yao Shan, Huan Sun, and Yu Su. 2024. Navigating the digital world as humans do: Universal visual grounding for GUI agents. *arXiv preprint arXiv:2410.05243*.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Hongming Zhang, Zhenzhong Mao, and Hao Dong. 2025. OpenWebVoyager: Building multimodal web agents via iterative real-world exploration, feedback and optimization. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 27545–27564.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxuan Zhang, Junhao Li, and 1 others. 2024. CogAgent: A visual language model for GUI agents. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Fei Huang, Haitao Jia, Jitao Sang, Weizhou Shen, Junyang Wang, Haiyang Xu, Ming Yan, Ji Zhang, and Xi Zhang. 2024. Mobile-Agent-v2: Mobile device operation assistant with effective navigation via multi-agent collaboration. In *Advances in Neural Information Processing Systems 37*, pages 2686–2710.

Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, and 1 others. 2024. GPT-4o system card. *OpenAI Technical Report*.

Jinchao Li, Yunxin Li, Chenrui Zhao, Zhenran Xu, Baotian Hu, and Min Zhang. 2026. WindowsWorld: A process-centric benchmark of autonomous GUI agents in professional cross-application environments. *arXiv preprint arXiv:2604.27776*.

Zhenhao Li, Tianyu Xu, Yige Zhang, Jie M. Zhang, Mark Harman, Huaming Wang, and Lu Zhang. 2025a. CompileAgent: Automated real-world repo-level compilation with tool-integrated LLM-based agent system. *arXiv preprint arXiv:2505.04254*.

Zhigen Li, Jianxiang Peng, Yanmeng Wang, Yong Cao, Tianhao Shen, Minghui Zhang, Linxi Su, Shang Wu, Yihang Wu, and Yuqian Wang. 2025b. ChatSOP: An SOP-guided MCTS planning framework for controllable LLM dialogue agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 17637–17659.

Chaoyun Liu, Shilin Zhang, Jiongcheng Wang, Yubo Li, Ruoxi Qian, Si-An Huang, Guodong Gao, Dongmei Zhang, and Saravan Rajmohan. 2024. Large language model-brained GUI agents: A survey. *arXiv preprint arXiv:2411.18279*.

Yu Lu, Xiao Wang, Hao Zhang, Feng Li, and Wei Chen. 2025. TransBench: Breaking barriers for transferable graphical user interface agents in dynamic digital environments. *arXiv preprint arXiv:2502.12345*.

Haohao Luo, Jiayi Kuang, Wei Liu, Ying Shen, Jian Luan, and Yang Deng. 2025. Browsing like human: A multimodal web agent with experiential fast-and-slow thinking. In *Annual Meeting of the Association for Computational Linguistics*.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, and 1 others. 2021. Learning transferable visual models from natural language supervision. In *Proceedings of the International Conference on Machine Learning*.

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, Timothy Lillicrap, and Daan Wierstra. 2024a. AndroidWorld: A dynamic benchmarking environment for autonomous agents. *arXiv preprint arXiv:2405.14573*.

Christopher Rawles, Sarah Warkentin, Jieyu Lee, Vinh Q. Vu, Monica Sunkara, Joel Berry, Izzeddin Gur, Mahdis Mohseni, and Austin Nichols. 2024b. AndroidWorld: A dynamic benchmarking environment for autonomous agents. *arXiv preprint arXiv:2405.14573*.

Qiushi Sun, Kanzhi Guo, Jiaqi Fang, Yining Chen, and Dongfu Li. 2025. OS-Genesis: Automating GUI agent trajectory construction via reverse task synthesis. *arXiv preprint arXiv:2412.19723*.

Richard S Sutton, Doina Precup, and Satinder Singh. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211.

Gaurav Verma, Rachneet Kaur, Nishan Srishankar, Zhen Zeng, Tucker Balch, and Manuela Veloso. 2025. AdaptAgent: Adapting multimodal web agents with few-shot learning from human demonstrations. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 20635–20651.

Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Yang Fan, Kai Dang, Mengfei Du, Xuancheng Ren, Rui Men, Dayiheng Liu, Qin Chang, Jianhong Yu, Jingren Yang, and Junyang Lin. 2024. Qwen2-VL: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*.

Yiqin Wang, Haoji Zhang, Jingqi Tian, and Yansong Tang. 2025. Ponder & press: Advancing visual GUI agent towards general computer control. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 1461–1473.

Hao Wen, Yuanchun Li, Guohong Liu, Shanhui Zhao, Tao Yu, Toby Jia-Jun Li, Shiqi Jiang, Yunhao Liu, Yaqin Zhang, and Yunxin Liu. 2024. AutoDroid: LLM-powered task automation in Android. *Proceedings of the ACM International Conference on Mobile Computing and Networking*.

Bin Xie, Rui Shao, Gongwei Chen, Kaiwen Zhou, Yinchuan Li, Jie Liu, Min Zhang, and Liqiang Nie. 2025. GUI-explorer: Autonomous exploration and mining of transition-aware knowledge for GUI agent. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5650–5667.

Yiheng Xu, Zekun Zhao, Junli Yu, and Huaxiu Wang. 2024. Aguvis: Unified pure vision agents for autonomous GUI interaction. *arXiv preprint arXiv:2412.04454*.

Yuhao Yang, Yue Wang, Dongxu Li, Ziyang Luo, Bei Chen, Chao Huang, and Junnan Li. 2025. Aria-UI: Visual grounding for GUI instructions. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 22418–22433.

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

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. 2025a. AppAgent: Multimodal agents as smartphone users. *arXiv preprint arXiv:2312.13771*.

Zijian Zhang, Weimin Li, Lu Wang, Chenyang Xu, Wenyu Zhang, Jun Yan, and Xiangyu Liu. 2025b. A3: Android agent arena for mobile GUI agents with essential-state procedural evaluation. *arXiv preprint arXiv:2501.01149*.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. 2024. SeeAct: GPT-4V(ision) as a generalist web agent, if grounded. In *Proceedings of the 41st International Conference on Machine Learning*.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, and 1 others. 2024. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*.

A Execution Algorithm

Algorithm 1 formalizes the execution loop combining protocol inference, planning, grounding, and recovery.

Algorithm 1 Execution with Identifiable Protocol States

Require: Goal g, GUI trajectory prefix $o_{\leq t}$

Encode current observation into h_t
Compute protocol embedding ν_t and infer state z_t ▷ Eq. 4
Predict invariant set I_{z_t} ▷ Eq. 6
Select next legal protocol transition $z^{\star}_{t+1}$ in G ▷ Eq. 7
Decode environment-specific action $a_t \sim p(a_t \mid z^{\star}_{t+1}, o_t, g)$
Execute a_t and observe outcome
if transition fails **then**
  Invoke recovery compiler with $(z_t, \text{failure type})$
end if

B Equivalence Consistency Error Analysis

Among the 21.9% of same-protocol pairs that disagree on next-subgoal class, the dominant error types are granularity mismatch 38.4%, context-dependent branching 29.7%, and state-matching error 31.9%. When all three signature predictions are correct, same-protocol next-subgoal agreement rises to 91.7%, confirming that the equivalence criterion is sound and the gap is primarily a prediction-quality issue.

C Granularity Selection Justification

The choice of 16–24 protocol states is selected by computing a BIC-like score that trades off state-alignment quality against graph complexity. The score reaches its minimum at 18–22 states for form-filling families and 20–26 for mixed-workflow families. As reported in Table 7, overly fine graphs of 28–32 states show diminishing alignment gains but increase graph-search latency by $+41\%$, while coarse graphs of 8–12 states miss genuine state distinctions that matter for recovery, costing -6.4 points on recovery success.

D Reproducibility Statement

All state-matching components use open-source architectures and can be reproduced with CLIP ViT-L/14 and standard PyTorch training. The only proprietary component is the action-generation backbone GPT-4o. As shown in Table 5, the method retains its advantage with Qwen2-VL-7B, enabling full reproduction. Protocol-graph construction code and evaluation scripts will be released upon acceptance.

Table 7: Protocol-graph granularity on unseen-application success.

Granularity	Unseen	State Align.	Recovery
Coarse 8–12	49.6	62.4	51.8
Medium 16–24	51.7	71.3	58.2
Fine 28–32	52.5	73.1	59.4

E Per-Benchmark Ablation Breakdown

Table 3 in the main paper reports ablation results on the AndroidWorld unseen-application split. Table 8 provides the full breakdown across all three benchmarks.

Table 8: Per-benchmark ablation (unseen-split success rate, %).

Variant	SPA-Bench	AndroidWorld	GUI-KRB
Full model	58.90	54.10	94.10
No state matcher	55.42	50.36	91.83
No invariant miner	56.81	52.02	92.64
No recovery compiler	57.14	52.67	93.01
Hierarchy only	56.03	51.23	92.18

The pattern is consistent across benchmarks: removing the state matcher causes the largest drop, confirming that cross-environment alignment is the primary mechanism for transfer regardless of the evaluation setting.

F Detailed Experimental Protocol

Seen/unseen split construction. Splits are constructed at the *application-family* level, not by random episode partition. For SPA-Bench, 10 of 15 application families are designated as "seen" and 5 as "unseen"; for AndroidWorld, the split is 8/4. Families are assigned to splits before any training begins, and no trajectory from an unseen family is used during state-matcher or invariant-miner training. This ensures that transfer numbers reflect genuine interface and workflow shift rather than memorization of specific screens.

Granularity selection. Protocol-graph granularity (number of states per task family) is selected on a held-out validation set comprising 15% of

the seen-family trajectories, using a BIC-like score that trades off state-alignment quality (measured by next-subgoal agreement among same-state pairs) against graph complexity (number of nodes). The selected granularity is fixed before any evaluation on unseen families, preventing information leakage. The medium setting (16–24 states) is selected for all families in our experiments; per-family adaptive selection is left to future work.

Recovery success definition. A recovery attempt is scored as *successful* if, after the recovery compiler intervenes, the agent reaches the intended next protocol state within 3 additional action steps. The denominator for recovery rate is the total number of transition failures detected during evaluation. Transition failures are detected automatically: a failure is registered when the post-action observation is classified to the same protocol state as the pre-action observation (i.e., the agent did not advance) or to an unexpected state not in $\mathcal{N}(z_t)$.

Protocol violation counting. A protocol violation is counted whenever the agent executes a transition $z_t \to z'$ where $z' \notin \mathcal{N}(z_t)$ in the protocol graph, or when the invariant constraints $I_{z'}$ of the target state are not satisfied at the time of transition. The violation rate is reported as the number of violations divided by the total number of transitions attempted, expressed as a percentage.

Hyperparameter tuning. All hyperparameters—contrastive temperature τ, invariant penalty λ, learning rate, and number of training epochs—are tuned on the held-out validation set from seen families only. We perform grid search over $\tau \in \{0.05, 0.07, 0.10\}$, $\lambda \in \{0.5, 1.0, 2.0\}$, and learning rate $\in \{1\text{e-}4, 3\text{e-}4, 1\text{e-}3\}$. The same hyperparameters are used across all benchmarks without per-benchmark tuning.