SUFFICE: INFERRING SCOPES AND INTERFACES FOR REUSABLE AGENT EXECUTION WITH SUFFICIENT INFORMATION FLOW

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-horizon agents often repeat similar tool calls and recovery steps across tasks, motivating the abstraction of repeated agent execution into reusable callable units (e.g., skills and task workflows) to reduce context and decision costs. Such abstraction therefore appears to require a tradeoff between omitting intermediate information that may alter subsequent decisions and retaining excessive history that erodes its benefits. We present **Suffice**, which solves this problem by deriving precise conditions on the information that must cross an abstraction boundary to preserve agent decisions. We further prove that passive trajectories generally cannot uniquely identify this information. Suffice uses controlled replay queries to hold candidate information fixed while varying other information and observing whether the agent's decision changes. Based on these queries, it identifies the minimal information required to preserve each decision and uses it to construct callable units. Across 200 synthetic programs, Suffice achieves an F1 score of 0.974 in recovering decision dependencies and exactly recovers 96.0% of abstraction boundaries. On AppWorld, Suffice improves Scenario Goal Completion (SGC) over the existing method ICAL by up to 23.5%, while achieving comparable performance to retaining all observed dataflow with 48.7% fewer interface tokens. Overall, agent execution abstraction appears to require a tradeoff between efficiency and task performance. Suffice shows that it need not.

1 INTRODUCTION

Long-horizon agents typically iterate between reasoning, tool calls, and observations, with each step determining the next action based on the preceding history (Yao et al. (2023); Schick et al. (2023); Qin et al. (2024)). When similar tasks recur, planning from scratch repeatedly produces similar low-level steps. This consumes reasoning budgets, expands the context, and leaves previously established local procedures exposed to online planning errors. To address these problems, Voyager writes successful code into a skill library (Wang et al. (2023)), Agent Workflow Memory induces reusable workflows from experience (Wang et al. (2024)), and ASI abstracts multi-step operations into verifiable programs (Wang et al. (2025)). Despite their different representations, these approaches can be viewed as abstracting agent behavior into callable units, including skills, task workflows, and delegated subagents.[1]

Recent methods further recover shared control flow from multiple successful trajectories (Guo et al. (2026); Sarch et al. (2024)) and compile executable workflows based on the producer-consumer relations of tool results (Yadouni & Li (2026)). These approaches are related to program slicing, which uses control flow and dataflow to identify relevant code (Horwitz et al. (1990); Tip (1995)), and provide a heuristic basis for proposing a scope and input/output relations of a callable unit. Once a candidate scope is encapsulated, the outer agent no longer selects its internal actions step by step and instead interacts with the encapsulated execution through the input and output interfaces. Preserving these actions therefore requires the interfaces to retain the parameters that carry the information needed for the relevant decisions.

[1] We use abstraction as an umbrella term for instances in which a segment of agent behavior is encapsulated behind a callable interface. Other uses of skills, workflows, or subagents are outside the scope of this work.

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.

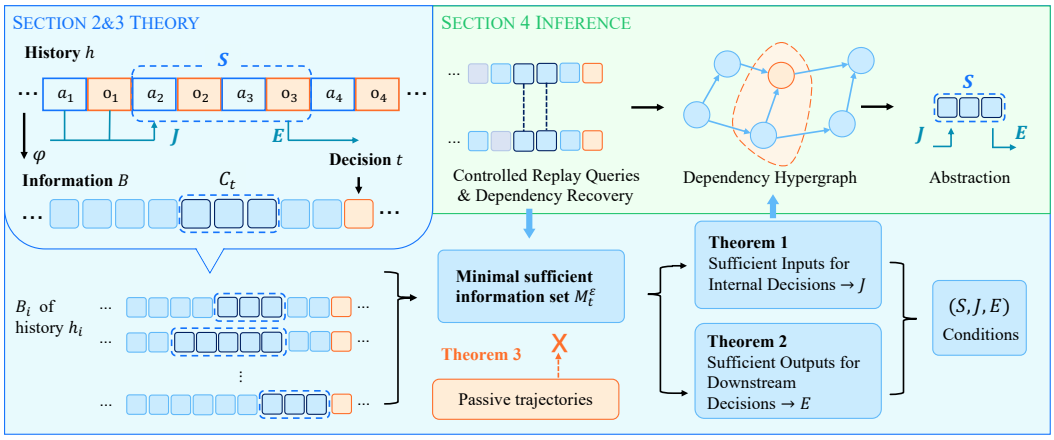


Figure 1: **Overview of Suffice.** An abstraction is characterized by its scope S and input/output interfaces J and E. Our theory derives the sufficient information conditions for preserving agent decisions and shows that the required information cannot generally be identified from passive trajectories alone. Suffice uses controlled replay queries to recover minimal sufficient information sets, constructs a dependency hypergraph, and infers the final abstraction (S, J, E).

Existing methods cannot guarantee that the constructed interfaces preserve all necessary parameters. With limited trajectories, parameters that distinguish different decisions may be insufficiently observed and therefore omitted from the interface. As a result, histories that require different actions may become indistinguishable through the interface. When the agent observes only this interface, it cannot reproduce these different decisions, resulting in performance degradation. Therefore, a method is needed to systematically derive interfaces that preserve all decision-relevant information.

Retaining the complete decision history in the interface avoids information loss, but carries the full context burden into the encapsulation, reducing the efficiency benefits of abstraction. Such long contexts increase inference costs and can degrade model performance. (Jiang et al. (2023); Li et al. (2023); Mu et al. (2023); Liu et al. (2024); Jiang et al. (2024); Pan et al. (2024); Xu et al. (2024)). Keeping all inputs based on the explicit dataflow observed during execution (Observed-Dataflow) suffers from the same problem. A value being used as input to a decision only indicates that it appeared in that execution, while removing it may leave the action unchanged. Efficient abstraction therefore requires a more precise information set that can reproduce the original agent's decisions without retaining excessive parameters. The problem then becomes how to determine a minimal sufficient information representation for the agent's action selection.

The key challenge is that a callable unit must simultaneously satisfy two competing requirements, neither omitting necessary information that affects decisions nor retaining redundant information irrelevant to those decisions. In this paper, we formalize the sufficient information conditions for reusable agent execution and derive the internal realizability and downstream substitutability theorems (Section 3). Based on these results, we construct a method for dependency recovery and boundary inference to precisely determine the scope and interfaces of callable units. Specifically, we identify the information necessary to preserve the agent's decisions by examining whether its removal changes those decisions across different histories. These dependencies then determine the abstraction boundary and the information that must cross it, as illustrated in Figure 1.

Suffice uses controlled replay queries (Section 4.1) to reliably identify minimal sufficient information sets, the smallest amount of information needed to preserve the original action distribution for each decision. Dependency recovery (Section 4.2) then searches for alternative sufficient sets and organizes these dependencies into a dependency hypergraph. Boundary and interface inference (Section 4.3) then derives candidate callable unit scopes and their interfaces from this dependency hypergraph. Information required within the scope but produced outside it forms the input interface, while information produced within the scope and required downstream forms the output interface. Given the deployment costs, Suffice jointly selects the scope of the callable unit and the parameters of its input and output interfaces.

We evaluate Suffice on a synthetic dataset and AppWorld (Trivedi et al. (2024)) (Section 5). The synthetic evaluation examines dependency and boundary recovery on 200 programs with complete

ground truth. Suffice achieves an overall Hyperedge F1 of 0.974 and a Boundary Exact score of 0.960, showing that it can accurately recover decision-relevant information and the corresponding abstraction boundaries from limited execution evidence. On AppWorld, Suffice improves Scenario Goal Completion (SGC) over the existing method ICAL (Sarch et al. (2024)) by up to 23.5%, showing that more accurate information identification improves task performance after encapsulation. Compared with retaining all information in the observed dataflow, Suffice achieves comparable task performance with 48.7% fewer interface tokens.

Our contributions are as follows.

We formulate the existing empirical problem of callable unit scope/interface construction as identifying the information dependencies required to preserve agent decisions and deriving the corresponding abstraction boundaries. To solve this problem, we establish a theoretical foundation for decision-preserving abstraction, and then develop dependency recovery and boundary inference methods to identify decision-relevant information and determine the scope and interfaces of callable units. Specifically, the key technical contributions are:

1. We derive theoretical conditions for internal realizability and downstream substitutability, and formalize the resulting decision dependencies using minimal sufficient information sets and a dependency hypergraph. Together, these results establish a candidate abstraction retains sufficient information to preserve the behavior of the original agent.
2. We develop a dependency recovery method based on controlled replay queries to identify the information that affects agent decisions across reachable histories. The recovered dependencies are then used to infer callable unit boundaries and interfaces that retain the information required for decision preservation.

We evaluate Suffice on synthetic programs and AppWorld, demonstrating accurate recovery of decision dependencies and abstraction boundaries while preserving task performance with substantially smaller interfaces in real executions.

2 Problem Setup

We study how to encapsulate a sequence of steps in agent execution, where the agent's policy selects each action based on the current history, into a callable unit. Before encapsulation, downstream actions can access the entire visible history; after encapsulation, internal decisions can only use information provided through the input interface or generated internally, while the outer agent must rely on the returned values for subsequent planning. The problem formulation therefore needs to capture what information is available during the original execution and how that information is provided to, generated within, and returned from the callable unit.

2.1 Long-Horizon Agent Execution

A long-horizon agent iterates between decisions and environment feedback. At position t ($t = 1, \ldots, T$), the policy selects an action A_t based on current history; after execution by a tool or the environment, an observation O_t is returned, introducing new information into downstream decisions.

Tool observations may contain implementation details that do not enter the agent context and are therefore unavailable to subsequent decisions. We use information representation functions ϕ to capture the information variables that do enter the context, where ϕ_t maps the action and observation at step t to the information visible to subsequent decisions:

$$B_t = \phi_t(A_t, O_t) = \{b_{t_1}, \ldots, b_{t_{mt}}\}.$$

Here, b_{t_k} may represent a file path, a search query, an error code, or a retrieved document. Information variables available before the agent takes its first action are represented by B_0, including user requirements, system constraints, and initial environment states. At position t, the history H_t^ϕ consists of B_0 and all preceding B_i available to the current decision. Given H_t^ϕ, the original policy induces the conditional action distribution $\pi_t(\cdot \mid H_t^\phi)$. Tool states and environment semantics constrain which histories are actually reachable; we denote their set by $\mathcal{H}_t^\phi$. Each variable is also

associated with its generation step, allowing decision dependencies to later be mapped back to the agent execution process.

2.2 Abstraction Model

We refer to the process of encapsulating a segment of agent behavior into a single callable unit as abstraction. The abstraction scope S specifies the agent actions inside the abstraction boundary, while J and E denote the input and output interfaces across this boundary. A candidate abstraction is therefore represented as a triplet (S, J, E). Once invoked, the outer agent provides arguments through J, the internal policy and tools continue to execute within S, and results required for downstream planning are returned through E.

The role of the same information depends on the choice of S. Information required by an internal decision must enter through J if it is generated outside S, whereas the same dependency can be satisfied internally if the information is generated within S. Similarly, information produced within S must enter E if it affects decisions after the call. Therefore, changing S also changes the input and output interfaces, and all three must be determined jointly.

3 Decision-Sufficient Agent Abstraction Extraction Theory

This section first considers what kind of (S, J, E) can preserve the behavior of the original agent, assuming that the information each decision requires to preserve the original policy is already known. Section 3.1 formalizes these information requirements as minimal sufficient information sets $\mathcal{M}_t^\epsilon$ and derives the constraints imposed by internal decisions on S and J. Section 3.2 derives the constraints imposed by downstream decisions on S and E. Section 3.3 then examines whether these information sets can be recovered from passive trajectories.

The following analysis assumes that the execution semantics of the original agent are fixed. The complete underlying assumptions are provided in Appendix A.1.

3.1 Internal Realizability

Encapsulation restricts the information available to internal decisions. If decision-relevant information is omitted from the input arguments, histories requiring different actions may become indistinguishable, preventing the internal policy from reproducing the original behavior.

For a decision t within S, let $C \subseteq \bigcup_{i<t} B_i$ be a candidate information set and $h[C]$ its values in history $h \in \mathcal{H}_t^\phi$. We measure how well the original policy can be reproduced using only C by

$$\rho_t(C) = \inf_{\kappa_C} \sup_{h \in \mathcal{H}_t^\phi} D_{\mathrm{TV}}\left(\pi_t(\cdot \mid h), \kappa_C(\cdot \mid h[C])\right).$$

Here, κ_C ranges over all conditional policies that depend only on C, and D_{TV} denotes the total variation distance between two action distributions. A smaller $\rho_t(C)$ means that C more closely reproduces the original agent's action distribution. A set C is called a minimal ϵ-sufficient information set if $\rho_t(C) \le \epsilon$ and no proper subset of C satisfies this condition. Here, sufficiency means that C alone can reproduce the original agent within the allowed error, while minimality means that removing any variable from C violates the error constraint. The collection of all such minimal sets is denoted by

$$\mathcal{M}_t^\epsilon = \{C \mid \rho_t(C) \le \epsilon,\ \rho_t(C') > \epsilon, \forall C' \subseteq C,\ C' \neq C\}.$$

Different minimal sets in $\mathcal{M}_t^\epsilon$ can substitute for one another. For example, if one minimal set is $\{y, z\}$, then y and z must be provided jointly; if $\mathcal{M}_t^\epsilon$ also contains $\{x\}$, then either $\{x\}$ or $\{y, z\}$ is sufficient to reproduce the same decision.

Theorem 1 (Internal Realizability[2]). *Given S and J, there exists an internal policy that achieves decision error at most ϵ_t for every $t \in S$ if and only if*

$$\forall t \in S, \exists C_t \in \mathcal{M}_t^{\epsilon_t} \text{ such that } C_t \subseteq J \cup \bigcup_{\substack{i<t \\ i \in S}} B_i.$$

[2]The corresponding error accumulation bound is provided in Appendix A.2.

Proof. For sufficiency, the stated inclusion condition ensures that each $C_t \in \mathcal{M}_t^{\epsilon_t}$ is available before decision t. By the definition of $\rho_t(C_t)$, a conditional policy using C_t can reproduce the original decision with error at most ϵ_t, yielding the required internal policy. For necessity, any internal policy achieving error at most ϵ_t implies that the information available before decision t is ϵ_t-sufficient. Removing redundant variables yields some $C_t \in \mathcal{M}_t^{\epsilon_t}$ satisfying the stated inclusion condition. □

Once C_t is fixed for each decision, variables in C_t that cannot be produced by S before the decision must be provided through the input arguments. The minimal input interface is

$$J(S; \{C_t\}) = \bigcup_{t \in S} \left(C_t \setminus \bigcup_{\substack{i<t \\ i \in S}} B_i \right).$$

3.2 Downstream Substitutability

We next consider the information required by downstream decisions. Let $\mathrm{Post}(S)$ denote the subsequent decisions outside S that are handled by the outer agent, and select $C_u \in \mathcal{M}_u^{\epsilon_u}$ for each $u \in \mathrm{Post}(S)$. The variables in these sets that are produced within S form the corresponding minimal output interface:

$$E(S; \{C_u\}) = \left(\bigcup_{t \in S} B_t \right) \cap \left(\bigcup_{u \in \mathrm{Post}(S)} C_u \right).$$

Theorem 2 (Downstream Substitutability)**.** *Suppose that J contains $J(S; \{C_t\})$, E contains $E(S; \{C_u\})$, and the encapsulated and original executions produce the same tool results, external side effects, and environment state transitions whenever their internal actions are the same. Let $P_{\mathrm{orig}}^{\mathrm{out}}$ and $P_{\mathrm{abs}}^{\mathrm{out}}$ denote the downstream execution distributions under the original and encapsulated executions, respectively. Then*

$$D_{\mathrm{TV}}\left(P_{\mathrm{orig}}^{\mathrm{out}}, P_{\mathrm{abs}}^{\mathrm{out}}\right) \le \sum_{t \in S} \rho_t(C_t) + \sum_{u \in \mathrm{Post}(S)} \rho_u(C_u).$$

If all selected sets are exactly sufficient, these distributions are identical.

Proof. By Theorem 1, each internal decision can reproduce the original action distribution with error $\rho_t(C_t)$. When the internal actions agree, the assumed tool semantics and environment transitions preserve the values returned through E and the external state. Therefore, each downstream decision u receives the required information in C_u and can reproduce the original action distribution with error $\rho_u(C_u)$. A union bound over the internal and downstream decision errors gives the stated result; when all errors are zero, the two distributions are identical. □

Together, Theorems 1 and 2 characterize when a candidate abstraction (S, J, E) preserves the behavior of the original agent. These conditions, however, assume that the sufficient information sets for the relevant decisions are known.

3.3 Identifiability from Agent Trajectories

The derivations above assume that $\mathcal{M}_t^{\epsilon}$ is known at each decision. Agent trajectories generated by natural tasks, however, cover only the histories that actually occur. Variables jointly determined by the task structure or preceding actions may therefore remain correlated, making it difficult to directly identify $\mathcal{M}_t^{\epsilon}$ from observed trajectories.

Suppose that binary variables X and Y can vary independently in the environment, but the passive trajectories cover only $X = Y$. Consider two policies, $\pi^{(X)}$ and $\pi^{(Y)}$, whose actions depend only on X and Y, respectively. Let q_0 and q_1 be two distinct action distributions, and define

$$\pi^{(X)}(\cdot \mid X, Y) = q_X, \qquad \pi^{(Y)}(\cdot \mid X, Y) = q_Y,$$

where $q_X = q_0$ when $X = 0$ and $q_X = q_1$ when $X = 1$, and analogously for q_Y.

Algorithm 1 Suffice: Dependency Recovery and Abstraction Inference

Dependency Recovery

Input: Trajectories D, decisions T, policy π, budgets Q, thresholds ϵ

$\mathcal{G} \leftarrow \emptyset$
for $t \in$ RELEVANTDECISIONS(D, T) **do**
  for each C visited by deletion search **do**
    $z_t(C) \leftarrow$ SUFFICIENCYTEST(C, t) $\triangleright\ \pi, \mathcal{H}_t^{\phi}, Q_t, \epsilon_t$
  end for
  $\widehat{\mathcal{M}}_t^{\epsilon} \leftarrow$ minimal sufficient sets certified by z_t
  $\mathcal{G} \leftarrow \mathcal{G} \cup \{C \to t \mid C \in \widehat{\mathcal{M}}_t^{\epsilon}\}$
end for

Output: Dependency hypergraph $\mathcal{G}$

Abstraction Inference

Input: Dependency hypergraph $\mathcal{G}$, decisions T, deployment cost L

$\mathcal{F} \leftarrow \emptyset$
for each dependency-consistent $(S, \{C_r\})$ **do**
  $J \leftarrow J(S; \{C_t\})$
  $E \leftarrow E(S; \{C_u\})$
  if (S, J, E) satisfies Thms. 1–2 **then**
    $\mathcal{F} \leftarrow \mathcal{F} \cup \{(S, J, E)\}$
  end if
end for
$(S^*, J^*, E^*) \leftarrow$ PARETOSELECT$(\mathcal{F}, L)$

Output: Selected abstraction (S^*, J^*, E^*)

Theorem 3 (Passive Non-identifiability)**.** *In general, minimal sufficient information sets cannot be uniquely recovered from the passive trajectory distribution alone.*

Proof. On the passive trajectories, $X = Y$, so $\pi^{(X)}$ and $\pi^{(Y)}$ induce the same action distributions and are indistinguishable from the passive trajectory distribution. However, on reachable histories where $X \neq Y$, the two policies make different decisions, with $\pi^{(X)}$ depending only on X and $\pi^{(Y)}$ only on Y. Their minimal sufficient information sets are therefore $\{X\}$ and $\{Y\}$, respectively. Hence, the same passive trajectory distribution is consistent with different minimal sufficient information sets, which cannot in general be uniquely recovered from that distribution alone. □

Theorems 1 and 2 assume that $\mathcal{M}_t^{\epsilon}$ is known at each decision, while Theorem 3 shows that these sets generally cannot be recovered from passive trajectories alone. Constructing a triplet (S, J, E) that satisfies the preceding conditions therefore requires information beyond passive observations. To obtain this information, we use controlled replay queries to identify minimal sufficient information sets and map their variables back to the steps where they are generated. The theoretical basis for replay queries is provided in Appendix A.3, and Section 4 presents the concrete inference procedure.

4 SUFFICE: ABSTRACTION INFERENCE

Section 3 establishes the conditions for preserving the behavior of the original agent, but the minimal sufficient information sets required by these conditions are unknown in practice.

To this end, Suffice uses controlled replay queries to determine whether an information set is sufficient to reproduce a decision (Section 4.1). It then searches for minimal sufficient information sets (Section 4.2) and uses the recovered information and its generation locations to infer the abstraction scope and interfaces (Section 4.3). Algorithm 1 provides an overview of the inference procedure.

4.1 CONTROLLED REPLAY QUERY

For decision t, Suffice branches from the original execution before action A_t to construct paired histories that agree on C but differ in other reachable information, and replays the frozen policy so that any observed action discrepancy can only result from changes in the other reachable information.

If the agent's action distribution changes, the history pair constitutes a counterexample showing that the current C is insufficient. If the action remains stable across the tested variants and the statistical evidence meets the certification condition, C is certified as sufficient. When the statistical evidence is insufficient, the method returns `unknown`. State restoration, reachable history construction, and action distribution estimation are described in Appendix B.1.

4.2 DEPENDENCY RECOVERY

We refer to each minimal sufficient information set required to reproduce a decision as a dependency. Suffice uses the above procedure to progressively recover these dependencies and organize them into a dependency hypergraph. Here, an information dependency is a minimal sufficient set of information required to reproduce the corresponding decision. Specifically, Suffice starts from all

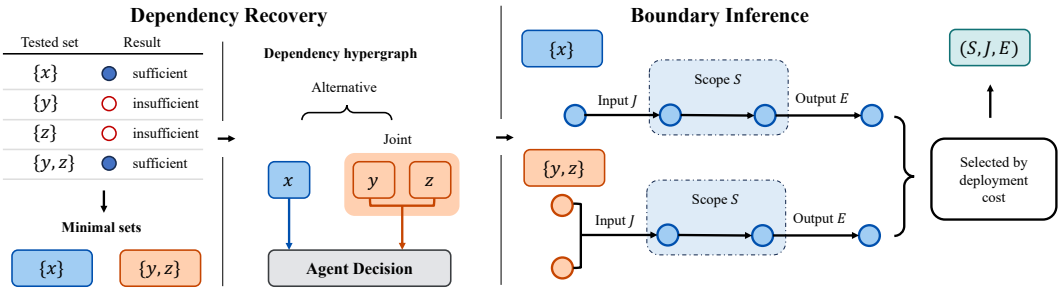


Figure 2: **Dependency recovery and boundary inference in Suffice.** Dependency recovery identifies minimal sufficient information sets and organizes them into a dependency hypergraph. Different minimal sets represent alternative ways to support the same decision, while variables in the same set are required together. Boundary inference derives candidate scopes and input/output interfaces from these dependencies and selects the final abstraction (S, J, E) according to deployment costs.

information visible before an action, removes one information variable at a time, and tests the remaining set with a controlled replay query. If the action remains unchanged, the removal is accepted and the set is further reduced; if a counterexample is found, the removed information is restored to the candidate set and the corresponding search branch terminates. Counterexample reuse and branch pruning are detailed in Appendix B.2.

After the search terminates, Suffice represents information variables as source nodes and decisions as target nodes, and encodes each certified minimal set as a hyperedge pointing to the corresponding decision. Information within the same hyperedge must appear jointly, while different hyperedges pointing to the same decision represent alternative sufficient information sets, forming the dependency hypergraph illustrated in Figure 2. Each information node also records its generation step and availability locations, allowing decision dependencies to be mapped back to the agent's execution process. Appendix A.4 provides a finite-sample recovery guarantee for this dependency search.

4.3 Boundary and Interface Inference

For a candidate scope S, boundary inference selects one incoming hyperedge for each internal decision and traces the information variables in each selected hyperedge back to their generation steps. If a generating step is already within S, the dependency is satisfied by an internal result; if it precedes S, the search can either retain its result in J or include the corresponding execution in S. Newly included decisions trigger further hyperedge selection until every internal decision can be supported by either input information or internal results.

Suffice also examines downstream decisions that remain with the outer agent and selects sufficient hyperedges for them, thereby identifying the information required by these decisions. The corresponding results produced within S enter E, while other internal results remain within the abstraction. The output interface is therefore determined by the dependencies required by the downstream decisions of the outer agent.

After identifying the feasible boundary candidates, we formulate their selection as a multi-objective optimization over deployment costs (e.g., interface tokens and scope size) and use Pareto-frontier search with dominance pruning. The search retains only boundaries that are non-dominated under the deployment objectives and selects the (S, J, E) to be used from the resulting frontier.

The resulting (S, J, E) therefore defines the callable unit, with S specifying the execution encapsulated, while J and E define its input and output interfaces. The recovered decision dependencies are thus translated into an explicit abstraction that preserves both internal and downstream decisions.

5 Experiments

5.1 Experimental Setup

Different methods produce different native outputs, so we map them to a common abstraction representation and apply the same evaluation protocol. We compare Suffice with ICAL (Sarch et al., 2024), which extracts reusable executions from successful trajectories; Observed-Dataflow, which retains all observed information flow across the same abstraction scope as Suffice; Pairwise, which

(a)

Subset	Method	Hyperedge F1	Input F1	Output F1	Boundary Exact
Overall	Suffice	**.974**	**.982**	**.975**	**.960**
	ICAL	.876	.913	.897	.875
	Pairwise	.152	.205	.168	.015
Hard	Suffice	**.945**	**.958**	**.946**	**.800**
	ICAL	.777	.842	.811	.675
	Pairwise	.011	.034	.018	.000

(b)

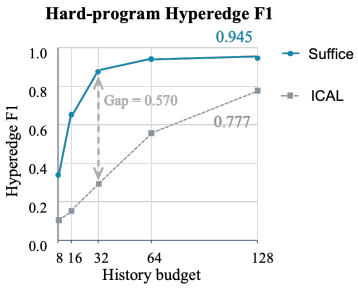


Figure 3: **Synthetic evaluation.** (a) Dependency and boundary recovery under the maximum history budget. (b) Hyperedge F1 under different history budgets.

constructs abstractions from individual information effects; and LLM-Semantic, which uses GPT-5.6 Sol (xhigh) to construct abstractions from trajectory and information semantics.

The synthetic evaluation includes 160 randomly generated programs and 40 hard programs with complex dependency structures, evaluated under history budgets from 8 to 128. We measure dependency and boundary recovery using Hyperedge F1, input/output F1, and boundary exact against exhaustively derived ground truth.[3] The AppWorld (Trivedi et al. (2024)) evaluation uses 25/10/30/30 scenarios from train/dev/test_normal/test_challenge, respectively, and measures task performance and deployment cost using Scenario Goal Completion (SGC) and interface tokens. All methods use the same original policy, tool environment, and execution protocol. Additional experimental and implementation details are provided in Appendix C.

5.2 Dependency and Boundary Recovery

Suffice relies on accurately recovering the minimal sufficient information sets for each decision and determining abstraction boundaries from these dependencies and their generation locations. Synthetic programs allow both stages to be evaluated under complete information, as their finite state spaces and policies are fully known. We exhaustively enumerate reachable histories to obtain the ground-truth minimal sufficient information sets for each decision, and use these ground-truth dependencies to enumerate all valid abstractions satisfying Theorems 1 and 2. We use Hyperedge F1 to measure dependency recovery, while input/output F1 and boundary exact measure whether the boundary inference procedure recovers these valid abstractions. Figure 3(a) reports the results under the maximum history budget. Suffice achieves an overall Hyperedge F1 of 0.974, input/output F1 scores of 0.982 and 0.975, and exactly recovers the complete abstraction boundary on 96.0% of the programs. On the hard programs, Hyperedge F1 is 0.945 and boundary exact is 80.0%.

Figure 3(b) further shows dependency recovery under history budgets from 8 to 128. Suffice improves from 0.340 Hyperedge F1 with 8 histories to 0.940 with 64 and 0.945 with 128, while ICAL improves from 0.105 to 0.780 over the same range. The gap reaches 0.570 at a budget of 32, showing that controlled replay queries recover useful dependency evidence with fewer observed histories.

ICAL recovers most dependencies when observed executions provide sufficient variation, but degrades on hard programs where correlated variables obscure the actual decision conditions. Suffice improves Hyperedge F1 by 0.168 on these programs, with the largest gains on alternative, proxy, and xor cases. Pairwise, which cannot represent joint information sets, drops to 0.011 Hyperedge F1, while Observed-Dataflow covers all necessary information variables (recall = 1.000) but retains many irrelevant ones (precision = 0.613). The construction of the synthetic programs and the breakdown of the hard dependency families are provided in Appendix C.1.

5.3 Task Performance

The synthetic experiments validate the accuracy of dependency and boundary recovery. We further evaluate whether the abstractions constructed by Suffice can retain the information required to

[3]The history budget is the number of histories available for dependency recovery. F1 uses the standard precision–recall definition. Hyperedges F1 match only by exact set equality; input and output F1 are computed separately over interface variables. Boundary exact requires the scope and both interfaces to match the same valid abstraction, with any valid abstraction accepted.

Table 1: Task performance and interface size on AppWorld.

Method	test_normal SGC ↑	test_challenge SGC ↑	Interface Tokens ↓
Original Agent (ReAct)	46.7	40.0	–
Suffice	76.7	**70.0**	257.2
ICAL	63.3	56.7	143.2
LLM-Semantic	53.3	36.7	**79.9**
Observed-Dataflow	**80.0**	66.7	501.5

reproduce the original agent's decisions and thereby preserve task performance in real executions. Table 1 reports SGC and interface tokens on test_normal and test_challenge.

Compared with ICAL, Suffice improves SGC by 21.1% and 23.5% on test_normal and test_challenge, respectively. ICAL constructs shorter interfaces from passive evidence in successful trajectories but may omit information required in new executions. LLM-Semantic produces even shorter interfaces, but its SGC drops further to 53.3/36.7. These results show that more completely retaining the information required to reproduce the original agent's decisions leads to better task performance after abstraction.

Suffice achieves comparable SGC to Observed-Dataflow on both splits while reducing interface tokens by 48.7%. This result shows that dependency recovery can substantially reduce interface redundancy while retaining the information needed to preserve task performance.

6 Related Work

Reusable Agent Architecture. Prior work converts successful trajectories into reusable memories, workflows, skills, or executable programs for subsequent tasks (Sutton et al. (1999); Bacon et al. (2017); Shinn et al. (2023); Wang et al. (2023); Cai et al. (2024); Sarch et al. (2024); Zhao et al. (2024); Wang et al. (2024); Zhang et al. (2025); Wang et al. (2025); Guo et al. (2026). Separately, program analysis techniques extract relevant components from observed execution structure (Weiser (1984); Ferrante et al. (1987); Horwitz et al. (1990); Tip (1995)). Suffice instead determines which steps can be hidden behind an abstraction boundary and which information must cross the boundary to preserve the original agent's internal execution and downstream planning.

Information Representation. Existing studies in feature selection and state abstraction represent the information available for prediction or control through explicit variables under predictive or behavioral criteria (Koller & Sahami (1996); Dean & Givan (1997); Ferns et al. (2004); Li et al. (2006); Abel et al. (2016); Chen et al. (2018); Gelada et al. (2019); Yoon et al. (2019); Zhang et al. (2021); Fu et al. (2021); Huang et al. (2022)). Suffice similarly represents the information available at each agent decision as explicit information units, providing a representation on which sufficiency can subsequently be tested.

Sufficiency Identification. Model interpretation methods characterize information relevant to preserving predictions by perturbing or restricting model inputs (Fong & Vedaldi (2017); Chen et al. (2018); Yoon et al. (2019); Vig et al. (2020)). Related methods further identify sufficient rules or minimal input subsets that preserve model predictions or decisions (Ribeiro et al. (2018); Carter et al. (2019)). Suffice adapts these techniques to agent decisions, identifying sufficient information sets under controlled variations of the available information.

7 Conclusion

We present Suffice for abstracting agent execution into reusable callable units while preserving agent decisions. Building on theoretically established preservation conditions, Suffice uses controlled replay queries to recover minimal sufficient information sets, then derives abstraction scopes and interfaces from the recovered dependencies. Across 200 synthetic programs, Suffice achieves a Hyperedge F1 of 0.974 and exactly recovers 96.0% of abstraction boundaries. On AppWorld, it improves SGC over ICAL by up to 23.5% and matches the task performance of retaining all observed dataflow with 48.7% fewer interface tokens. These results establish decision-relevant information as a principled basis for deriving agent abstraction scopes and interfaces.

AI USE STATEMENT

In this work, we used generative AI tools to assist with language polishing, as none of the authors are native English speakers. All research ideas, theoretical development, and mathematical derivations were developed by the authors. We have reviewed all AI-assisted work and take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide the formal definitions and main theoretical results in the main text, with additional theoretical analyses in the appendix. The Suffice inference procedure is described in Section 4 and Algorithm 1, with implementation details for controlled replay queries, dependency recovery, and boundary inference provided in the appendix. The appendix also documents the synthetic program construction, AppWorld evaluation setup, baselines, and implementation configurations used in our experiments. Source code will be released upon acceptance.

REFERENCES

David Abel, David Hershkowitz, and Michael L. Littman. Near optimal behavior via approximate state abstraction. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 2915–2923, 2016. URL https://proceedings.mlr.press/v48/abel16.html.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, pp. 1726–1734, 2017.

Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. Large language models as tool makers. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=qV83K9d5WB.

Brandon Carter, Jonas Mueller, Siddhartha Jain, and David Gifford. What made you do this? understanding black-box decisions with sufficient input subsets. In *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics*, volume 89 of *Proceedings of Machine Learning Research*, pp. 567–576, 2019. URL https://proceedings.mlr.press/v89/carter19a.html.

Jianbo Chen, Le Song, Martin J. Wainwright, and Michael I. Jordan. Learning to explain: An information-theoretic perspective on model interpretation. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 883–892, 2018. URL https://proceedings.mlr.press/v80/chen18j.html.

Thomas Dean and Robert Givan. Model minimization in markov decision processes. In *Proceedings of the 14th AAAI Conference on Artificial Intelligence*, pp. 106–111, 1997.

Norman Ferns, Prakash Panangaden, and Doina Precup. Metrics for finite markov decision processes. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, pp. 162–169, 2004.

Jeanne Ferrante, Karl J. Ottenstein, and Joe D. Warren. The program dependence graph and its use in optimization. *ACM Transactions on Programming Languages and Systems*, 9(3):319–349, 1987. doi: 10.1145/24039.24041.

Ruth C. Fong and Andrea Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3449–3457, 2017. doi: 10.1109/ICCV.2017.371.

Xiang Fu, Ge Yang, Pulkit Agrawal, and Tommi Jaakkola. Learning task informed abstractions. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 3480–3491, 2021. URL https://proceedings.mlr.press/v139/fu21b.html.

Carles Gelada, Saurabh Kumar, Jacob Buckman, Ofir Nachum, and Marc G. Bellemare. DeepMDP: Learning continuous latent space models for representation learning. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2170–2179, 2019. URL https://proceedings.mlr.press/v97/gelada19a.html.

Zhongxin Guo, Danrui Qi, Hanwen Gu, Peng Cheng, and Yongqiang Xiong. SKILL-DISCO: Distilling and compiling agent traces into reusable procedural skills. *arXiv preprint arXiv:2606.26669*, 2026. URL https://arxiv.org/abs/2606.26669.

Susan Horwitz, Thomas Reps, and David Binkley. Interprocedural slicing using dependence graphs. *ACM Transactions on Programming Languages and Systems*, 12(1):26–60, 1990. doi: 10.1145/77606.77608.

Biwei Huang, Chaochao Lu, Liu Leqi, José Miguel Hernández-Lobato, Clark Glymour, Bernhard Schölkopf, and Kun Zhang. Action-sufficient state representation learning for control with structural constraints. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 9260–9279, 2022. URL https://proceedings.mlr.press/v162/huang22k.html.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LLMLingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 13358–13376, 2023. doi: 10.18653/v1/2023.emnlp-main.825. URL https://aclanthology.org/2023.emnlp-main.825/.

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LongLLMLingua: Accelerating and enhancing LLMs in long context scenarios via prompt compression. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pp. 1658–1677, 2024. doi: 10.18653/v1/2024.acl-long.91. URL https://aclanthology.org/2024.acl-long.91/.

Daphne Koller and Mehran Sahami. Toward optimal feature selection. In *Proceedings of the 13th International Conference on Machine Learning*, pp. 284–292, 1996.

Lihong Li, Thomas J. Walsh, and Michael L. Littman. Towards a unified theory of state abstraction for MDPs. In *Proceedings of the International Symposium on Artificial Intelligence and Mathematics*, 2006.

Yucheng Li, Bo Dong, Frank Guerin, and Chenghua Lin. Compressing context to enhance inference efficiency of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 6342–6353, 2023. doi: 10.18653/v1/2023.emnlp-main.391. URL https://aclanthology.org/2023.emnlp-main.391/.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024. doi: 10.1162/tacl_a_00638. URL https://aclanthology.org/2024.tacl-1.9/.

Jesse Mu, Xiang Lisa Li, and Noah D. Goodman. Learning to compress prompts with gist tokens. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, H. Vicky Zhao, Lili Qiu, and Dongmei Zhang. LLMLingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 963–981, 2024. doi: 10.18653/v1/2024.findings-acl.57. URL https://aclanthology.org/2024.findings-acl.57/.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=dHng2O0Jjr.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

Gabriel Herbert Sarch, Lawrence Jang, Michael J. Tarr, William W. Cohen, Kenneth Marino, and Katerina Fragkiadaki. VLM agents generate their own memories: Distilling experience into embodied programs of thought. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/8ac50fd0a4eeeb1f077b17bb7c5353c3-Abstract-Conference.html.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

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. doi: 10.1016/S0004-3702(99)00052-1.

Frank Tip. A survey of program slicing techniques. *Journal of Programming Languages*, 3:121–189, 1995.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.850.

Jesse Vig, Sebastian Gehrmann, Yonatan Belinkov, Sharon Qian, Daniel Nevo, Yaron Singer, and Stuart Shieber. Investigating gender bias in language models using causal mediation analysis. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023. URL https://arxiv.org/abs/2305.16291.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024. URL https://arxiv.org/abs/2409.07429.

Zora Zhiruo Wang, Apurva Gandhi, Graham Neubig, and Daniel Fried. Inducing programmatic skills for agentic tasks. In *Proceedings of the 2nd Conference on Language Modeling*, 2025. URL https://arxiv.org/abs/2504.06821.

Mark Weiser. Program slicing. *IEEE Transactions on Software Engineering*, SE-10(4):352–357, 1984. doi: 10.1109/TSE.1984.5010248.

Yang Xu, Yunlong Feng, Honglin Mu, Yutai Hou, Yitong Li, Xinghao Wang, Wanjun Zhong, Zhongyang Li, Dandan Tu, Qingfu Zhu, Min Zhang, and Wanxiang Che. Concise and precise context compression for tool-using language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 16430–16441, 2024. doi: 10.18653/v1/2024.findings-acl.974. URL https://aclanthology.org/2024.findings-acl.974/.

Salma El Yadouni and Guanyi Li. TraceCompiler: Skill-guided mining and compilation of LLM agent traces into mostly deterministic workflows. *arXiv preprint arXiv:2608.02680*, 2026. URL https://arxiv.org/abs/2608.02680.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Jinsung Yoon, James Jordon, and Mihaela van der Schaar. INVASE: Instance-wise variable selection using neural networks. In *International Conference on Learning Representations*, 2019.

Amy Zhang, Rowan McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning invariant representations for reinforcement learning without reconstruction. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=-2FCwDKRREu.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=z5uVAKwmjf.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024. doi: 10.1609/aaai.v38i17.29936.

A Theory Details

This appendix presents the technical conditions, strengthened error bounds, replay criterion, and finite-sample recovery results omitted from Section 3.

A.1 Technical Conditions

Throughout the analysis, the decision alignment, information representation function ϕ, original policy, and reachable history set $\mathcal{H}_t^\phi$ are fixed. Candidate information consists of a finite set of information variables, and the action space is finite at each decision. κ_C may select different conditional action distributions for different values of $h[C]$.

At the start of a call, every variable in J must already be available. Operations within S are executed in causal order using the same tool semantics and environment transitions as the original agent. Persistent tool calls, permission changes, and transactional states must be reproduced by the wrapper or explicitly declared. Candidates for which these side effects cannot be preserved within a single call are not considered feasible triplets.

A.2 Accumulation of Decision Errors

Theorem 1 characterizes internal realizability at each decision. We further show that the corresponding decision-level errors accumulate additively over the abstraction scope.

Let Π_S and $\widetilde{\Pi}_S$ denote the joint distributions over the sequence of actions within S induced by the original policy π and an internal policy satisfying Theorem 1, respectively.

Corollary 1 (Accumulated Decision Error)**.** *Suppose that, for every decision t within S, the internal policy reproduces the original conditional action distribution with total variation error at most ϵ_t. Then*

$$D_{\mathrm{TV}}\left(\Pi_S, \widetilde{\Pi}_S\right) \leq \sum_{t \in S} \epsilon_t.$$

Proof. Couple the original and internal policies sequentially. At each decision t, conditioned on the two executions having agreed up to that decision, their conditional action distributions differ in total variation by at most ϵ_t. By maximal coupling, the probability that their actions first diverge at decision t is therefore at most ϵ_t. Applying a union bound over the decisions within S gives

$$\Pr[\text{the two action sequences differ}] \leq \sum_{t \in S} \epsilon_t.$$

Since total variation distance is bounded by the disagreement probability under any coupling,

$$D_{\mathrm{TV}}\left(\Pi_S, \widetilde{\Pi}_S\right) \leq \sum_{t \in S} \epsilon_t.$$

□

A.3 Pairwise Replay Characterization

For a candidate set C, define the maximum difference in action distributions between reachable histories that agree on C as

$$\Delta_t(C) = \sup_{\substack{h,h' \in \mathcal{H}_t^{\phi}: \\ h[C]=h'[C]}} D_{\mathrm{TV}}\left(\pi_t(\cdot \mid h), \pi_t(\cdot \mid h')\right).$$

Lemma 1 (Replay Characterization)**.** *For any candidate set C,*

$$\rho_t(C) \leq \Delta_t(C) \leq 2\rho_t(C).$$

Proof. For each value of $h[C]$, choose a representative history from the corresponding equivalence class and let κ_C equal the original action distribution at that representative history. This gives

$$\rho_t(C) \leq \Delta_t(C).$$

Conversely, for any κ_C that depends only on C, two histories h and h' with the same $h[C]$ share the same κ_C. By the triangle inequality for total variation distance, the difference between their action distributions is at most the sum of their respective distances to κ_C. Taking the supremum over histories and then the infimum over κ_C gives

$$\Delta_t(C) \leq 2\rho_t(C).$$

□

Therefore, $\Delta_t(C) = 0$ if and only if $\rho_t(C) = 0$. For approximate tests, the pairwise replay difference can similarly be converted into upper and lower bounds on $\rho_t(C)$.

A.4 Finite-Sample Dependency Recovery

Theorem 4 (Finite-Sample Dependency Recovery)**.** *Suppose there are N_H reachable histories at the target position and K possible actions, and the deletion search estimates at most Q conditional action distributions. Assume that the valid history generator samples each history with probability at least $p_{\min} > 0$, and that the true pairwise difference of every candidate is separated from the decision threshold by at least $\gamma > 0$. For any failure probability $\eta \in (0, 1)$, if the number of history replays N and the number of repeated action samples for each conditional action distribution M satisfy*

$$N \geq p_{\min}^{-1} \log \frac{2N_H}{\eta},$$

and

$$M \geq \frac{8K^2}{\gamma^2} \log \frac{4KQ}{\eta},$$

then, in a deletion search that covers all candidates, Suffice recovers all minimal sufficient information sets with probability at least $1 - \eta$. Repeated sampling is unnecessary when the conditional action distributions can be read exactly.

Proof. The probability that any particular history remains unobserved after N replays is at most $\exp(-Np_{\min})$. Taking a union bound over the N_H histories, the first condition ensures that the probability of incomplete coverage is at most $\eta/2$. Applying Hoeffding's inequality to the K action probabilities of each of the Q conditional distributions and taking a union bound, the second condition ensures that all empirical distributions simultaneously achieve the accuracy required by the classification margin, with failure probability at most $\eta/2$. Complete coverage together with correct classification guarantees that the deletion search returns the true minimal sufficient information sets. □

B Method Details

This appendix provides the details of state restoration and decision thresholds omitted from section 4.

B.1 Replay Construction and Sufficiency Test

For each decision, Suffice saves the model, system prompt, conversation context, tool definitions, tool session, permissions, environment snapshot, and decoding configuration of the original policy. Information outside the candidate set is completed using compatible trajectory branches, tool simulators, or environment snapshots. Generated histories must satisfy field types, API schemas, causal order, tool preconditions, session state, and permission constraints; combinations that are not actually reachable are excluded from replay.

Tool actions are normalized into the tool name, arguments, and control labels. When logits are available, the conditional action distributions are compared directly; otherwise, the same saved state is replayed repeatedly to estimate action frequencies. Histories, actions, and environment transitions are cached by decision to reuse completed agent calls.

Each test produces one of three outcomes. If the confidence upper bound on action deviation is below ϵ_{in}, C is marked as certified; if the confidence lower bound exceeds ϵ_{out}, C is marked as insufficient and accompanied by a counterexample; if coverage or sampling accuracy is insufficient, the result remains unknown. Valid histories, action distributions, counterexamples, and coverage status together form a query table indexed by (t, C).

B.2 Counterexample Reuse and Search Pruning

The deletion search reuses counterexamples obtained from previous replay queries to avoid repeatedly testing candidate sets that can already be ruled out. If a pair of reachable histories agree on a candidate set C but the original policy produces different action distributions, then C is insufficient to preserve the decision. Since any subset of C contains no more information than C, the same counterexample also rules out every $C' \subseteq C$. The search can therefore directly prune the corresponding branch bounded above by C without issuing additional replay queries.

Conversely, when C is determined to be sufficient, the search continues removing information variables to determine whether a smaller set remains sufficient. C is retained as a minimal sufficient information set only when it has been certified as sufficient and removing any necessary information produces a counterexample. Multiple minimal sets obtained through different deletion paths are retained separately, allowing the same decision to have alternative sufficient dependencies.

When the available replay evidence is insufficient to certify C as sufficient and does not provide a counterexample that rules out C, the candidate remains unknown and is not treated as a recovered dependency. The resulting minimal sufficient information sets are then encoded as dependency hyperedges as described in Section 4.2.

B.3 Discussion of Offline Abstraction Construction

Suffice performs dependency recovery and boundary inference offline when constructing an abstraction. Controlled replay queries may require substantial computation at this stage, but this cost is incurred during construction. The resulting abstraction is intended to be reused across future executions, allowing its one-time construction cost to be amortized over repeated online use. Our primary objective is therefore to derive abstractions that reduce recurring deployment costs while retaining task performance, rather than to minimize the one-time cost of abstraction construction.

C Experimental Details

This appendix provides additional details on the synthetic experiments and evaluation metrics omitted from Section 5.

Experiments use Python 3.11 and AppWorld 0.1.3.post1, with qwen3.8-27b at xhigh reasoning effort for the agent.

Table 2: Hyperedge F1 across hard dependency families.

Family	Suffice	ICAL
alternative	**1.000**	0.413
conjunction	**0.818**	0.786
entry_only	**0.992**	0.953
exit_only	**1.000**	**1.000**
nested	**1.000**	**1.000**
overlap	0.924	**0.946**
proxy	**0.972**	0.559
range_growth	**0.933**	0.923
rare_branch	**0.818**	0.771
xor	**0.989**	0.418

C.1 Synthetic Program Construction and Hard Dependency Families

Synthetic programs consist of a finite set of boolean variables and deterministic computation rules. In addition to randomly generated programs, we construct hard programs from 10 dependency families: alternative, conjunction, entry_only, exit_only, nested, overlap, proxy, range_growth, rare_branch, and xor. These families cover alternative sufficient paths, conjunctive dependencies, dependencies that cross only the input or output boundary, nested and overlapping scopes, proxy variables, dependency expansion, rare branches, and higher-order XOR dependencies.

Each hard program contains 10 boolean root variables and organizes the dependencies among root variables, intermediate computations, and target decisions according to its corresponding family. We enumerate all input assignments and re-execute each program to record intermediate states, target decisions, and information dependencies that cross the scope. This provides the complete ground-truth minimal sufficient information sets for each decision, from which we further determine all valid (S, J, E).

Ground-truth enumeration is independent of the evidence available to each method. The evaluated methods can access only the finite histories generated under a given history budget, so discriminative states may not be sufficiently observed from the available evidence. For example, the critical branch in rare_branch and the states that distinguish joint dependencies in xor may be sampled infrequently. This setup allows us to evaluate how well different methods recover the true dependencies from limited execution evidence while retaining complete ground-truth semantics for evaluation.

The final experiment contains 160 random programs and 40 hard programs covering the 10 dependency families above. All methods are evaluated under history budgets of 8, 16, 32, 64, and 128.

C.2 AppWorld Evaluation Details

To evaluate whether the learned abstractions can be effectively reused during real agent execution, we conduct an end-to-end evaluation on AppWorld. For all methods, we use a shared abstraction compiler that converts each generated (S, J, E) into a directly callable Tool/Skill, where S specifies the execution encapsulated within the Tool/Skill and J and E define its input and output interfaces, respectively. All methods use the same compiler, interface representation, and integration procedure, allowing their abstractions to be compared under a common execution environment.

The agent identifies and invokes the generated Tool/Skill through its natural-language description. To avoid method-specific manual design in Tool/Skill implementation, the abstraction compiler uses a third-party model independent of the abstraction methods to generate the natural-language description, interface schema, and internal execution logic from each method's (S, J, E) under the same procedure. The generated Tool/Skill can access only the external inputs defined by J and return information only through E. Each generated Tool/Skill is statically checked before execution, and any generation that violates the abstraction boundary is rejected and regenerated using the same procedure.

The end-to-end evaluation re-executes each complete task from the initial AppWorld environment state. The generated Tool/Skill is added to the agent's available tool set, and the agent autonomously decides whether and how to invoke the abstraction during normal reasoning and tool use before continuing execution until task completion. This evaluation therefore measures whether the reusable Tool/Skill constructed from an abstraction can preserve end-to-end task completion during actual agent execution.

We evaluate the final execution using the official AppWorld evaluator and report Scenario Goal Completion (SGC) following its official definition. A scenario is considered successful only if all tasks within that scenario pass the evaluator. All methods use the same set of scenarios, agent configuration, and abstraction integration procedure; only the generated abstractions differ across methods, enabling a controlled comparison of their end-to-end reusability.

We additionally report interface tokens to quantify the interface cost of each abstraction. For each encapsulated execution, we serialize the input interface J_i and output interface E_i retained by the method, and count their tokens using the same tokenizer as the agent context. The interface token cost is defined as

$$C_i = \mathrm{Tok}(J_i) + \mathrm{Tok}(E_i).$$

The overall interface tokens uses the same scenario-level macro aggregation as SGC:

$$C_{\mathrm{interface}} = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \frac{1}{|I_s|} \sum_{i \in I_s} \left[\mathrm{Tok}(J_i) + \mathrm{Tok}(E_i) \right].$$

Together with SGC, this metric evaluates the abstraction in terms of both end-to-end reusability and interface cost.