PURPOSE IR: COMPILING USER PURPOSE FOR PRIVACY-PRESERVING AGENT WORKFLOW STATE

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM-agent workflows routinely transform rich observations into plans, summaries, memories, handoffs, and other reusable state. This creates a privacy boundary that task relevance alone does not capture: an observation may be needed as evidence for a task without its full semantics being authorized to persist. We call this mismatch the *purpose-to-state semantic gap*. The key distinction is that *evidence authority is not release authority*. We introduce **Purpose IR**, a bounded typed intermediate representation that compiles natural-language user purpose into an executable semantic boundary for reusable agent state. A Purpose program is an acyclic semantic dataflow graph in which evidence leaves specify task-required semantics, typed dependencies constrain authorized derivations, and declared sinks determine which values or actions may cross the reusable-state boundary. A deterministic compiler constructs and validates the authoritative program, while a typed runtime reconstructs reusable state only from validated sink outputs. Under explicit execution contracts, this design yields *release confinement* and *purpose noninterference*, characterizing what factual semantics may be released and which observation differences must become irrelevant to reusable state. Across 1,200 cases spanning LangGraph, CrewAI, and MobileRun, Purpose IR achieves **99.6% Privacy**, **83.3% Task**, and **83.3% Privacy-Compliant Success (PCS)**, outperforming the strongest high-privacy baseline by **27.5 PCS points**. On 493 PrivacyLens trajectories, it achieves the highest **Safe Action Rate** at **77.3%**, 10.4 points above the next-best method, extending the same boundary from reusable state to downstream actions. These results show that privacy-preserving agent state requires more than filtering sensitive or task-irrelevant content: it requires an explicit, executable representation of what the user's purpose authorizes to survive. Code is available at `https://anonymous.4open.science/r/Purpose-IR-0C05`.

1 INTRODUCTION

LLM agents increasingly operate through multi-step workflows that combine language-model reasoning with tool use, environment observations, coordination, and persistent state (Yao et al., 2023; Schick et al., 2023; Wu et al., 2024; Park et al., 2023; Packer et al., 2023). During execution, these workflows routinely transform rich observations into plans, summaries, memories, handoffs, and other intermediate artifacts that are reused by later steps or later agents. This reuse is operationally useful, but it creates a distinct privacy boundary: information that is legitimately observed for computation can persist far beyond the semantic result the user actually requested.

Consider a user who asks, "Do I have a meeting reminder?" A workflow may inspect a notification containing a sender name, meeting time, office location, phone number, and full message body. These fields can all help determine whether the notification is a meeting reminder, yet the requested result is only a Boolean predicate. As illustrated in Figure 1, these task-relevant details can propagate through plans, summaries, memories, and handoff state, causing reusable state to preserve the semantics of the *observation* rather than those of the *requested result*. We call this mismatch the **purpose-to-state semantic gap**: the semantics retained in reusable workflow state can exceed the semantics required by the user's purpose. The key distinction is that the information an agent is authorized to inspect as evidence can be strictly richer than the information it is authorized to retain

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.

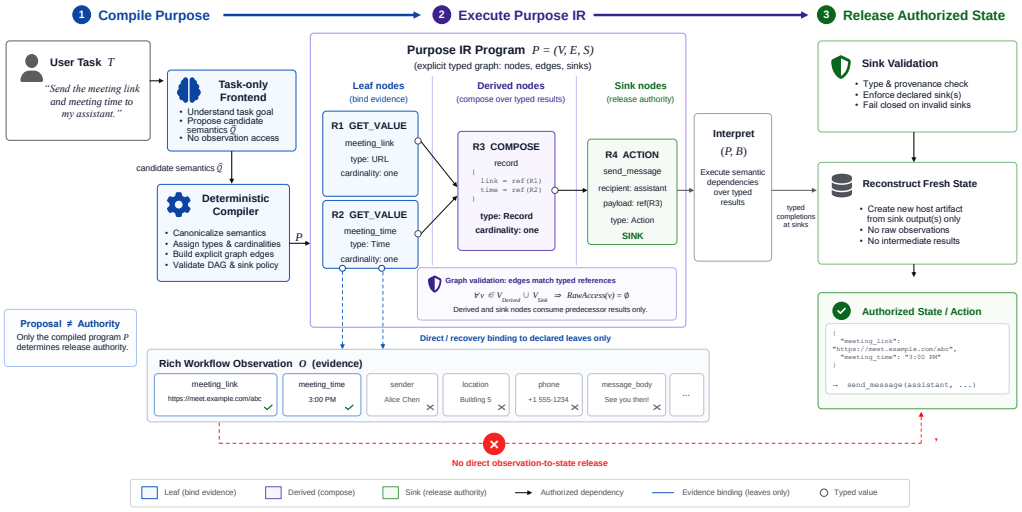


Figure 1: **The purpose-to-state semantic gap.** Task-relevant observation semantics may propagate through workflow intermediate states and persist in reusable state even when the user's authorized result is substantially narrower.

as reusable state. Task relevance therefore governs what may be useful as evidence, not what may persist as state.

The purpose-to-state semantic gap exposes three fundamental challenges. **(P1) Semantic specification.** A user's task specifies the desired outcome but not the boundary between semantics needed as evidence and those authorized to persist. **(P2) Purpose compilation.** Natural-language purpose is not directly executable and must be compiled into a representation that makes evidence, derivation, and release authority explicit. **(P3) Semantic preservation.** The compiled boundary must remain authoritative as information flows through heterogeneous workflow representations, preventing richer observation semantics from re-entering reusable state. Existing span-based privacy tools, contextual-privacy models, runtime policies, and information-flow controls address important parts of these challenges, such as detecting sensitive content, judging contextual relevance, or restricting propagation (Microsoft, 2026; Ngong et al., 2025; Bagdasarian et al., 2024; Costa et al., 2025; Wang et al., 2026; Siddiqui et al., 2025; Kolluri et al., 2026). Existing studies also reveal leakage in reasoning traces, inter-agent communication, memories, and actions (Green et al., 2025; El Yagoubi et al., 2026; Shao et al., 2024). What remains missing is a common, task-derived semantic boundary that specifies what result semantics may survive into reusable state and makes that boundary executable across workflows.

We introduce **Purpose IR** to provide this boundary. The core idea is simple: evidence may be richer than the result, but reusable state should not be. Purpose IR compiles a natural-language task into a bounded typed semantic dataflow program: evidence leaves specify what task semantics may be acquired, typed dependencies constrain how results may be derived, and declared sinks determine what values or actions may cross the reusable-state boundary. A task-only semantic frontend proposes candidate semantics, while a deterministic compiler constructs the authoritative program; workflow adapters ground its evidence requirements, and a typed runtime reconstructs reusable state only from validated sink outputs. Purpose IR therefore governs which semantics may survive, rather than how the agent performs the task.

This structure yields two formal properties that characterize the reusable-state boundary independently of any particular workflow representation. First, *release confinement* requires every factual semantic value in reusable state to be justified by the authorized ancestry of an explicit sink. Second, *purpose noninterference* requires observations that induce the same typed results at all sink-relevant Purpose leaves to produce equivalent reusable factual state under deterministic Purpose execution. Together, these properties capture both what information may be released and which observation differences must become irrelevant once the authorized result has been computed.

We evaluate Purpose IR in two complementary settings. The main evaluation contains 1,200 English workflow cases balanced equally across LangGraph, CrewAI, and MobileRun. Purpose IR achieves **99.6% Privacy**, **83.3% Task Success Rate**, and **83.3% Privacy-Compliant Success (PCS)**; the strongest high-privacy baseline reaches 55.8% PCS. We further evaluate external transfer on 493 PrivacyLens trajectories (Shao et al., 2024), where Purpose IR achieves the highest **Safe Action Rate** at **77.3%**, 10.4 points above the next-best method.

In summary, this paper makes three contributions aligned with the three challenges above:

(1) A semantic formulation of reusable-state privacy. We identify the **purpose-to-state semantic gap** and distinguish task-relevant evidence from the smaller result semantics authorized to persist. **(2) Purpose compilation into an executable semantic representation.** We introduce **Purpose IR**, a bounded typed dataflow IR that separates task-only semantic interpretation from deterministic authority construction and makes evidence, dependencies, and release/action sinks explicit. **(3) Semantic preservation with formal and cross-workflow evidence.** We formalize the compiled reusable-state boundary through *release confinement* and *purpose noninterference*, and evaluate it across three workflow frameworks and on PrivacyLens.

2 Related Work

Privacy leakage in agent workflows. Recent work shows that privacy leakage in LLM agents is not confined to final outputs. Green et al. (2025) demonstrate that reasoning traces of personal agents can contain sensitive user data, while AgentLeak instruments inter-agent messages, shared memory, tool arguments, and related channels, showing that output-only auditing can miss substantial internal leakage (El Yagoubi et al., 2026). Multi-agent contextual-privacy work studies how upstream privacy errors propagate through workflow topology (Li et al., 2025), and CIPL models privacy leakage across heterogeneous agent pipelines as a channel-inversion problem (Huang et al., 2026). Together, these studies establish intermediate workflow artifacts and internal communication as important privacy surfaces. Purpose IR focuses on the semantics retained when legitimately accessed observations are transformed into reusable workflow state, and on how those retained semantics can be bounded by the user's task.

Task-aware privacy and agent enforcement. Several lines of work incorporate task context into privacy protection or enforce explicit constraints over agent execution. AirGapAgent minimizes the private source information exposed to a conversational agent (Bagdasarian et al., 2024), while contextual-privacy approaches evaluate disclosure in terms of relevance and necessity for the user's goal (Ngong et al., 2025). Fides and related work apply deterministic information-flow controls to agent planning and LLM calls (Costa et al., 2025; Siddiqui et al., 2025; Kolluri et al., 2026); AgentSpec provides a DSL for runtime safety constraints (Wang et al., 2026); and agent firewalls, MNC, and CXI introduce structured or typed boundaries for agent communication and execution (Abdelnabi et al., 2025; Xu et al., 2026; Santos-Grueiro, 2026). AgentSecBench further separates data flow from authority and formalizes policy-enforced projections under intent-to-execution noninterference (Alpay & Alpay, 2026). Purpose IR contributes a task-derived semantic representation for this setting: natural-language purpose is compiled into typed evidence requirements, authorized derivations, and release sinks that govern the observation-to-state boundary.

Purpose, declassification, and policy languages. Purpose-based access control treats intended use as an explicit authorization dimension (Byun & Li, 2008; Kebede et al., 2023). Classical information-flow control and declassification formalize restrictions on propagation and controlled release (Sabelfeld & Myers, 2003; Sabelfeld & Sands, 2005), while systems such as Jeeves, Thoth, and Lifty use declarative policies or types to enforce privacy constraints (Yang et al., 2012; Elnikety et al., 2016; Polikarpova et al., 2020). Purpose IR builds on this broader view of explicit release authority and instantiates it at the LLM-agent state boundary. In particular, it lowers natural-language user purpose into a bounded typed dataflow IR that specifies which evidence semantics may be acquired, how typed results may be derived, and which values or actions may be released into reusable state.

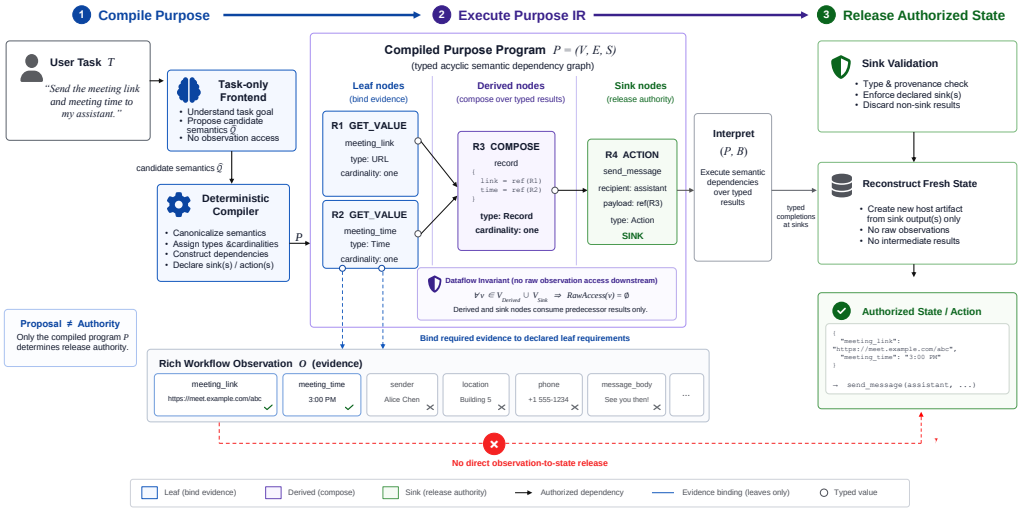


Figure 2: **Purpose IR overview.** User purpose is compiled into a typed semantic program that governs evidence binding, typed computation, and sink-constrained release.

Table 1: Purpose IR node roles. Only leaf nodes may acquire workflow evidence; downstream nodes operate on typed predecessor outputs.

Role	Operators	Input authority	Output
Leaf	`EXISTS`, `COUNT`, `GET_VALUE`, `DIAGNOSE`	Declared workflow evidence	Typed result
Derived	`COMPOSE`	Typed predecessor results	Schema-bounded value
Sink	`ACTION` / direct release	Literals + typed predecessors	Reusable value or action

3 METHOD

Purpose IR makes the observation-to-state boundary executable. Given a user task T and workflow observation O, the system compiles the task into a Purpose program, grounds the program in typed evidence, executes its semantic dataflow, and reconstructs only authorized outputs:

$$P = \text{Compile}(T), \qquad B = \text{Bind}(P, O), \qquad C = \text{Interpret}(P, B), \qquad A_{\text{out}} = \text{Release}(P, C).$$

The central separation is that O supplies evidence for the computation, whereas P defines the semantic authority under which results may enter reusable state. Figure 2 gives an overview of this compiler–runtime design.

3.1 PURPOSE IR: TYPED SEMANTIC DATAFLOW

A compiled Purpose program is a typed acyclic graph

$$P = (V, E, S),$$

where V is a finite set of typed Purpose nodes, E is the authorized dependency relation, and S is the set of release or action sinks. Each node declares an operator, an output schema, and typed predecessor references; leaf nodes additionally declare a semantic target and evidence cardinality. Optional activation guards are bounded predicates over predecessor values. The graph captures *semantic dependencies*, not agent control flow: tool selection, navigation, retry, and planning remain outside the IR.

Purpose nodes have three roles, summarized in Table 1: evidence-binding leaves, typed derived nodes, and release/action sinks. Only leaves may bind workflow evidence; derived and sink nodes operate only on typed predecessor results.

This role separation creates the central dataflow invariant. For every derived or sink node v,

$$\text{RawAccess}(v) = \varnothing, \qquad \text{Inputs}(v) \subseteq \{C_u : u \in \text{Pred}(v)\}.$$

Once observation content has been materialized as a typed Purpose result, downstream computation may use that result but cannot reopen the richer source observation that produced it.

For example, the task "Send the meeting link and meeting time to my assistant" compiles to the following Purpose program:

```
R1 = GET_VALUE(meeting_link, one)
R2 = GET_VALUE(meeting_time, one)
R3 = COMPOSE(record,
             meeting_link=ref(R1),
             meeting_time=ref(R2))
R4 = ACTION(send_message,
            destination="assistant",
            content=ref(R3))
sinks = [R4]
```

The program exposes exactly the semantics authorized by the task—the meeting link and time, their bounded composition, and a destination-scoped action— without prescribing how the upstream agent discovers those values. The graph representation becomes important when authorized results require multiple semantic values to be composed, conditionally propagated, or bound to structured actions. Purpose IR therefore represents the semantic computation from task-required evidence to authorized release, rather than merely selecting a flat set of fields.

3.2 Compiling Natural-Language Purpose

Compilation separates semantic interpretation from executable release authority. A Qwen3.5-0.8B frontend reads only the user task T and proposes candidate operations, semantic targets, multiplicity, and decomposition:

$$T \xrightarrow{M_\theta} \widehat{Q} \xrightarrow{\text{Compile}} P.$$

The frontend proposal $\widehat{Q}$ carries no execution or release authority. The deterministic Purpose Compiler resolves it into the canonical graph P by assigning semantic targets, operator contracts, schemas, cardinalities, dependencies, and sinks, and validating the result. It also separates task-level semantic targets, such as `meeting_link` or `meeting_time`, from generic observation carriers such as "text", "notification", or "message body", which cannot become release targets. Validation rejects malformed structure, including duplicate identifiers, unknown references, cycles, invalid guards or schemas, unsupported cardinalities, malformed action arguments, and derived references to raw observations. Once constructed, P is the sole authority consumed by the runtime; frontend proposals cannot directly expand evidence access or release.

The compiler provides executable authority rather than a guarantee that the resulting program perfectly captures the user's intended semantics. The formal properties below are therefore conditional on the compiled program P and the stated binding, operator, and reconstruction contracts; task-to-program semantic accuracy is evaluated empirically. Appendix A and Appendix C give the complete node schema and lowering rules.

3.3 Grounding and Typed Execution

Requirement-directed evidence binding. Workflow representations differ, so Purpose IR separates semantic requirements from evidence carriers. For each leaf node v, the workflow adapter produces transient bound evidence

$$B_v = \text{Bind}(v, O),$$

annotated with semantic type, value, assertion status, source kind, and record identity.

Direct Binding reads an explicitly declared native carrier when one is available. *Recovery Binding* may recover a missing value from a sealed raw observation through a declared carrier mapping. Recovery changes where evidence is obtained, not what the Purpose program is authorized to materialize or release: the recovered value must still satisfy the node's semantic type, assertion constraints, cardinality, and codec. An adapter may inspect a declared carrier to recognize or extract the requested semantic value, but only the validated typed binding is materialized into Purpose execution state. For scalar targets, binding therefore returns the validated scalar rather than the larger carrier that contains it.

Topological typed interpretation. The interpreter executes nodes in topological order. For a node v,

$$C_v = [\![v]\!] \left(B_v, \{C_u : u \in \mathrm{Pred}(v)\}\right),$$

with $B_v = \varnothing$ for derived and sink nodes.

`EXISTS` distinguishes verified presence, verified absence, and unknown; `COUNT` operates over distinct matching record identities; `GET_VALUE` validates type, codec, and declared cardinality; and `DIAGNOSE` produces only schema-declared bounded fields. `COMPOSE` can construct only values admitted by its output schema, and `ACTION` can consume only declared literals and typed predecessor references.

Ambiguous values, missing evidence, invalid cardinality, schema mismatch, or an incomplete sink leave the corresponding result unresolved rather than causing the runtime to widen the semantic release. Appendix D and Appendix B specify the binding and operator contracts.

3.4 Purpose-Constrained Release

Typed completions are computation state, not automatically reusable state. The release layer validates the declared sinks and reconstructs a fresh host-framework artifact from valid sink outputs only:

$$A_{\mathrm{out}} = \mathrm{Reconstruct}\left(\{C_s : s \in S,\ C_s \text{ valid}\}\right).$$

Source prose, evidence records, provenance, frontend candidates, compiler diagnostics, and non-sink completions do not become reusable merely because they participated in the computation.

Release is therefore a graph-level operation: factual content can enter reusable state only through validated sink outputs, and a typed dependency carries an authorized value without granting authority over unrelated fields from the same source observation. Dependency reachability and release authority are consequently distinct.

Returning to the meeting-reminder example, sender, time, location, phone number, and message body may all support the evidence-binding process for deciding whether a reminder exists, yet the reusable output can still be exactly the Boolean requested by the task. The richer carrier supports the computation without becoming part of the released result.

The Purpose language, interpreter, and release rule are shared across LangGraph, CrewAI, and MobileRun; workflow-specific code is confined to evidence adapters and native reconstruction. Appendix E and Appendix F provide the implementation and formal details.

3.5 Formal Properties

The graph discipline above yields two properties that characterize the semantic boundary enforced by Purpose IR. The properties concern factual semantics released into reusable state. Structured action sinks obey the same typed dependency and sink-validation discipline for their emitted arguments, while environment-level side effects are outside the scope of $\mathrm{FactSem}$ and of the two propositions below. We call an execution *valid* when leaf bindings respect their declared semantic targets and cardinalities, node evaluation respects the corresponding operator and schema contracts, and reconstruction emits factual content only from validated sink outputs.

Proposition 1 (Release confinement)**.** *Let $P = (V, E, S)$ be a valid Purpose IR program and consider a valid execution. For every factual semantic value x visible in the released artifact A_{out}, there exists a sink $s \in S$ such that x is contained in the semantic output of s and its factual provenance is confined to semantics authorized by the Purpose ancestors of s:*

$$x \in \mathrm{FactSem}(A_{\mathrm{out}}) \Longrightarrow \exists s \in S :\ x \in \mathrm{Sem}(C_s)\ \wedge\ \mathrm{Prov}(x) \subseteq \mathrm{Auth}(\mathrm{Anc}(s)).$$

Release confinement states that graph reachability cannot create authority. A value may flow to a sink only through typed dependencies already admitted by the Purpose program; neighboring fields in the original observation do not become releasable merely because they appeared in the same evidence carrier or supported the same computation.

For the second property, let $L_S \subseteq V$ denote the leaf ancestors of declared sinks. We call two observations *purpose-equivalent* for P when they induce the same typed completion at every sink-relevant leaf:

$$O_1 \equiv_P O_2 \quad \Longleftrightarrow \quad \forall \ell \in L_S,\ C_\ell(O_1) = C_\ell(O_2).$$

Proposition 2 (Purpose noninterference)**.** *For deterministic valid executions of P, if $O_1 \equiv_P O_2$, then the two observations induce the same reusable factual semantic state:*

$$O_1 \equiv_P O_2 \Longrightarrow \mathrm{FactSem}(A_{\mathrm{out}}(O_1)) = \mathrm{FactSem}(A_{\mathrm{out}}(O_2)).$$

The proof follows the DAG structure: equal sink-relevant leaf completions imply equal derived completions by induction over topological order, hence equal sink outputs; deterministic sink-only reconstruction then yields equal factual semantic state. Intuitively, once two observations agree on everything the compiled purpose is authorized to use for its sinks, all remaining differences in the raw observations become irrelevant to reusable factual state. Appendix F gives the execution contracts and proofs in detail.

4 Experiments

4.1 Evaluation Design

Main workflow-state benchmark. We evaluate on 1,200 English workflow cases spanning LangGraph, CrewAI, and MobileRun, with 400 cases per workflow. The three subsets share a common semantic task space while preserving their native workflow-state representations. Cases cover semantic-boundary conditions such as ambiguous evidence, negation, mixed private content, duplicate records, and semantic minimality. For each case, all methods receive the same user task and the same fixed workflow input.

We evaluate the downstream reusable artifact with a unified deterministic evaluator. **Privacy** measures whether the artifact excludes semantically unauthorized content, while **Task Success Rate** measures whether task-required result semantics remain available downstream. Our primary metric is their per-case conjunction, **Privacy-Compliant Success (PCS)**,

$$\mathrm{PCS}(i) = \mathrm{Privacy}(i) \wedge \mathrm{TaskSuccess}(i).$$

External transfer. We additionally evaluate on 493 PrivacyLens privacy-sensitive action trajectories (Shao et al., 2024). Unlike the main benchmark, this setting evaluates downstream actions rather than reusable workflow state. We follow the action-level protocol with a fixed local Mistral-7B-Instruct-v0.2 judge, while Purpose IR uses the same task-only Qwen3.5-0.8B semantic frontend as in the main evaluation.

We report **Privacy Pass**, **Helpfulness** ≥ 2, and their per-trajectory conjunction, **Safe Action Rate**,

$$\mathrm{SafeAction}(i) = \mathrm{PrivacyPass}(i) \wedge [\mathrm{Helpfulness}(i) \geq 2].$$

Baselines. On the main benchmark, we compare against No Guard, Presidio (Microsoft, 2026), IBM Contextual Privacy LLM (Ngong et al., 2025), Clean Task-text Guard, Task-derived Structured Projection, Fides constrained inspection (Costa et al., 2025), and AgentSpec (Wang et al., 2026). On PrivacyLens, we compare against Full trajectory, Task-text reduction, Presidio, AgentSpec, and Fides on the same 493 trajectories. Detailed baseline definitions, model roles, and evaluator settings are provided in Appendix G.3.

Compositional compilation. Beyond end-to-end workflow performance, we evaluate purpose compilation on a controlled 300-instance compositional suite requiring explicit semantic dependencies, including argument binding, record composition, conditional routing, grouping, and multi-stage dependency. We report validator-valid IR, semantic-contract accuracy, and privacy, isolating whether the frontend and compiler recover the intended computation rather than merely the authorized leaf semantics.

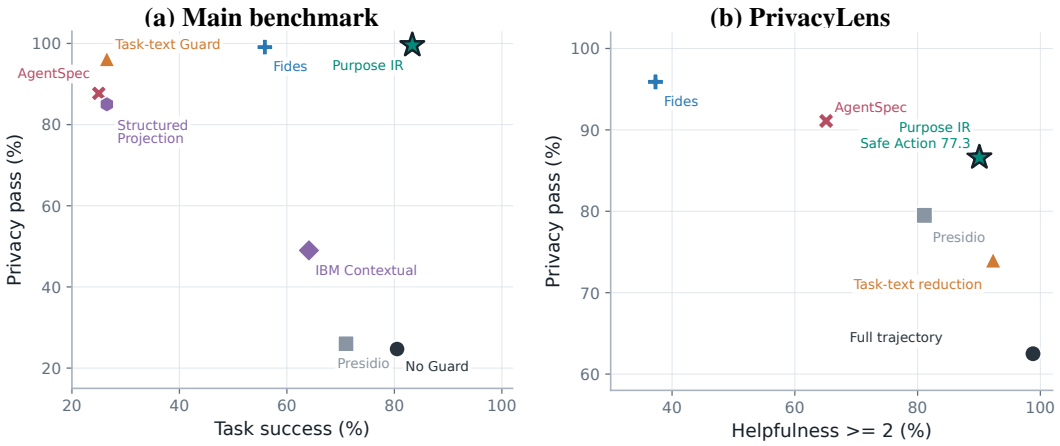


Figure 3: **Privacy–utility trade-offs.** (a) Task Success Rate versus Privacy on the main benchmark. (b) Helpfulness versus Privacy Pass on PrivacyLens.

Table 2: **End-to-end results.** PCS and Safe Action require privacy and utility on the same case or trajectory.

(a) Main benchmark

	Primary	Secondary	
Method	**PCS ↑**	Privacy	Task Success Rate
No Guard	**23.9**	24.7	80.5
Presidio	**20.8**	26.0	71.0
IBM Context.	**32.3**	49.0	64.1
Task-text	**25.3**	96.2	26.5
Struct. Proj.	**24.4**	85.0	26.5
Fides	**55.8**	99.1	55.9
AgentSpec	**16.2**	87.7	25.0
Purpose IR	**83.3**	99.6	83.3

(b) PrivacyLens

	Primary	Secondary	
Method	**Safe Action ↑**	Priv. Pass	Help.≥2
Full traj.	**61.7**	62.5	98.8
Task-text	**66.9**	74.0	92.3
Presidio	**62.3**	79.5	81.1
AgentSpec	**56.2**	91.1	65.1
Fides	**34.1**	95.9	37.3
Purpose IR	**77.3**	86.6	90.1

4.2 End-to-End Privacy–Utility Results

Table 2(a) reports results on the main workflow-state benchmark. Purpose IR reaches **99.6% Privacy**, **83.3% Task Success Rate**, and **83.3% Privacy-Compliant Success (PCS)**, improving PCS by **27.5 percentage points** over Fides at a comparable privacy level. The baselines expose the expected privacy–utility trade-off: broad retention preserves task information but also unauthorized semantics, whereas conservative filtering suppresses task utility. Structured Projection reaches only 24.4% PCS, showing that task-aware field selection alone is insufficient. Across individual workflows, Privacy ranges from 99.5–99.8% and PCS from 79.8–85.5%, with 100% validator-valid IR across all three frameworks; see Table 10 in the appendix. Figure 3(a) visualizes this high-privacy, high-utility operating point; the Purpose language, compiler, and interpreter are shared across workflows, with only evidence grounding and native reconstruction being adapter-specific.

External transfer. Table 2(b) and Figure 3(b) show the same pattern on 493 PrivacyLens trajectories. Purpose IR achieves the highest **Safe Action Rate** at **77.3%**, **10.4 percentage points** above Task-text reduction. Full trajectory favors utility and Fides favors privacy, whereas Purpose IR achieves the strongest conjunction of the two. Because PrivacyLens changes both the input representation and the evaluated release object, this result provides external evidence beyond the main workflow-state benchmark.

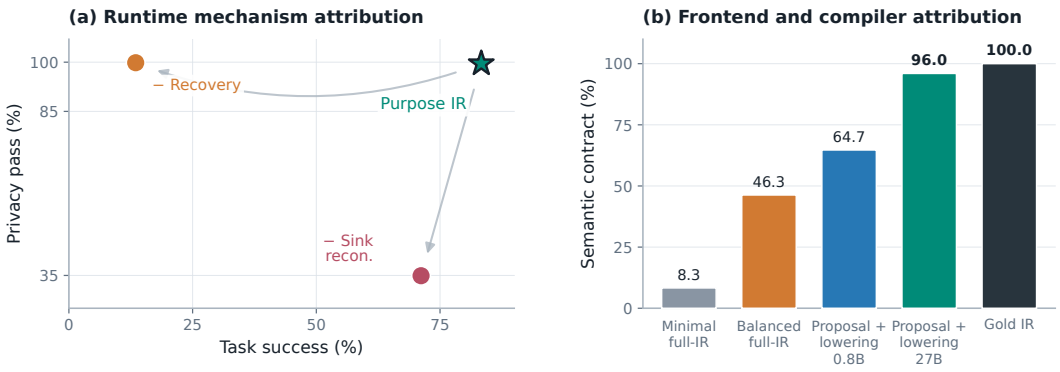


Figure 4: **Mechanism and compilation attribution.** (a) Paired runtime ablations on the 1,200-case main benchmark. (b) Semantic-contract accuracy on the controlled 300-instance compositional suite.

4.3 Mechanism Analysis

We isolate two complementary sources of performance: runtime enforcement on the main workflow benchmark and semantic compilation on the compositional suite. Figure 4 summarizes both analyses.

Runtime mechanisms. All runtime variants replay the same 1,200 compiled Purpose programs with no additional model calls and use the same deterministic evaluator as Full Purpose IR. Removing recovery binding leaves Privacy at 99.8% but reduces Task Success Rate and PCS from 83.3% to **13.5%**. In contrast, removing sink-constrained reconstruction retains 71.2% Task Success Rate but reduces Privacy from 99.6% to **34.9%** and PCS to **24.2%**. The mean released artifact grows from 203.16 to 679.40 bytes without sink-constrained reconstruction. These paired interventions separate the two runtime roles: recovery binding supplies task-required typed evidence, whereas reconstruction determines which computed semantics may persist.

Frontend and compiler. On the controlled compositional suite, semantic-contract accuracy rises from 25/300 with minimal direct full-IR prompting to 139/300 with balanced prompting, and to **194/300** with 0.8B semantic proposal plus deterministic lowering. Under the same compilation and evaluation protocol, a 27B semantic frontend reaches **288/300**. Privacy is 300/300 for all evaluated variants. This progression separates executable IR construction from semantic interpretation and shows the benefit of deterministic lowering together with stronger semantic frontends.

Finally, we validate the language boundary independently of end-to-end task performance. Contract-mutation tests reject invalid cardinalities, dangling dependencies, raw-observation references, invalid sinks, and dependency cycles before execution. Cardinality-specific tests further confirm that ambiguous `cardinality=one` evidence is withheld rather than widened into a multi-value release. Together, these checks verify that the structural constraints assumed by Purpose IR are enforced by the implementation. Detailed intervention results are reported in Appendix H.2.

5 Conclusion

We introduced **Purpose IR**, a compiler-based approach to governing what semantics may persist in reusable LLM-agent state. Its central distinction is between evidence authority and release authority: rich observations may be needed to compute a task result without their full semantics being authorized to persist. Purpose IR makes this boundary executable through typed semantic dependencies and explicit release sinks, with *release confinement* and *purpose noninterference* characterizing the resulting reusable-state semantics. Our evaluations show that this boundary can preserve task utility while substantially reducing unauthorized retention. More broadly, agent privacy should govern not only what information may be accessed, but also what semantics the user's purpose authorizes to survive.

AI USE STATEMENT

We partially used generative AI tools for code development and for improving the clarity and readability of the manuscript. All AI-assisted code and text modifications were carefully reviewed, verified, and tested by the authors.

ETHICS STATEMENT

This work does not involve human subjects, sensitive personal data, or applications that raise significant ethical concerns. We have considered potential ethical risks associated with this research and found no additional issues requiring further discussion.

REPRODUCIBILITY STATEMENT

We are committed to making this work reproducible. In the main paper and appendix, we provide as much detail as possible about the method, model architecture, and experimental details needed to reproduce this work, and we provide an anonymized code repository link to facilitate reproduction.

REFERENCES

Sahar Abdelnabi, Amr Gomaa, Eugene Bagdasarian, Per Ola Kristensson, and Reza Shokri. Firewalls to secure dynamic LLM agentic networks. *arXiv preprint arXiv:2502.01822*, 2025. URL https://arxiv.org/abs/2502.01822.

Faruk Alpay and Taylan Alpay. AgentSecBench: Measuring prompt injection, privacy leakage, and tool-use integrity in LLM agents. *arXiv preprint arXiv:2605.26269*, 2026. URL https://arxiv.org/abs/2605.26269.

Eugene Bagdasarian, Ren Yi, Sahra Ghalebikesabi, Peter Kairouz, Marco Gruteser, Sewoong Oh, Borja Balle, and Daniel Ramage. AirGapAgent: Protecting privacy-conscious conversational agents. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security*, pp. 3868–3882. Association for Computing Machinery, 2024. doi: 10.1145/3658644.3690350. URL https://doi.org/10.1145/3658644.3690350.

Ji-Won Byun and Ninghui Li. Purpose based access control for privacy protection in relational database systems. *The VLDB Journal*, 17(4):603–619, 2008. doi: 10.1007/s00778-006-0023-0. URL https://doi.org/10.1007/s00778-006-0023-0.

Manuel Costa, Boris Köpf, Aashish Kolluri, Andrew Paverd, Mark Russinovich, Ahmed Salem, Shruti Tople, Lukas Wutschitz, and Santiago Zanella-Béguelin. Securing AI agents with information-flow control. *arXiv preprint arXiv:2505.23643*, 2025. URL https://arxiv.org/abs/2505.23643.

Faouzi El Yagoubi, Godwin Badu-Marfo, and Ranwa Al Mallah. AgentLeak: A benchmark for internal-channel privacy leakage in multi-agent LLM systems. *IEEE Access*, 14:94960–94978, 2026. doi: 10.1109/ACCESS.2026.3704541. URL https://doi.org/10.1109/ACCESS.2026.3704541.

Eslam Elnikety, Aastha Mehta, Anjo Vahldiek-Oberwagner, Deepak Garg, and Peter Druschel. Thoth: Comprehensive policy compliance in data retrieval systems. In *25th USENIX Security Symposium (USENIX Security 16)*, pp. 637–654. USENIX Association, 2016. URL https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/elnikety.

Tommaso Green, Martin Gubri, Haritz Puerto, Sangdoo Yun, and Seong Joon Oh. Leaky thoughts: Large reasoning models are not private thinkers. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 26507–26529. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.1347. URL https://aclanthology.org/2025.emnlp-main.1347/.

Tao Huang, Chen Hou, Guosen Wu, and Jiayang Meng. Observable channels, not just storage: Evaluating privacy leakage in LLM agent pipelines. *arXiv preprint arXiv:2603.22751*, 2026. URL https://arxiv.org/abs/2603.22751.

Milen G. Kebede, L. Thomas van Binsbergen, Tom M. van Engers, and Dannis G. van Vuurden. Towards a purpose-based access control model derived from the purpose limitation principle. In *Legal Knowledge and Information Systems: JURIX 2023*, volume 379 of *Frontiers in Artificial Intelligence and Applications*, pp. 143–148. IOS Press, 2023. doi: 10.3233/FAIA230958. URL https://doi.org/10.3233/FAIA230958.

Aashish Kolluri, Rishi Sharma, Manuel Costa, Boris Köpf, Tobias Nießen, Mark Russinovich, Shruti Tople, and Santiago Zanella-Béguelin. Optimizing agent planning for security and autonomy. In *International Conference on Learning Representations*, 2026. URL https://iclr.cc/virtual/2026/poster/10008186.

Wenkai Li, Liwen Sun, Zhenxiang Guan, Xuhui Zhou, and Maarten Sap. 1-2-3 check: Enhancing contextual privacy in LLM via multi-agent reasoning. In *Proceedings of the First Workshop on LLM Security (LLMSEC)*, pp. 115–128. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.llmsec-1.9/.

Microsoft. Presidio: Data protection and de-identification sdk. Open-source software, 2026. URL https://github.com/microsoft/presidio. Accessed 2026-08-31.

Ivoline C. Ngong, Swanand Ravindra Kadhe, Hao Wang, Keerthiram Murugesan, Justin D. Weisz, Amit Dhurandhar, and Karthikeyan Natesan Ramamurthy. Protecting users from themselves: Safeguarding contextual privacy in interactions with conversational agents. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 26196–26220. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-acl.1343. URL https://aclanthology.org/2025.findings-acl.1343/.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023. URL https://arxiv.org/abs/2310.08560.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pp. 1–22, 2023. doi: 10.1145/3586183.3606763. URL https://doi.org/10.1145/3586183.3606763.

Nadia Polikarpova, Deian Stefan, Jean Yang, Shachar Itzhaky, Travis Hance, and Armando Solar-Lezama. Liquid information flow control. *Proceedings of the ACM on Programming Languages*, 4(ICFP):105:1–105:30, 2020. doi: 10.1145/3408987. URL https://doi.org/10.1145/3408987.

Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, 2003. doi: 10.1109/JSAC.2002.806121. URL https://doi.org/10.1109/JSAC.2002.806121.

Andrei Sabelfeld and David Sands. Dimensions and principles of declassification. In *18th IEEE Computer Security Foundations Workshop*, pp. 255–269, 2005. doi: 10.1109/CSFW.2005.15. URL https://doi.org/10.1109/CSFW.2005.15.

Igor Santos-Grueiro. Context-to-execution integrity for LLM agents. *arXiv preprint arXiv:2607.06000*, 2026. URL https://arxiv.org/abs/2607.06000.

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, pp. 68539–68551, 2023. doi: 10.52202/075280-2997. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.

Yijia Shao, Tianshi Li, Weiyan Shi, Yanchen Liu, and Diyi Yang. PrivacyLens: Evaluating privacy norm awareness of language models in action. In *Advances in Neural Information Processing Systems*, volume 37, pp. 89373–89407, 2024. doi: 10.52202/079017-2837. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/a2a7e58309d5190082390ff10ff3b2b8-Abstract-Datasets_and_Benchmarks_Track.html.

Shoaib Ahmed Siddiqui, Radhika Gaonkar, Boris Köpf, David Krueger, Andrew Paverd, Ahmed Salem, Shruti Tople, Lukas Wutschitz, Menglin Xia, and Santiago Zanella-Béguelin. Permissive information-flow analysis for large language models. *Transactions on Machine Learning Research*, 2025. URL https://openreview.net/forum?id=ufYRO8y3mr.

Haoyu Wang, Christopher M. Poskitt, and Jun Sun. AgentSpec: Customizable runtime enforcement for safe and reliable LLM agents. In *2026 IEEE/ACM 48th International Conference on Software Engineering (ICSE)*, 2026. doi: 10.1145/3744916.3764546. URL https://doi.org/10.1145/3744916.3764546.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversations. In *Proceedings of the First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=BAakY1hNKS.

Jinghan Xu, Longze Fan, Zeyuan Wang, Xinjin Li, and Hankai Liu. MNC: Scope-bound semantic declassification for private LLM-agent communication. *arXiv preprint arXiv:2608.01719*, 2026. URL https://arxiv.org/abs/2608.01719.

Jean Yang, Kuat Yessenov, and Armando Solar-Lezama. A language for automatically enforcing privacy policies. In *Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 85–96, 2012. doi: 10.1145/2103656.2103669. URL https://doi.org/10.1145/2103656.2103669.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. 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.

A Purpose IR Specification

This appendix specifies the bounded typed dataflow language used by Purpose IR and the static constraints enforced before execution.

A.1 Language boundary

Purpose IR is a semantic dataflow language for user purpose, not a full agent execution language. It represents task-required evidence semantics, typed dependencies among authorized intermediate results, and declared release or action sinks. Tool selection, navigation strategy, retry policy, loops, recursion, and unrestricted user-defined computation remain outside the IR.

A Purpose program is

$$P = (V, E, S),$$

where V is a finite set of typed nodes, E is an acyclic dependency relation, and S is the set of declared sinks. The operator vocabulary is partitioned by role:

$$\mathcal{O}_{\text{leaf}} = \{\texttt{EXISTS}, \texttt{COUNT}, \texttt{GET_VALUE}, \texttt{DIAGNOSE}\},$$

$$\mathcal{O}_{\text{derived}} = \{\texttt{COMPOSE}\}, \qquad \mathcal{O}_{\text{sink}} = \{\texttt{ACTION}\}.$$

A completed leaf or derived node may also be declared directly as a release sink when the task result itself is the reusable output.

A.2 Canonical node contract

Each node has a stable identifier, operator, output schema, and optional dependency metadata. Leaf nodes additionally declare a semantic target and evidence cardinality. Derived and sink nodes reference predecessor results explicitly. Table 4 summarizes the core fields.

A representative compositional program is:

```
R1 = GET_VALUE(error_code, cardinality=one)
R2 = GET_VALUE(device_model, cardinality=one)
R3 = COMPOSE(record,
             error_code=ref(R1),
             device_model=ref(R2))
R4 = ACTION(send_message,
            destination="administrator",
            content=ref(R3))
sinks = [R4]
```

The graph makes internal and external values explicit. $R1$–$R3$ can complete without becoming reusable state; $R4$ is the declared sink.

A.3 Semantic targets and cardinality

Semantic targets belong to a bounded registered namespace with auditable binding paths. Examples include `meeting_reminder`, `meeting_link`, `tracking_number`, `error_code`, `device_model`, `app_version`, and `delivery_status`. Generic carriers such as `text`, `clipboard`, `notification`, `screen`, or `message_body` are not semantic targets.

Leaf nodes additionally declare bounded evidence cardinality. The implementation supports single-value and bounded multi-value requirements, including `one`, `many`, `at_most_k`, and `all_matching`. Cardinality is checked during binding and materialization rather than inferred from field order at runtime.

A.4 Dependency, composition, and sink semantics

Dependencies are first-class typed references. A node may consume only declared predecessor outputs, and the compiler rejects unknown references and cycles. Derived `COMPOSE` nodes construct schema-bounded records or templates from predecessor values; they do not receive an observation handle. Structured `ACTION` nodes similarly consume declared literals and typed predecessor results and define the downstream destination and arguments.

This separation yields the leaf-access invariant

$$v \in V_{\text{derived}} \cup V_{\text{sink}} \Longrightarrow \text{RawAccess}(v) = \varnothing.$$

A dependency therefore transports an already-authorized typed value; it does not expand the semantic authority of the consumer.

B Operator and Failure Semantics

Table 3 summarizes the typed operator roles.

B.1 Three-valued existence semantics

The runtime distinguishes verified absence from insufficient evidence:

$$s \in \{\text{PRESENT}, \text{ABSENT}, \text{UNKNOWN}\}.$$

The operator semantics are

$$\text{EXISTS}(s) = \begin{cases} \text{true}, & s = \text{PRESENT}, \\ \text{false}, & s = \text{ABSENT}, \\ \bot, & s = \text{UNKNOWN}. \end{cases}$$

Table 3: Typed Purpose IR operator contracts.

Operator	Role	Input	Output
EXISTS	Leaf	Verified predicate evidence	Boolean
COUNT	Leaf	Verified matching records	Non-negative integer
GET_VALUE	Leaf	Validated typed evidence	Typed scalar/collection
DIAGNOSE	Leaf	Valid schema-matched evidence	Bounded structured result
COMPOSE	Derived	Typed predecessor outputs	Schema-bounded value
ACTION	Sink	Literals + typed predecessor outputs	Structured action arguments

Table 4: Canonical Purpose IR node fields.

Field	Meaning	Deterministic constraint
id	Stable node identifier	Unique within the graph
operator	Purpose operation	Registered operator
target_type	Semantic evidence type for leaf nodes	Registered semantic target
output_schema	Typed result shape	Compatible with operator
cardinality	Evidence multiplicity	Declared bounded mode
depends_on	Predecessor node identifiers	Existing nodes; acyclic graph
condition	Optional typed activation guard	References typed predecessor results
arguments	Structured inputs for composition/action	Literals or typed references only
release	Node-local release metadata	Consistent with sink declaration
grounding	Evidence-binding metadata	Descriptive; grants no authority

A verified negative answer is a valid task result. Missing or conflicting evidence remains unresolved.

B.2 Counting, values, and cardinality

COUNT accepts a validated non-negative aggregate when the adapter establishes that it represents the requested count; otherwise it counts distinct matching records. With stable record identities,

$$\mathrm{COUNT}(E) = |\{\mathrm{record_id}(e) : e \in E\}| .$$

GET_VALUE validates semantic type, codec, and declared cardinality. A one requirement must resolve to one unique value; bounded multi-value modes validate their declared limits. Conflicting candidates fail closed rather than being resolved by iteration order.

A scalar result is released as the validated scalar, not as a larger carrier containing adjacent semantics. For example,

```
value=E42; workspace=group-alpha; owner=member-7
```

is not a valid scalar representation of E42.

B.3 Bounded diagnosis and composition

DIAGNOSE produces only fields declared by its output schema. It is not a license to copy raw debug prose, screenshots, source quotes, or complete native records. COMPOSE operates only over predecessor completions and must satisfy its declared result schema. The derived node has no raw-observation interface.

B.4 Structured action sinks

ACTION is a sink rather than an unrestricted text generator. Its destination and argument schema are declared by the Purpose program, and each argument is either a contract literal or a typed reference to a predecessor completion. The runtime validates the complete sink before release. A sink cannot widen its inputs by consulting source text, raw observations, or undeclared intermediate state.

Table 5: Representative result codecs.

Codec	Accepted output	Rejected output
`boolean`	Boolean value	Arbitrary prose or record-shaped text
`integer`	Non-negative integer	Negative or malformed number
`uri`	Exactly one HTTP(S) URI	Zero/multiple URIs or mixed span
`bounded_scalar`	Single typed scalar	Dict/list/multiline carrier
`object/array`	Schema-valid structured value	Extra or ill-typed fields

B.5 Representative codecs

B.6 Fail-closed behavior

The runtime treats uncertainty or invalid graph structure as a reason to withhold release rather than widen it. Representative failures include unsupported operators or targets, duplicate identifiers, unknown dependencies, dependency cycles, invalid conditions, unsupported cardinality, ambiguous evidence, codec failure, schema mismatch, missing action arguments, derived raw-observation references, incomplete sinks, and reconstruction postcondition violations. Graph-level validation therefore extends the value-level fail-closed behavior of the leaf operators to the entire Purpose program.

C Semantic Frontend and Purpose Compiler

The compiler separates neural semantic proposal from deterministic authority construction.

C.1 Task-only semantic frontend

The Qwen3.5-0.8B frontend reads only the natural-language task. It does not observe native workflow state, raw observations, artifact fields, privacy labels, or benchmark annotations. It proposes candidate operations, semantic targets, multiplicity, and task decomposition for the compiler. The proposal carries no release authority.

C.2 Deterministic lowering and static validation

The Purpose Compiler normalizes operator and target names, validates targets against the semantic registry, constructs leaf requirements, assigns output schemas and cardinalities, resolves dependencies, constructs structured sinks, and checks the resulting DAG. Static validation rejects duplicate identifiers, unknown references, cycles, incompatible schemas, invalid cardinality, malformed typed guards, illegal sink arguments, and derived references to raw observations.

Only the compiler-admitted canonical graph is consumed by the runtime.

C.3 Scope of the language

Purpose IR is deliberately bounded around semantic computation and release. It supports typed evidence requirements, acyclic semantic dependencies, bounded cardinality, typed activation conditions, schema-bounded composition, and structured sinks. General agent control flow—including loops, recursion, arbitrary tool planning, unrestricted expressions, and mutable program state—remains the responsibility of the host agent rather than the Purpose IR.

D Fact IR, Binding, and Recovery

Purpose IR is workflow-independent, but evidence representations are not. The runtime therefore converts native workflow state into a common transient evidence interface before typed execution.

D.1 FACT IR

The native draft is atomized into complete information units. A representative Fact IR atom is:

```
{
  "atom_id": "a17",
  "fact_id": "f17",
  "source_path": ["answer", "text"],
  "role": "CONTENT",
  "assertion_status": "ASSERTED",
  "structural_context": "answer.text",
  "proposition":
      "meeting_link=https://meet.example.com/j/8421"
}
```

Fact IR separates semantic content from processing metadata. Source paths, provenance, source quotes, control schema, and reasoning traces remain internal metadata; their presence does not make them factual task outputs.

D.2 BOUND EVIDENCE

The shared interpreter consumes transient `BoundEvidence` objects:

```
{
  "requirement_id": "R1",
  "semantic_type": "meeting_link",
  "value": "https://meet.example.com/j/8421",
  "source_kind": "direct_binding",
  "assertion_status": "ASSERTED",
  "role": "CONTENT",
  "record_id": "record-17"
}
```

Before materialization, the runtime validates the requirement identifier, assertion state, evidence role, and semantic type. Bound evidence is transient; it is not itself reusable workflow state.

D.3 DIRECT AND RECOVERY BINDING

Direct Binding. Direct Binding obtains typed evidence from structured or already exposed workflow state. It is preferred when the adapter has an explicit carrier for the requested semantic type.

Recovery Binding. When Direct Binding is insufficient, the adapter may recover missing evidence from the sealed raw observation. Recovery is requirement-directed and uses declared carrier mappings. A recovered value must still match the requirement identifier and semantic type, have an admissible role and assertion state, and pass the corresponding codec.

Recovery therefore changes where evidence is obtained, not what the task is authorized to release.

D.4 REPRESENTATIVE CARRIER MAPPINGS

Carrier aliases belong to workflow adapters. The shared Purpose interpreter does not contain native field names.

D.5 AMBIGUITY AND MIXED SPANS

For scalar and action requirements, adapters may rank declared candidate carriers. If two equally admissible candidates contain conflicting values, the runtime returns no binding instead of choosing one according to iteration or field order.

Table 6: Representative semantic targets and adapter-owned evidence carriers.

Semantic target	Representative carrier	Codec	Validation
`meeting_link`	Declared URL field/carrier	`uri`	Exactly one URI
`tracking_number`	Declared value field	`bounded_scalar`	Single-line scalar
`error_code`	Error-code carrier	`bounded_scalar`	No mixed record span
`device_model`	Device/debug record field	`bounded_scalar`	Exact semantic type
`app_version`	Version field	`bounded_scalar`	Single-line scalar
`meeting_reminder`	Typed notification category	`boolean`	Predicate + assertion
`unread_count`	Count/unread aggregate	`integer`	Non-negative integer
`crash_stack`	Declared debug field	`structured_summary`	Structured evidence

Table 7: Shared and workflow-specific components.

Component	Shared	Workflow-specific
Purpose language	✓	
Operator semantics	✓	
Compiler contract validation	✓	
Typed codecs	✓	
Release-authority checks	✓	
Fact IR atomization		✓
Native carrier aliases		✓
Direct Binding		✓
Recovery Binding		✓
Native reconstruction		✓
Workflow contamination checks		✓

Likewise, a prose span is not accepted as a scalar merely because the desired value appears within it. URI extraction requires one unambiguous URI, and the bounded scalar codec rejects multiline or record-shaped spans containing adjacent `field=value` material. This constraint is semantic rather than a blacklist of private keywords: the released object must have the shape authorized by the Purpose requirement.

E WORKFLOW ADAPTERS

The Purpose language and interpreter are shared across workflows, while representation-specific operations remain in adapters. Table 7 summarizes this boundary.

LangGraph. LangGraph cases expose structured graph-state surfaces. The adapter traverses native state, atomizes content with path information, maps declared typed carriers into BoundEvidence, and reconstructs graph-compatible state from authorized results.

CrewAI. CrewAI cases expose task, description, result, plan, answer, and handoff-like prose surfaces. The adapter treats them as separate native carriers during atomization and reconstruction. Repeated prose surfaces do not survive merely because they are part of the native task object.

MobileRun. MobileRun cases expose manager, executor, memory, action, and related state surfaces. The adapter maps these native paths into the same evidence interface and reconstructs only the authorized result into the host schema.

The language, operator semantics, and release rule remain fixed across the three workflows; adapters provide the representation-specific translation into and out of the shared typed evidence interface. This separation is the portability boundary evaluated in our main suite.

F FORMAL PROPERTIES

This section states the execution contracts and proof obligations behind Propositions 1 and 2. These contracts define valid execution for the shared Purpose IR semantics.

F.1 EXECUTION CONTRACTS

Let $P = (V, E, S) = \mathrm{Compile}(T)$ be a valid acyclic Purpose program and O a workflow observation. A valid execution satisfies the following contracts.

1. **Binding contract.** For every leaf ℓ, each value in $B_\ell = \mathrm{Bind}(\ell, O)$ denotes the semantic target declared by ℓ, satisfies its assertion constraints, and preserves record identity when required by the operator.
2. **Operator and codec contract.** For every node v, $[\![v]\!]$ produces only values admitted by the node's declared output type, cardinality, and schema. Derived and sink nodes read only typed predecessor outputs and declared literals.
3. **Reconstruction contract.** $\mathrm{Reconstruct}$ emits factual semantic content only from validated sink outputs; source observations, bound evidence, provenance, compiler diagnostics, and non-sink completions are not copied into factual reusable state.
4. **Determinism** (for Proposition 2). Given the same typed predecessor completions, guards, operator semantics, and sink outputs, graph execution and factual reconstruction are deterministic.

Together, these contracts define the executions to which the two propositions apply.

F.2 PROOF OF RELEASE CONFINEMENT

For each non-leaf node v, the language invariant requires

$$\mathrm{RawAccess}(v) = \varnothing, \qquad \mathrm{Inputs}(v) \subseteq \{C_u : u \in \mathrm{Pred}(v)\}.$$

Consider any $x \in \mathrm{FactSem}(A_{\mathrm{out}})$. By the reconstruction contract, x must be emitted from a validated sink output C_s for some $s \in S$. Each factual component of C_s is either a contract literal or is obtained through a typed predecessor reference. Following predecessor references backward in the finite acyclic graph reaches only leaf completions and declared literals. By the binding contract, each leaf completion denotes its declared semantic target; by the operator and codec contract, derived nodes cannot introduce factual inputs outside their declared predecessor semantics. Therefore every factual provenance path for x is contained in semantics authorized by $\mathrm{Anc}(s)$. This establishes Proposition 1.

The result is a confinement property of the language and release discipline: dependency edges transport already admitted values, but do not grant authority over other semantics co-located in the same observation carrier.

F.3 PROOF OF PURPOSE NONINTERFERENCE

Let L_S be the leaf ancestors of the declared sinks and suppose $O_1 \equiv_P O_2$. By definition,

$$\forall \ell \in L_S, \qquad C_\ell(O_1) = C_\ell(O_2).$$

We proceed by induction over a topological ordering of the sink-relevant subgraph. The base case holds for all leaves in L_S. For an internal node v, all predecessor completions are equal under the induction hypothesis. Because activation guards and operator interpretation are deterministic functions of those typed predecessors, $C_v(O_1) = C_v(O_2)$. The argument therefore reaches every declared sink, giving

$$\forall s \in S, \qquad C_s(O_1) = C_s(O_2).$$

The reconstruction contract and determinism then imply

$$\mathrm{FactSem}(A_{\mathrm{out}}(O_1)) = \mathrm{FactSem}(A_{\mathrm{out}}(O_2)),$$

Table 8: Operational boundary used for the compared methods.

Method	Evaluation boundary
No Guard	Uses the fixed workflow artifact without privacy filtering.
Presidio	Applies the official analyzer/anonymizer to artifact content.
IBM Contextual Privacy LLM	Uses the contextual-privacy component with its fixed model configuration.
Clean Task-text Guard	Uses Qwen3.5-0.8B with the natural-language task as direct filtering context, without Purpose IR authority.
Fides constrained inspection	Instantiates a Fides-based IFC/constrained-inspection path with Qwen3.5-0.8B semantic inspection.
AgentSpec manual privacy rule	Applies an independently specified AgentSpec-style static privacy-rule interpreter.
Purpose IR	Uses the task-only semantic frontend, deterministic Purpose compiler, typed evidence binding, topological execution, sink validation, and release reconstruction.

which proves Proposition 2.

The equality is semantic rather than byte-level: framework-specific control fields, serialization details, or non-factual provenance may differ without changing the factual reusable state. The property therefore captures the intended privacy boundary—changes to observation semantics that do not alter the purpose-relevant leaf results cannot alter released factual semantics.

G Evaluation Protocol

G.1 Main metrics and evaluator

We report Privacy, Task Success Rate, and Privacy-Compliant Success (PCS) per case:

$$\mathrm{PCS}(i) = \mathrm{Privacy}(i) \wedge \mathrm{TaskSuccess}(i).$$

Privacy checks whether downstream-visible reusable state avoids explicitly disallowed content and semantically unauthorized payloads. Task Success Rate checks whether all task-required results remain available in downstream-visible semantic content or target-bound typed outputs. PCS requires both conditions on the same execution. The same deterministic evaluator is applied post hoc to the fixed outputs of all methods, without output modification.

G.2 PrivacyLens external transfer

PrivacyLens contains 493 privacy-sensitive action trajectories (Shao et al., 2024). We use the benchmark's action-level leakage and helpfulness judges with a fixed local Mistral-7B-Instruct-v0.2 evaluation model. The Purpose IR frontend is Qwen3.5-0.8B, and the Purpose runtime does not receive benchmark sensitive labels or final-action labels. The downstream action model receives only the authorized sink artifact.

We report Privacy Pass, Helpfulness ≥ 2, Mean Helpfulness, and Safe Action Rate. Safe Action Rate is computed per trajectory as

$$\mathrm{SafeAction}(i) = \mathrm{PrivacyPass}(i) \wedge [\mathrm{Helpfulness}(i) \geq 2].$$

G.3 Baseline boundary

All methods operate at the same fixed workflow-artifact boundary used by Purpose IR. Appendix J records model roles and execution settings.

Table 9: Aggregate results over the 1,200-case main workflow-state benchmark.

Method	Privacy	Task Success Rate	PCS
No Guard	296 (24.7)	966 (80.5)	287 (23.9)
Presidio	312 (26.0)	852 (71.0)	249 (20.8)
IBM Contextual Privacy LLM	588 (49.0)	769 (64.1)	387 (32.3)
Clean Task-text Guard	1155 (96.2)	318 (26.5)	304 (25.3)
Fides constrained inspection	1189 (99.1)	671 (55.9)	670 (55.8)
AgentSpec manual privacy rule	1052 (87.7)	300 (25.0)	195 (16.2)
Purpose IR	**1195 (99.6)**	**1000 (83.3)**	**1000 (83.3)**

Table 10: Per-workflow Purpose IR results on the main 1,200-case benchmark. Counts are shown with percentages in parentheses. Valid IR reports successful static validation of the compiled Purpose program.

Workflow	Privacy	Task Success	PCS	Valid IR
LangGraph	398 (99.5)	342 (85.5)	342 (85.5)	400 (100.0)
CrewAI	398 (99.5)	319 (79.8)	319 (79.8)	400 (100.0)
MobileRun	399 (99.8)	339 (84.8)	339 (84.8)	400 (100.0)
Aggregate	**1195 (99.6)**	**1000 (83.3)**	**1000 (83.3)**	**1200 (100.0)**

H Additional Results

H.1 Main aggregate results

Purpose IR reaches 83.3% PCS with 99.6% Privacy and 83.3% Task Success Rate. Fides is the strongest high-privacy baseline at 99.1% Privacy and 55.8% PCS, giving Purpose IR a 27.5-point PCS advantage at a similar privacy level.

H.2 Runtime mechanism ablations

We isolate the contributions of recovery binding and sink-constrained reconstruction while preserving the compiled programs and common evaluator.

Removing recovery binding isolates the contribution of requirement-directed evidence recovery, while removing sink-constrained reconstruction tests whether restricting reusable state to validated sink outputs is necessary for privacy.

I Validation of IR Semantics

I.1 Static validation

Static validation enforces graph well-formedness, typed dependency constraints, raw-access restrictions, sink validity, and release ordering. Programs that violate these constraints are rejected before execution, supporting the structural assumptions used by the formal analysis.

I.2 Cardinality semantics

The main benchmark contains three cases in which a scalar requirement encounters multiple distinct typed candidates. Full Purpose IR enforces `cardinality=one` and withholds ambiguous scalar release. Removing the cardinality constraint changes the aggregate result to 1185/1200 Privacy, 1014/1200 Task Success Rate, and 1005/1200 PCS. The comparison captures the intended trade-off: releasing an ambiguous multi-value result can recover a task outcome while violating the declared scalar contract.

Table 11: Paired runtime ablations on the main benchmark. Counts are out of 1,200 cases; percentages are shown in parentheses.

Variant	Privacy	Task Success Rate	PCS
Full Purpose IR	1195 (99.6)	1000 (83.3)	1000 (83.3)
w/o recovery binding	1198 (99.8)	162 (13.5)	162 (13.5)
w/o sink-constrained reconstruction	419 (34.9)	854 (71.2)	290 (24.2)

Table 12: Frontend/compiler results on the controlled compositional suite.

Frontend / compilation path	Valid IR	Semantic contract	Privacy
Minimal full-IR, 0.8B	206/300	25/300 (8.3)	300/300
Balanced full-IR, 0.8B	234/300	139/300 (46.3)	300/300
Semantic proposal + lowering, 0.8B	293/300	194/300 (64.7)	300/300
Semantic proposal + lowering, 27B	298/300	**288/300 (96.0)**	300/300
Gold IR	300/300	300/300 (100.0)	300/300

I.3 Flat state schema

The main benchmark's canonical programs do not require `COMPOSE`, multi-leaf sinks, or graph depth of at least two. Replacing the graph representation with a flat state schema therefore produces the same aggregate Privacy, Task Success Rate, and PCS as Full Purpose IR on this distribution. We evaluate compositional language semantics separately below.

I.4 Controlled compositional suite

The suite contains 60 semantic scenarios, each instantiated with five paraphrases, yielding 300 task instances. It covers argument binding, record composition, conditional routing, explicit grouping, and multi-stage dependency. Semantic-contract scoring is invariant to node identifiers and serialization order.

Of the 300 instances, 270 use two evidence leaves with depth-one dependency structure, while 30 exercise grouping or two-stage dependencies with three leaves and depth two. The 0.8B semantic-proposal frontend satisfies 182/270 depth-one instances and 12/30 deeper instances; the 27B frontend reaches 266/270 and 22/30, respectively.

I.5 PrivacyLens external transfer

I.6 Representative semantic-boundary examples

J Implementation and Evaluation Details

J.1 Model roles

The neural frontend proposes task semantics before compilation. Binding, typed execution, sink validation, and release reconstruction are deterministic and do not use task text, benchmark labels, or case identifiers as independent authorization sources.

J.2 Implementation scope

The implemented system realizes the bounded Purpose IR described in this paper: task-only semantic proposal, deterministic graph compilation and validation, workflow-specific typed evidence adapters, topological interpretation, structured sinks, and reconstruction-based release. The language represents authorized semantic dataflow rather than general agent control flow; tool planning, loops, retries, and arbitrary execution strategy remain in the host agent.

Table 13: External transfer on 493 PrivacyLens trajectories. Safe Action requires both Privacy Pass and Helpfulness ≥ 2 on the same trajectory.

Method	Privacy Pass	Help. ≥ 2	Safe Action	Mean Help.
Full trajectory	308 (62.5)	487 (98.8)	304 (61.7)	2.375
Task-text reduction	365 (74.0)	455 (92.3)	330 (66.9)	2.191
Presidio	392 (79.5)	400 (81.1)	307 (62.3)	1.870
AgentSpec manual privacy rule	449 (91.1)	321 (65.1)	277 (56.2)	1.499
Fides constrained inspection	473 (95.9)	184 (37.3)	168 (34.1)	0.826
Purpose IR	427 (86.6)	444 (90.1)	**381 (77.3)**	2.081

Table 14: Representative observation-to-state reductions.

User task	Rich observation may contain	Authorized result
Check whether I have a meeting reminder.	Sender, time, location, phone number, full notification body.	`EXISTS(meeting_reminder)` $\rightarrow$ Boolean.
Has my package been delivered?	Retailer, delivery address, timestamp, notification prose.	Typed delivery-status result.
Retrieve the requested meeting link.	URL, phone number, access code, surrounding message text.	Unique validated `meeting_link` scalar.

Table 15: Model and deterministic component roles.

Role	Model or execution mode
Base-case upstream workflow generation	Fixed workflow artifacts generated by the 35B model
Controlled stress cases	Deterministic workflow-native state
Task semantic frontend	Qwen3.5-0.8B, task-only input
Purpose compilation	Deterministic lowering and graph validation
Evidence binding	Deterministic workflow adapters
Purpose interpreter	Deterministic
Main evaluator	Deterministic
PrivacyLens judge	Mistral-7B-Instruct-v0.2, evaluation only
Fides constrained inspection	Qwen3.5-0.8B semantic inspection in independent baseline
IBM baseline	`llama3.2:3b` in fixed contextual-privacy component