ChatGPT

Integrating a Semantic Knowledge Layer into Agent Control Workflows

Introduction

Autonomous AI agents often execute complex **control flow graphs** – sequences of tasks and decisions – to achieve their goals. In previous work, we discussed using control flow graphs to regulate agentic workloads by defining what actions can occur and in what sequence. However, a pure control flow perspective captures structure **without context**. It tells us *how* tasks flow, but not *why* they occur or whether their content and intent are benign or malicious. To address this gap, we propose adding a **semantic knowledge layer** on top of the control flow graph. This layer uses **semantic graphs** (knowledge graphs with ontologies and taxonomies) to imbue the workflow with rich context and meaning. By integrating semantic understanding, we can identify the intent behind actions, anticipate side effects, and pinpoint critical "centers of gravity" in the workflow that have outsized impact on outcomes. In short, combining control flow graphs with semantic knowledge graphs yields a more **context-aware, explainable, and safe** agent workflow.

Organizations deploying large language model (LLM) based agents have learned that **context and factual grounding are essential** for reliable performance. Early AI implementations often fell short due to hallucinations, lack of context, and inability to distinguish critical information from trivia [1]. A semantic knowledge layer provides this missing foundation by preserving structure, injecting domain context, and ensuring the agent's understanding aligns with reality [2]. In the following sections, we delve into why a semantic layer is needed and how it enhances control flows, particularly in distinguishing benign versus malicious behaviors and increasing the determinism and reliability of agentic systems.

The Limits of Control Flow Graphs Alone

A control flow graph gives a **structural view** of an agent's possible actions – it defines states, transitions, and allowed sequences. This is invaluable for **constraining what can happen** in an agent's execution. For example, it might enforce that a data retrieval step must occur before a data analysis step, or prevent an agent from reaching a certain decision without specific prior checks. However, what the control flow graph **does not capture is context or intent**. Two identical sequences of actions might be benign in one context and malicious in another. Consider a sequence where an agent reads a database and then writes to a file – harmless for a data processing task, but potentially malicious if the read data was confidential and the write is exfiltrating it. The control flow alone can't tell the difference.

What's missing is a **semantic understanding of the actions and data**. The control flow graph knows *that* Action A leads to Action B, but not the meaning of A and B. This is why we introduce a semantic knowledge graph that represents the concepts, entities, and relationships involved in the workflow. By mapping actions and data to an ontology, we give the system knowledge of **what those actions represent, what data is being handled, and the intent or goal behind them**. For instance, the semantic layer could know that *"write to file X"* is intended for logging (benign) versus *"write to external file Y"* contains sensitive data (potentially suspicious), based on an ontology of data sensitivity and permitted operations.

In essence, **control flow graphs provide the syntax of the agent's behavior, while semantic graphs provide the semantics**. An analogy can be drawn to input validation in security: syntactic validation checks format and structure, whereas semantic validation checks meaning and appropriateness [3]. Likewise, the control flow is like syntax (structure of tasks) and the semantic layer checks whether the sequence **makes sense in context**. Without semantics, we lack the ability to flag when a perfectly valid sequence of actions is being used with malicious intent or when an "allowed" action has unsafe consequences.

Why Context Matters: Differentiating Benign vs. Malicious Behavior

In real-world workflows, the line between a power user's heavy-but-legitimate activity and a rogue agent's malicious activity can be **paper-thin**. The classic "paperclip maximizer" thought experiment illustrates how an AI pursuing an initially legitimate goal can veer into harmful behavior if context and intent aren't accounted for. In more grounded terms, an AI agent might aggressively optimize a task in a way that is technically allowed by its rules but ultimately undesirable or unsafe – for example, an LLM-based assistant might find a creative but unapproved method to solve a problem, or escalate its privileges because it *thinks* it needs to, even if no explicit rule forbade that sequence.

A semantic knowledge layer helps distinguish these cases by embedding **ontologies of benign and malicious patterns** into the agent's reasoning. We can formalize what constitutes normal vs. abnormal behavior. For instance, an ontology for user operations might classify "export data" actions and link them to concepts of "data sharing" or "exfiltration." If an agent's actions start to align with a known pattern of data exfiltration (e.g., reading large amounts of sensitive data then sending network requests), the semantic graph would recognize this pattern and flag it, even if each step alone was permitted. Conversely, if a power user is doing something intensive but within a known **"data analysis" pattern**, the semantic context would show it as benign usage of resources.

Recent research in multi-agent systems supports this need for context. In complex AI systems, no individual action may appear malicious on its own, yet **systemic failures can arise from a combination of benign-looking steps**. Coordination-based threats involve subtle issues like feedback loops or collusion between agents, where *"no individual agent may appear malicious; rather, systemic failure arises from cascading misalignments"* [4]. Only by looking at the semantic relationships between actions can we identify the emergent risk. A control flow graph alone might not reveal that a series of messages between agents is trending toward an unsafe consensus, whereas a semantic analysis could identify that the content of those messages matches a scenario of **collusive behavior** or misuse.

In summary, context is king. By modeling the agent's knowledge and intent in a graph, we equip the system to **understand what the agent is trying to do**, not just what steps it's taking. This is crucial for catching the thin line where a well-intended task turns into an out-of-control exploit. The semantic layer essentially asks, *"Do these actions make sense together given what we know about the domain and the agent's goals?"* – a question that must be answered to confidently separate acceptable behavior from malicious behavior.

Unpredictable Failure Modes of LLM Agents

Large Language Models can be incredibly capable in following instructions and carrying out tasks, but they also have well-documented **failure modes that differ from human errors**. A human expert or even a traditional program usually fails in relatively predictable ways or within understood error

margins. LLM-based agents, by contrast, might perform flawlessly on a task for a while, then suddenly produce a completely off-base or nonsensical action due to a subtle shift in prompt or state. These systems can **"hallucinate" outputs, misinterpret instructions, or take actions that violate constraints** in ways that a human never would [5]. Such failures are often non-deterministic – if you reset and run the same task again, the agent might not repeat the mistake, or might make a different mistake.

This unpredictability stems from the underlying probabilistic nature of LLMs. They don't truly understand the tasks; they pattern-match and improvise based on training data. As a result, they might succeed on complex reasoning one moment and fail on a simpler related problem the next, especially if adversarial or unexpected inputs are involved. For example, an LLM agent might correctly use a software tool via an API call in one instance, but later, given a slightly tweaked context, construct an invalid or harmful API call. We have seen situations where an LLM-powered agent works **amazingly well until it "wobbles"** – exhibiting minor errors – and then suddenly behaves in a completely **unintuitive or unsafe manner**. This kind of abrupt failure is hard to catch with testing alone because it's context-dependent and probabilistic.

By **measuring and enforcing determinism** in agentic solutions, we aim to mitigate these issues. This is where the semantic knowledge layer again plays a vital role. The semantic layer can act as a **safety net for LLM behavior** by continuously checking that each action and decision still aligns with known valid patterns and constraints. If the LLM starts veering off into an odd or forbidden direction, the semantic rules can catch it. For instance, if an LLM suddenly produces an output that doesn't match the expected schema or ontology (perhaps a hallucinated tool invocation or an illogical intermediate result), the semantic layer can flag this as a deviation. In a sense, the semantic graph is monitoring the *meaning* of the agent's actions in real time, providing an additional check beyond what the control flow allows.

An analogy can be made to runtime monitoring in multi-agent systems: researchers have proposed **runtime monitors that build graphs of agent interactions and detect anomalies via semantic reasoning** [6]. One such approach models agents and tools as nodes and their communications or actions as edges, enabling reasoning about failures at multiple levels [7]. The monitor (e.g., *SentinelAgent*) can spot when an agent's behavior becomes inconsistent with normal semantics – for example, if a typically data-reading agent suddenly tries to execute deletion commands, or if two agents begin exchanging messages that fit a pattern of a known exploit chain [8] [9]. By **bridging high-level anomaly semantics with actionable oversight**, the system can intervene before a minor LLM quirk escalates into a serious issue [6].

In summary, LLM-driven agents require an extra layer of semantic oversight because of their unpredictable failure modes. The semantic knowledge layer provides a form of **semantic determinism** – not making the agent fully deterministic (which might be impossible with current LLMs), but constraining the agent's operations within the bounds of understood, meaningful actions. This greatly reduces the chance of bizarre out-of-left-field behaviors going unnoticed or causing harm.

The Role of Ontologies and Taxonomies in Semantic Graphs

At the heart of the semantic knowledge layer is the construction of **ontologies and taxonomies** that describe the domain of the agent's operation. An ontology is a formal representation of concepts (entities, actions, conditions) and the relationships between them, while a taxonomy arranges those concepts into hierarchical categories. Together, they form a **knowledge graph** that the agent (and the overseers of the agent) can use to interpret the meaning of each step in a workflow.

Why are ontologies and taxonomies so important? Because they encode **contextual rules and expectations**. When we design an ontology for our agent's tasks, we are effectively answering questions like:

- *What are the valid types of actions and data involved?* (e.g., "QueryDatabase" and "SendEmail" might be action types; "UserPersonalData" and "AggregateReport" might be data types.)
- *What relationships exist between actions and data?* (e.g., "QueryDatabase" action **produces** a "Dataset" output; a "SendEmail" action **should not include** raw "UserPersonalData" as content without anonymization.)
- *What are the allowed sequences or combinations of concepts?* (e.g., If action = "DeleteRecord", then context must confirm user permission level is "Admin"; or "ScheduleTask" action must be followed by "ConfirmSchedule" in certain workflows.)
- *What is the classification of different behaviors?* (e.g., define what constitutes a "Data Exfiltration" pattern versus a "Data Backup" pattern, based on counts of records accessed, external endpoints contacted, etc.)

By capturing this knowledge explicitly, the semantic graph can assign **meaning and intent** to the raw steps in the control flow. Each node and edge in the control flow graph can be annotated with semantic labels from the ontology. This enables advanced analysis: we can ask the system to highlight all parts of the flow dealing with sensitive data, or identify if an agent action is an *"attempt to circumvent policy"* versus a normal operation, based on the presence of certain semantic markers.

In practical terms, building such an ontology might involve merging several existing ontologies and taxonomies (for example, a security ontology for defining malicious behaviors, a process ontology for defining standard operating procedures, etc.) and then adding custom relationships relevant to our specific agent. The result is a **rich schema of the agent's world**.

Notably, this semantic schema goes far beyond a typical data schema. A standard data schema (or JSON schema) can enforce that a field is an integer or that an object has certain sub-fields, but it doesn't understand context. It might ensure a field "user_id" is numeric, but it can't express "user_id must belong to a currently active user" without additional logic. By contrast, an ontology can encode that *"user_id" is associated with a User entity that has a status property*, and a rule can state *active status = required* for certain operations. This is the difference between **syntactic checks and semantic checks**. As OWASP notes in the context of security, *"semantic validation should enforce correctness of values in a specific business context (e.g. start date is before end date, price is within expected range)"* [3]. In our case, the ontology defines the "business context" of the AI agent's operation.

Additionally, the ontology/taxonomy approach allows us to **rate or score behaviors**. We can tag certain actions or sequences with a risk level. For example, in the ontology, "UseAdminAPI" could be tagged as high-risk and require a certain justification concept to be present (like a "UserApproval" node). If the agent attempts a high-risk action without the proper context nodes in place, this deviation is immediately detectable via the semantic layer.

In summary, ontologies and taxonomies are the **knowledge base** that powers the semantic graph. They turn a vague question like "Is this action okay?" into a precise check against a body of encoded knowledge. They allow the system to know that *two occurrences* of a certain action are acceptable but *three is not*, or that *Action X followed by Y* is suspicious whereas *X followed by Z* is normal. This kind of nuanced understanding is exactly what's needed to oversee powerful AI agents.

Two Levels of Semantic Integration in Workflows

When we talk about adding a semantic knowledge layer, it actually operates at **two distinct levels** in the agent workflow architecture:

1. Semantic Workflow Graph (Contextual Oversight Layer)

The first level is a **semantic overlay on the workflow itself**. Here we take the entire control flow graph of tasks and enrich it with a parallel semantic graph that represents the meaning of each step and the relationships between steps in terms of goals, effects, and requirements. This can be visualized as an annotation or augmentation of the control flow: every node (task) and edge (transition) in the control flow links to corresponding nodes and relationships in the semantic knowledge graph.

In this layer, we perform **understanding, visualization, and analysis** of the workflow. By examining the semantic graph, we can answer high-level questions: *What is the agent trying to achieve through this series of actions? Is there any point in the flow where a policy is being violated or a risky action is being taken?* Because the semantic graph encodes intent and effect, analyzing it can reveal patterns like a chain of actions leading to an undesirable outcome (e.g., reading personal data → converting to an unrestricted format → sending externally might form a pattern "data leak"). This is akin to how the SentinelAgent research built an *execution interaction graph* to detect *"high-risk execution trajectories or attack chains"* by tracing paths through the graph that indicate multi-step failures [8].

The semantic workflow graph also greatly aids **explainability**. If something goes wrong or an alert is raised, we can trace through the semantic annotations to explain why. For example, an alert might say: *"Task 5 ('Export Data') is flagged because it operates on a 'ConfidentialDataset' and is followed by a network call outside the trusted network, matching a potential exfiltration pattern."* This kind of explanation is only possible because the semantic knowledge layer knows what "ConfidentialDataset" means and understands the significance of an external network call. In a purely control flow view, we would just see function names or API calls without context.

Visualization tools can leverage the semantic layer to provide a **higher-level map of the workflow**. Instead of a tangle of technical steps, stakeholders could see an abstracted graph: e.g., *User Query → Data Gathering (safe) → Data Analysis (safe) → Data Export (warning: involves sensitive data)*. This helps developers and auditors to reason about the agent's design and to refine the ontology further if needed.

In summary, the semantic workflow graph layer sits **above the control flow**, observing and interpreting it. It provides situational awareness of the agent's behavior, much like a supervisory mind that watches the agent's actions and compares them against a body of knowledge about what those actions mean.

2. Semantic Schema Enforcement (Structured Handover Layer)

The second level of integration is at the **boundary between tasks** – essentially in the inputs and outputs that are passed along the workflow. The idea here is to enforce that the data exchanged between tasks (and between the agent and external tools or users) adheres not just to a structural schema, but to a **semantic schema** defined by the knowledge graph.

In practice, this means when one task hands off data to the next, the data should be packaged in a form that is validated against the ontology. Instead of free-form text or loosely structured JSON, the

intermediate data could be an instance of an ontology class or a reference to a knowledge graph entity. By doing this, we ensure **consistency and correctness at each step**. Each handover is an opportunity to catch errors or malicious injections early: if a task output doesn't match the expected semantic type or violates a constraint, it can be rejected or adjusted before it propagates further.

Consider how many attacks or failures occur due to **injections or misunderstandings at handover points**. For example, a prompt injection attack might involve sneaking a malicious command in text that gets passed from one agent to another. If the receiving agent expects a certain semantic structure (say, a summary of a document that should contain facts X, Y, Z and nothing else), an injection that introduces an out-of-place instruction would violate the semantic schema and be caught. Traditional validation might only check for syntax (like allowed characters) and might miss the fact that the content, while syntactically valid, has a malicious directive. By enforcing semantic rules, we add a robust layer of defense. Indeed, security experts emphasize using both syntactic and semantic validation to reject inputs that are structurally correct but contextually inappropriate [3] [10]. In an AI agent context, the "inputs" are often prompts or data between components, and semantic validation becomes crucial to prevent things like prompt injections, tool misuse, or logic corruptions.

Furthermore, semantic schema enforcement improves **reliability and determinism**. If every intermediate result must conform to a known type or format (as defined in the ontology), we reduce the degrees of freedom for the agent's output at each step. The agent cannot easily drift into an undefined state because any off-schema output will be flagged. This means the overall process becomes more deterministic in its progression – there's less chance of a weird, off-schema result causing a cascade of errors. The flow has guardrails at each juncture.

For example, suppose an agent has a step to produce a plan (a common pattern in agentic workflows). Instead of allowing a free-form text plan, we enforce that the plan must be an instance of a "Plan" object defined in our ontology, which might require a goal statement, a list of actions, and each action must be one of the known action types with valid parameters. This schema can require, say, that if an action is "DeleteFile," it must have a justification attached or a pre-approval token in the data. If the LLM agent tries to output a plan where an action is "DeleteFile" without that justification field, the semantic validator will reject it as invalid output. Thus the LLM is **forced to produce plans that make sense** in context (or else the system will intervene). Over time, this can even feed back into training or prompt engineering: the model learns that nonsense or context-violating outputs won't succeed, nudging it toward more reliable behavior.

In sum, this second layer ensures that **every handshake between tasks is semantically sound**. It treats the knowledge graph as a contract that each step's output must satisfy. By doing so, it significantly boosts the explainability (every piece of data is labeled and structured) and reduces the risk of things like injection attacks or silent misunderstandings propagating through the system.

Enhancing Determinism and Reliability through Semantic Control

One of the ultimate goals of adding a semantic knowledge layer is to **increase the determinism, predictability, and reliability** of agentic AI systems. As noted, LLM-based agents can be stochastic and their error boundaries are not always clear. By instrumenting the agent's workflow with semantic constraints and context, we effectively tighten those error bounds.

We can draw an analogy to **Wardley Maps** and their concept of evolution stages (Genesis → Custom-Built → Product → Commodity). In early "Genesis" stages of an AI solution, some unpredictability and

failure is expected – you are experimenting with new capabilities. In "Custom-Built" stage, it's still bespoke and perhaps tolerated that it occasionally fails in odd ways. But by the time you reach a "Product" stage deployed to users, you expect a degree of repeatability and robustness, and in a "Commodity" stage (ubiquitous utility), you demand extremely high reliability (components become *"utility-like reliable"* and standardized [11]). The semantic layer is a mechanism to **push an AI agent system further along that evolution in terms of reliability**.

With semantic controls, our AI agent can exhibit more **product-like or utility-like consistency** even if internally it's powered by a non-deterministic LLM. We achieve this by standardizing how it understands and communicates information (through the ontology) and by strictly governing its actions (through semantic rules and schema checks). Essentially, we *commoditize* the knowledge and behavior patterns – they become well-defined components in the knowledge graph that are reused and consistently applied.

For example, if our ontology defines a "SafeMode" behavior for certain risky operations, and we always require that concept to be present when performing those operations, the agent's behavior in those scenarios becomes far more predictable (it always has to engage SafeMode, or else it can't proceed). This is analogous to standardizing a procedure: no matter which LLM instance or prompt variation, the semantic requirement ensures certain steps are always done a certain way.

By continuously **mapping and rating behaviors** via the semantic graph, we can also quantitatively track reliability. We might measure how often the agent's actions stay within low-risk categories vs. how often it attempts something in a higher-risk category that is then blocked or corrected. This provides feedback to developers about where the model might need more training or prompt adjustments. Over time, as the system matures, we can tighten the semantic rules (shrinking what is considered acceptable variation) to align with the expected reliability at that stage of evolution. In the early stage we might allow more leeway and just log semantic deviations; later on we might treat any semantic deviation as an error.

To put it another way, the semantic knowledge layer helps turn the agent's free-form reasoning into a **controlled, almost typed program**. Each part of the agent's process has a type (from the ontology) and must behave according to the contracts of that type. This doesn't remove the creativity or flexibility of the LLM, but it bounds it with guardrails. The outcome is an agent that's much more **trustworthy** – a term echoed by industry approaches that combine knowledge graphs with AI to ensure *"reliable results"* and proper understanding of context [1].

Crucially, this added determinism does not come at the cost of capability; rather it channels capability in the right direction. It is similar to how a compiler's type system prevents certain classes of bugs at compile-time – here the ontology prevents certain classes of mistakes or misbehaviors at "plan-time" or "run-time" for the AI.

Explainability and Observability Benefits

Another major benefit of layering semantics into agent control flows is the boost to **explainability** and **observability** of the system. Modern AI systems are often criticized as "black boxes," especially LLM-based agents that make decisions in opaque ways. By introducing a semantic knowledge graph, we force the agent (and ourselves) to work with **explicit concepts and relationships** that are understandable.

For example, if someone asks “Why did the agent refuse to perform action X?”, without a semantic layer we might only have a technical answer like “the policy engine disallowed it.” With the semantic layer, we can answer: “Action X was deemed to have a *malicious intent* because it matched the ontology pattern of a known attack (it would have resulted in an unauthorized data disclosure). The agent’s knowledge graph identified the request as violating the rule *DataPrivacyRule123* which states personal data cannot be sent to unknown external recipients.” This kind of answer is not only satisfying to a human overseer but is also actionable – it points to the exact rule or concept that was triggered. We could even visualize the relevant portion of the graph: showing the nodes and edges that constituted the offending pattern.

From an **audit and compliance** perspective, semantic graphs are a goldmine. They log not just what happened, but what it *meant*. If an incident occurs, investigators can replay the semantic trace of the agent’s actions. Perhaps the agent made a decision that was semantically allowed but turned out poorly – the trace might reveal that our ontology lacked a rule in that case, suggesting we update our taxonomy of risks. On the other hand, if the semantic layer successfully caught an issue, the trace provides evidence that proper controls are in place.

Observability is also improved because we can monitor high-level metrics derived from the semantic understanding. Instead of just CPU usage or API call counts, we can monitor things like “number of high-risk actions attempted per hour” or “taxonomy category of user requests over time (how many requests fell into ‘unusual’ category)”. These are possible because each action is labeled and contextualized. In complex multi-agent environments, such semantic logging helps pinpoint where in the chain a misunderstanding occurred. For instance, if two agents disagreed or a task failed, the semantic log might show that Agent A passed an object of type “Report” but Agent B interpreted it as type “Command” due to a missing context link, leading to a wrong action. That insight can then inform a fix (maybe the ontology needs to clarify that this kind of data should not be interpreted as a command).

It’s worth noting that academic work on explainable AI for multi-agent systems often leverages structured representations. The graph-based SentinelAgent monitor we discussed provides **interpretable diagnostics and execution traces** by virtue of its graph model [12] [13]. When an anomaly is detected, it’s not just an alert – the system can highlight the path in the interaction graph that led to it, which corresponds to a human-comprehensible sequence of events. Our approach with semantic layering is very much aligned with this philosophy: make the agent’s decision process legible by mapping it onto known concepts.

In conclusion, by adding a semantic knowledge layer, we transform an AI agent workflow from a opaque series of steps into a **transparent, well-labeled, and monitorable process**. This not only increases trust with users and stakeholders but also accelerates development (since debugging and refining the agent becomes easier when you can see what it *thinks* it’s doing at a conceptual level).

Conclusion

Integrating a semantic knowledge layer with control flow graphs creates a powerful synergy for managing agentic AI workloads. The control flow graph provides the **structure** – it tells us what could happen. The semantic knowledge graph provides the **meaning** – it tells us what those happenings signify. Together, they enable a level of control and insight that neither alone could achieve.

By introducing semantic ontologies and taxonomies, we gain the ability to differentiate between superficially similar behaviors, labeling one as benign and another as malicious based on context and intent. We can catch the subtle transitions where a well-meaning agent’s behavior starts to drift into unsafe territory, because the semantic patterns will reveal the shift. We also guard against the

unpredictable nature of LLMs by enforcing semantic consistency at every step, thereby reigning in the randomness and preventing nonsensical or harmful outputs from slipping through. This approach moves us closer to deterministic, reliable AI agents that can be trusted in production settings.

Moreover, the semantic layer brings significant improvements in explainability and governance. Every action and decision can be explained in terms of the agent's ontology – why it was allowed, or why it was flagged. We essentially document the agent's **implicit knowledge and intentions explicitly** in the knowledge graph. This not only helps in current operations (monitoring and safety) but also provides a blueprint for future agent designs, and a basis for continuous improvement of the AI system.

In practice, implementing this vision would involve building out the domain ontologies, instrumenting the agent platform to validate actions against the knowledge graph, and creating monitoring tools that visualize the semantic graph in tandem with the control flow. It's a non-trivial effort, but the payoff is an **AI agent that behaves more like a well-governed, context-aware system and less like an unpredictable black box**. As AI agents take on increasingly sensitive and critical tasks, such semantic knowledge layers may well become a standard component to ensure **integrity, safety, and trustworthiness** of autonomous systems.

Sources:

- Andreas Blumauer, *"Why Semantic Layers and GraphRAG Are Essential for Trustworthy AI,"* Graphwise (Oct 2025) – discusses how adding semantic layers and knowledge graphs provides context and structure to AI, helping ensure factual accuracy and reliable results [1] [2].

- OWASP Foundation, *"Input Validation Cheat Sheet,"* – emphasizes the need for semantic validation (contextual correctness) in addition to syntactic checks [3], underscoring the importance of context-aware rules to prevent misuse.

- Joseph Oladokun et al., *"SentinelAgent: Graph-based Anomaly Detection in LLM-based Multi-Agent Systems,"* arXiv 2505.24201 (2025) – proposes modeling multi-agent executions as graphs capturing both structure and semantics, enabling detection of misuses (e.g., prompt injection, tool abuse) and highlighting that many failures stem from **semantic misinterpretations** rather than explicit rule violations [9]. Describes how graph-based oversight can catch covert anomalies and provide interpretable traces of agent behavior [8] [6].

- Wardley Maps (Evolution Stages) – defines how technologies evolve from genesis to commodity, with increasing reliability and standardization [11]. This concept illustrates why our approach must tighten agent behavior as it matures into a utility-like service.

---

1 2 Why Semantic Layers and GraphRAG Are Essential for Trustworthy AI
https://graphwise.ai/blog/why-semantic-layers-and-graphrag-are-essential-for-trustworthy-ai/

3 Input Validation - OWASP Cheat Sheet Series
https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html

4 5 6 7 8 9 12 13 SentinelAgent: Graph-based Anomaly Detection in LLM-based Multi-Agent Systems
https://arxiv.org/html/2505.24201v1

10 Input validation errors: The root of all evil in web application security
https://www.invicti.com/blog/web-security/input-validation-errors-root-of-all-evil

11 Evolution Stages - Strategic Terms | Wardley Maps
https://www.wardleymaps.com/glossary/evolution-stages