# AIOracles: Modeling Agentic AI Security*

F. Betül Durak
*Microsoft Research*
betul.durak@microsoft.com

Tadayoshi Kohno
*Georgetown University*
yoshi.kohno@georgetown.edu

Franziska Roesner
*University of Washington*
franzi@cs.washington.edu

***Abstract*—Today, we increasingly deploy agentic AI systems with significant autonomy, yet our confidence in their security rests largely on benchmarks and qualitative claims, neither of which captures behavior under adaptive adversaries. Cryptography faced a similar problem and developed formal definitions, adversarial models, and compositional reasoning that enabled explicit, falsifiable security claims. Inspired by this evolution, we introduce AIOracles as an abstraction for the security of LLM and agentic AI systems. The aim is to make security claims and their assumptions explicit and comparable across designs.**

**We develop a formal framework for security in agentic AI. It is built around an ideal predicate $\phi$ that decides, for a given $(\mathsf{prompt}, \mathsf{context}, \mathsf{result})$ triple, whether the agent's behaviour is acceptable. Concretely, $\phi$ checks three things: that the agent's actions are permitted by policy and aligned with the user's intent; that the data it relies on and the result it produces are sound, correct, and useful; and that the information flows respect the system's trust boundaries. Our framework parameterizes the adversary by capabilities $\mathsf{ATK} \subseteq \{\mathsf{white\_box}, \mathsf{black\_box}, \mathsf{inject\_learn}, \mathsf{inject\_infer}\}$ and defines completeness and security as game-based properties. Within this setting, we prove four formal results: (i) a confidentiality-utility tradeoff showing that any useful generic classifier must leak a quantifiable amount of training-data membership information; (ii) a *dual construction* that composes a creative AIOracle with a filtering AIOracle and inherits completeness and security from its components; (iii) a decomposition of $\phi$ into seven sub-predicates that enables modular security reasoning for agents and makes the responsibility of each defensive mechanism explicit; and (iv) a reduction of multi-step agent security to single-call security. AIOracles are intended as a starting point: a way to state what an agent is supposed to do, what an adversary can do, and how the resulting bounds compose.**

## 1. Introduction

Recent advances in Artificial Intelligence (AI), including the rapid maturation of (Large) Language Models (LMs), have enabled deployed systems to take actions on a user's behalf. LM-based *agents*, which autonomously plan, call tools, and act on a given instruction, have quickly became a dominant computing paradigm. As these systems increasingly act with a high degree of autonomy and authority, the security community has naturally turned its attention to studying their security, for example in relation to the access control permissions granted to such agents [2], [3], [4], [5]. Today, however, our confidence in the security of these systems rests largely on benchmarks and qualitative claims. Cryptography was at a similar state decades ago. In the early days of modern cryptography, algorithms were proposed based on heuristic arguments and were considered secure until a new attack was discovered that they could not withstand. At the time, there was no rigorous method to precisely measure the security. The difficulty was twofold: to formally define what it means to "break" a system and to formally prove the hardness of breaking a system. Thanks to the solid foundation from complexity theory, cryptography now has formal definitions and reduction-based proofs. Similarly, we argue that cryptography-inspired formal definitions for LMs will enable a subfield of security research to rigorously reason about the security of LMs.

**On the value of cryptography-inspired definitions.** In the field of modern cryptography, there is still a significant gap between what can be formally analyzed and proven and real systems. Still, it is indisputable that advancements in the theoretical foundations of modern cryptography have led to an overall advancement in cryptographic security. Thus, while clearly of scholarly value, there are limits on the utility of formal definitions in cryptography. Formal definitions of security for LMs are useful within bounds. As a lower bound, ours can server as a reference point for cryptographers assisting with LM design and evaluation, and as a foundation for studying what is achievable and what is not. As an upper bound, we recognize that LM designers are unlikely to use our formalism directly, even when they are aware of it.

**On cryptography-inspired definitions.** Cryptographic primitives are usually defined by algorithms (how they run), a notion of correctness, and a notion of security. The *correctness* requirement characterizes the expected behavior of these algorithms under honest execution in the absence of adversarial interference. In contrast, the *security* requirement formulates the unexpected behavior even when the algorithms are executed in the presence of an adversary employ-

* An earlier version of this work with surveyed literature can be found at [1].

ing arbitrary but computationally bounded strategies. Cryptography further defines security by a game that an adversary "plays" with various capabilities such as access to some information or oracles in a black-box manner. The advantage of an adversary is quantified to say that a system is secure if every efficient adversary has a negligible advantage. Then, the security of a cryptographic system is formally reduced to a well-studied computational problem. This methodology has a long history in cryptography and forms the foundation of provable security. In that spirit, we develop a formal framework for studying security of LM-based agentic systems.

The main challenge of this work is to capture the expected behavior of AI. Essentially, an AI aims at replicating an intelligent human behavior in an algorithmic way. Assessing whether an AI fulfills this goal (in an algorithmic way) implies that we have succeeded in mimicking human assessments. This is a circular problem. Hence, we abstract such an assessment by an ideal predicate (sometimes called an oracle) and have correctness and security be defined relative to it. Such an ideal abstraction was also used in a different manner by [6] and others.

**Our contributions.** We introduce the AIOracle abstraction as a unifying formal description for AI systems (section 2), defined by a learning phase, an inference phase, and a predicate $\phi$ that captures both helpfulness and harmlessness objectives. In this work, we cleanly focus on building the formal language with detailed proofs and show how our framework can be used in designing and analyzing secure agentic systems. Our key contributions are:

1) We define AIOracles with completeness and security notions parameterized by the predicate $\phi$, which can be instantiated for specific applications (section 3).
2) Using our framework, we showcase the tradeoff between training-data confidentiality and utility with our formalism for a classifier with respect to a strong confidentiality game. More specifically, we quantify this well-known tradeoff for a generic AIOracle classifier (Theorem 1 in subsection 3.2).
3) We introduce the *dual construction*, which composes a creative AIOracle (CAIO) and a filtering AIOracle (BAIO), and prove its soundness: if each component is complete and secure for its respective sub-predicate, the composition is complete and secure for the conjunction (Theorem 2 in section 4).
4) We decompose the predicate $\phi$ into seven sub-predicates in three groups (action, data, flow) that enable modular security reasoning for agents, and show how multi-step agent execution reduces to single-call security (Theorem 3 in section 5).

We emphasize that the goal of this work is not to prescribe a definitive security definition for all AI systems. Our aim is to establish a methodology in which security claims for agentic AI become precise enough to be falsifiable: a system designer can instantiate $\phi$ with concrete, application-specific requirements, formulate a claim, and either prove it via reduction or refute it by exhibiting an adversarial strategy. Concrete steps a designer follows to instantiate the framework are summarized as a recipe in section 6.

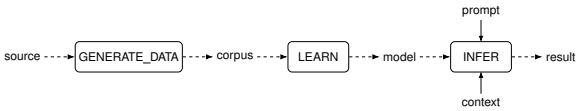


Figure 1: Data and algorithm pipeline in AIOracle.

**Notation.** Table 3 summarizes the notation used throughout the paper.

## 2. Language Models as AIOracles

In this section, we use the AIOracle abstraction as a unifying model that captures a broad class of AI systems, including LMs, chatbots, classifiers, and agents. A chatbot is based on LMs with iterated inferences. Instead of only outputting the next token, a chatbot makes complete sentences. A classifier is very similar to an LM except that the output of the inference algorithm is in a much smaller set of categories. Finally, an agent can be defined as "the result of inference is code to be executed in interaction with tools." [1] Given these notions, we arrive at the notion of an "AIOracle" (based on commonality), which we will further clarify and detail later in section 3.

### 2.1. The AIOracle Abstraction

A Language Model (LM) is a set of algorithms that work in two phases: learning and inference. The learning phase is a multi-round process including training, fine-tuning and reinforcement learning. The training algorithm inputs some data called a corpus and outputs the first model. Then the fine-tuning process takes this model along with some labeled data to output an updated model. Finally, the reinforcement learning phase further trains the model with some ranked outputs to output the final model. On the other hand, there is only one algorithm running in the inference phase, which takes a query from a user and uses the model to output a mapping from the dictionary (or vocabulary) to a probability. The inference phase can be run iteratively many times. Overall, given the token (a word, part of a word, or punctuation) probabilities, LM algorithms select the next token in the sequence based on this distribution. Moving ahead, we aggregate all rounds of the learning phase into a single LEARN algorithm and all iterations of inference into a single INFER algorithm as shown in Figure 1.

1. We will give a concrete example in section 5.

| Notation | Meaning |
|---|---|
| *AIOracle framework (section 2, section 3)* | |
| AIOracle | Abstraction of an AI system specified by data generation, learning, and inference algorithms together with a predicate. |
| GENERATE_DATA | Data-generation procedure used to build the training corpus. |
| source | Underlying data distribution / environment from which GENERATE_DATA samples. |
| LEARN, INFER | Learning and inference algorithms of an AIOracle. |
| corpus | Training input used by LEARN. |
| model | Model produced by LEARN and consumed by INFER. |
| prompt, context, result | Input, surrounding context, and output of an inference call. |
| empty | Empty/null value (e.g. unavailable view, oracle, or context). |
| *Predicates (section 3, section 5)* | |
| $\phi$ | Objective predicate capturing helpfulness and harmlessness; defines completeness and security of an AIOracle. |
| $\phi'$ | Decisional companion of $\phi$: $\phi(\text{prompt}, \text{context}, \text{result}) \Leftrightarrow \phi'((\text{prompt}, \text{result}), \text{context}, \text{accept})$. |
| $\phi_{\text{subpred}}$ | A sub-predicate from a decomposition $\phi = \bigwedge_i \phi_i$, with $\phi_{\text{allow}}, \phi_{\text{intent}}, \phi_{\text{content}}, \phi_{\text{correct}}, \phi_{\text{useful}}, \phi_{\text{inflow}}, \phi_{\text{outflow}}$ (action / data / flow groups). |
| *Security game (subsection 3.3)* | |
| $\mathcal{A}$ | Adversary $\mathcal{A}$ in the security/completeness games. |
| white_box, black_box | $\mathcal{A}$ sees the model (white-box) or only an inference oracle (black-box). |
| inject_learn, inject_infer | $\mathcal{A}$ corrupts the corpus during LEARN, or injects a chosen context during INFER. |
| ATK | Set of adversarial capabilities, drawn from {inject_learn, inject_infer, black_box, white_box}. |
| Security_Game | Security game between a challenger and an adversary, parameterized by ATK and source. |
| state | Adversary's persistent state passed between phases of the game. |
| view | Adversary's view in the inference phase (model if white_box $\in$ ATK, else empty). |
| b, $b'$ | Challenge coin sampled by the challenger and the adversary's guess. |
| $\psi$ | Predicate on the game trace that identifies trivial attacks; when $\psi(\text{trace}) = 1$ the adversary's win is not counted. |
| Adv | Advantage of $\mathcal{A}$ in winning a security game. |
| $\epsilon$ | Upper bound on the adversary's advantage; an AIOracle is $\epsilon$-ATK-$\psi$-secure for $\phi$. |
| *Dual construction (section 4, Theorem 2)* | |
| BAIO | Filtering AIOracle: accepts/rejects candidates with respect to $\phi'$. |
| CAIO | Creative AIOracle: proposes candidate results for BAIO. |
| e | Probability that CAIO's result fails $\phi_1$ on benign inputs (dual-construction slack). |
| *Agentic systems (section 5)* | |
| $n$ | Number of steps (inference calls) in the multi-step agent security game. |
| Proceed | Structured output format returned by INFER at step $i$, carrying the next $\text{tool\_action}_i$ and updated context. |
| $\text{tool\_action}_i$, $\text{tool\_out}_i$ | Tool call issued at step $i$ and the (adversarial) response returned to the agent. |

TABLE 1: Summary of notation used throughout the paper, grouped by topic.

## 2.2. AIOracle Helpfulness and Harmlessness Constraints

An AIOracle is designed with two specific objectives: to be *helpful* and to be *harmless* to the user, to society, or even to the provider. *These objectives are formalized by an ideal predicate $\phi$ that will be decomposed into seven sub-predicates to enable the model's security reasoning.* The first difficulty is that helpfulness requires two notions: the result of the query must be *correct*, and the result must be *useful*. If the user asks their very secure AIOracle "what is my next meeting?" and the result is "it is the one which follows your current meeting," it is a correct but useless answer to the question. In multi-purpose and complex systems like AIOracles, neither correctness nor usefulness can be algorithmically defined. Defining what is useful is specifically harder because we need an end user to judge, which might not work well in the real world.

**The predicate $\phi$.** The predicate $\phi$, defined with three inputs as (prompt, context, result) which are defined in Figure 1, is the formal object that encodes what it means for an AIOracle to behave as intended: it returns 1 when the result to the prompt in context is helpful and harmless, and 0 otherwise. Crucially, the framework takes $\phi$ as an idealized object. It assumes that we can *name* what we want and reason about approximations to it. We start with this because specification is the part of agentic AI security that is most often left implicit, and an implicit specification cannot be reasoned about formally. Indeed, [5] defines our predicate $\phi$ as a set $\Omega_{\text{prompt}}$ and [7] defines it as a ground truth function $F^*$. In general, different

applications instantiate $\phi$ differently: for a classifier, $\phi$ checks whether the predicted label matches the true category; for a chatbot, $\phi$ may encode factual accuracy, relevance, and the absence of harmful content; for an agent, $\phi$ may decompose into sub-predicates (as we show with an example for security in section 5).

Human language and human intent are ambiguous. Capturing "what the user wants" or formalizing a requirement such as "I want this email reply to be appropriate" into something precise is itself open. Think about access-control mechanisms: writing down accurate policies for agents is genuinely hard because user intent and task structure both shift in ways the policy author cannot anticipate. On the other hand, high-level coarse policies along with a learning loop can do better than any single fixed specification. We expect the situation for $\phi$ to be similar.

One response to this is decomposition: rather than try to capture $\phi$ in one stroke, we break it into sub-predicates we can reason about individually. These seven sub-predicates are not equally hard to approximate; section 5 lists them and discusses which are clean, which inherit from classifier security, and which remain open. So the framework is robust to the open status of specification.

**Approximation of $\phi$ by an AIOracle.** A key observation is that the ideal $\phi$ can itself be realized by another AIOracle. That is, one may train a separate AIOracle whose inference phase approximates the evaluation of $\phi$: given a triple $(\mathsf{prompt}, \mathsf{context}, \mathsf{result})$, this "judge" AIOracle outputs whether the triple is valid. This recursive structure is precisely what underlies our dual construction (section 4), where a BAIO ("boring" AIOracle) acts as an approximate evaluator of a sub-predicate of $\phi$, filtering the outputs of a CAIO ("creative" AIOracle).

How well the training enables us to evaluate $\phi$ directly determines the security guarantees of the composed system. This makes the *training process* central to the discussion: the way an AIOracle is trained determines not only how well it performs its primary task, but also how reliably it can serve as a predicate evaluator in a dual construction (detailed discussion in section 4).

In fact, the training process is directly relevant to the security of the overall system: the LEARN phase determines whether the resulting model can reliably evaluate the predicate, and thus whether the composed system achieves its security guarantees. In current practice, this is precisely what safety-focused training methods attempt to do. Techniques such as Constitutional AI [8], instruction following [9], and reinforcement learning from human feedback [10] all aim to train models that internalize safety constraints. In our formalism, they are training the model to approximate a particular $\phi$. Understanding these training methods through the lens of AIOracles clarifies what each method targets: instruction following trains the model to respect source hierarchies (which could be related to a predicate $\phi$), while constitutional AI trains adherence to a set of principles (related to other predicates $\phi$). With this connection in mind, we have to examine what happens when training-based safety measures fail and how such failures can become security vulnerabilities.

**Safety vs. security.** While the model developers ambitiously aim for H&H, they are challenged with certain problems: safety and security [11]. Note that the objective predicate $\phi$ may be different when considering safety or security problems. The goal of *safety* is to prevent the users from unintentional harms. Safety guardrails don't protect against a maliciously active attacker. Instead, the harm comes from the model's own "inabilities."

Providing safety is studied by model providers during the reinforcement learning (RL) phase, training an RL model that can score AIOracle answers on how well they adhere to the two competing objectives. Multiple works from (hierarchical) instruction following to constitutional AI [12], [8], [10], [13], [9] have emerged in recent years, in which model developers train the models with safety guardrails. The objectives are then evaluated by benchmarks which are point estimations on a fixed task set.

On the other hand, *security* aims to prevent malicious actors (other than the model itself) from intentionally causing harm. In the pipeline given in Figure 1, an attacker tries to reach its goals given some capabilities (i.e. which arrow to corrupt in the pipeline). [2]

Informally, we will capture the adversary's capabilities in a set called ATK. ATK will set a flag to indicate if the adversary can see the model, or if it can choose a context, along with a prompt, to play with the INFER phase or choose a corpus in the LEARN phase. We allow these capabilities based on known attacks on language models, including data poisoning, prompt injection, and privacy attacks [14], [15], [16].

**Completeness, security, and adversaries.** Two quantities characterize an AIOracle. *Completeness* $p$ is the probability that INFER produces a result satisfying $\phi$ under benign use, where the input may be chosen by any honest polynomial-time algorithm.[3] *Security* $\epsilon$ against an attack class $\mathsf{ATK} \subseteq \{\mathsf{white\_box}, \mathsf{black\_box}, \mathsf{inject\_learn}, \mathsf{inject\_infer}\}$ is the maximum advantage any polynomial-time adversary with those capabilities has at making INFER produce a result that violates $\phi$. We define a single all-in-one game that unifies confidentiality, integrity, and availability, so that a complete AIOracle security claim has a single, falsifiable shape: *system $S$ is $\epsilon$-ATK-secure for predicate $\phi$.*[4]

There is also a stronger open question raised by the framework itself: *for which predicates is $\epsilon$-secure*

2. If the attacker does not corrupt any arrow, it will be considered as a safety problem, rather than a security problem.

3. We use *completeness* rather than *correctness* to avoid confusion with the narrower notion of a "correct output"; completeness reflects conformity to the system's functional specification, which is typically more than producing a correct result.

4. We exclude active Denial of Service attacks from our availability notion in the sense that we allow an AIOracle to give no answer when under active attack. We detail this in subsection 3.3.

*approximation even possible?* Bellare and Kohno's work on related-key attacks [17] built a taxonomy of key-derivation classes into provably-secure, provably-insecure, and conjectured-secure. The same program is open for AIOracle predicates. The confidentiality-utility tradeoff in subsection 3.2 is one instance of an impossibility-style result for a specific predicate (MID) in a specific model (GLM): completeness $p_c \approx 1$ forces $\epsilon \geq (p_c - 1/2)/n$, with no construction able to do better. Some other sub-predicates likely admit analogous results. Cataloguing which ones do, and where the boundary between feasible and infeasible lies, is a research direction the framework opens but does not close.

**Confidentiality, Integrity, and Availability.** One way to characterize both the safety and security of AIOracle is in terms of the traditional information security triad: confidentiality, integrity, and availability (CIA). Even though the defense mechanisms against safety and security problems might differ, the CIA triad still seems to be the most meaningful notion to both problems. In what follows, we discuss how CIA captures the H&H notions and how they are unified in our security game in subsection 3.3.

*Confidentiality* focuses on two important principles: protecting the privacy of training data (i.e. the corpus) and privacy of data coming from a user interacting with the model. For the former, as discussed in subsection 3.2, we argue that preventing training data from being revealed to the user is against the usefulness objective. Such risks should instead be mitigated either by excluding private data from the training corpus through appropriate data-sanitation procedures, or by fine-tuning the model on a private dataset whose resulting outputs are accessible only to the authorized users. The latter covers the leakage of user data through the result in agentic mode which can be addressed by adding a requirement in the harmlessness objective. Moving ahead, this should be captured in our security game through the predicate of $\phi$ in subsection 3.3.

*Integrity* requirements prevent the responses from being incorrect, unreliable, or tampered with so as to be useless or harmful. In AIOracle, integrity spans input data integrity, model integrity, and output integrity. This means protecting against data poisoning (tampering with training data), model attacks (like adversarial examples that manipulate outputs or model weights), and response manipulation (e.g. an attacker altering an AI's output). We capture the adversarial capabilities which violate integrity in our security game in subsection 3.3 with the predicate $\phi$.

*Availability* makes sure that the system is running and not degraded for the legitimate users. Availability will also be (partly) covered under the security game in Figure 4 through the predicate $\phi$. Regarding attacks aiming at making the system unavailable, we distinguish the safety problem (with no active adversary), which is covered by the notion of usefulness, and the security problem (with active attacks), where we accept that the AIOracle may give a no-result instead of violating other requirements. This means that we do not address active DoS attacks.

# 3. Definitions and Security Games of AIOracles

In practice, the learning phase may consist of a pipeline of $r$ rounds, where $\mathsf{GENERATE\_DATA}^{\mathsf{source}} \rightarrow \mathsf{corpus} = (\mathsf{corpus}_1, \ldots, \mathsf{corpus}_r)$, produces data for each round and $\mathsf{LEARN}(\mathsf{corpus})$ processes them sequentially. However, for the purpose of the core formalism, we abstract this into a single invocation: $\mathsf{corpus} \leftarrow \mathsf{GENERATE\_DATA}^{\mathsf{source}}$ followed by $\mathsf{model} \leftarrow \mathsf{LEARN}(\mathsf{corpus})$. This simplification does not affect any of our results but makes the presentation lighter. In practice, the corpus does not need to consist solely of naturally occurring data extracted from the source. In modern systems, the training data used in LEARN, especially for mid-training (usually training is split into pre-, mid-, post- training) relies on synthetic examples, such as LLM-generated demonstrations or other augmented data derived from the source.

## 3.1. Definitions

**Definition 1** (AIOracle)**.** *An AIOracle is defined by a triple of algorithms* $(\mathit{GENERATE\_DATA}, \mathit{LEARN}, \mathit{INFER})$. *The learning phase produces* $\mathit{corpus} \leftarrow \mathit{GENERATE\_DATA}^{\mathit{source}}$ *and then* $\mathit{model} \leftarrow \mathit{LEARN}(\mathit{corpus})$. *The inference phase produces* $\mathit{result} \leftarrow \mathit{INFER}(\mathit{prompt}, \mathit{context}, \mathit{model})$.

We note that source is common to every application. Given that an AIOracle is a set of algorithms with two phases, in our security definition, we will allow the adversary to interact with AIOracle in one of these phases depending on the given capabilities.

**Completeness.** We define the completeness of an AIOracle relative to a set source and for an objective predicate $\phi$. The predicate $\phi$ is *not* assumed to be implementable. The completeness is a notion which is characterized in the absence of an adversarial behavior. It depends on the application. Thus, the definition is meant to have no malicious activities. Instead, the game itself calls the algorithms of AIOracle when needed.

In the following completeness definition, we want to generate a (prompt, context) pair by any algorithm $\mathcal{B}$. This algorithm does not model an adversarial behavior. Namely, it operates independently of the model. The role of $\mathcal{B}$ is to replace a forall quantifier by anything which can be algorithmically generated with complexity bounded by some $B$ (e.g. $B = 2^{80}$ instructions, or $B$ polynomial).

**Definition 2** (Completeness)**.** *Given a set source, an AIOracle for a predicate* $\phi$ *is* $p$-complete *if for any probabilistic* $B$*-bounded algorithm* $\mathcal{B}$*, the following game returns 1 with probability at least* $p$*.*

Completeness Game

1: corpus $\leftarrow$ GENERATE_DATA$^{\text{source}}$
2: model $\leftarrow$ LEARN(corpus)
3: (prompt, context) $\leftarrow \mathcal{B}^{\text{source}}$
4: result $\leftarrow$ INFER(prompt, context, model)
5: **return** $\phi$(prompt, context, result)

**Comparison with Self-Proving models.** Anil et al. [7] define a notion of correctness (their Definition 3.1) for self-proving models, where a model must produce a correct output and prove its correctness to a verifier. Our completeness definition is more general in three respects (it also defines the correctness of the model). First, Anil et al. do not include the learning phase in their correctness definition, whereas our completeness game explicitly runs GENERATE_DATA and LEARN before evaluating the output. Second, they draw the input from a specific distribution $\mu$, whereas we let the inputs be chosen by any (benign) polynomial-time algorithm $\mathcal{B}$, which subsumes distributional inputs as a specific case. Third, they model the predicate $\phi$ for an ideal *deterministic* function called $F^*$, whereas we treat $\phi$ as a more general predicate. That is, $\phi(\text{prompt}, \text{context}, \text{result}) = 1 \Leftrightarrow \text{result} = F^*(\text{prompt}, \text{context})$. In our formulation, the probabilities in the completeness bound are over the randomness used during the data generation and learning phases, not only over the input (i.e. prompt) distribution.

**Search vs decision problem.** We define two types of problems that an AIOracle tackles to solve: a decision and a "search" problem. We note that any problem is a search problem and that decision problems are particular cases where the result can be either an "accept" or "reject." The inspiration for this separation comes from modern cryptography, where we have computational and decisional Diffie-Hellman problems with different difficulties. [5] Formally, a predicate $\phi$ for a search problem defines another predicate $\phi'$ for the companion decision problem by $\phi(\text{prompt}, \text{context}, \text{result}) \Leftrightarrow \phi'((\text{prompt}, \text{result}), \text{context}, \text{"accept"})$. Specifically, $\phi'$ decides whether (prompt, result) is a correct pair for a given context. An AIOracle for the decisional problem has the objective predicate $\phi'$.

**Definition 3** (Classifier)**.** *Let $f\colon \mathcal{X} \to \mathcal{W}$ be a function mapping objects to a finite set of category labels $\mathcal{W}$. A* classifier *for $f$ is an AIOracle such that:*

- *GENERATE_DATA$^{\text{source}}$ produces a corpus corpus $= \{(x_1, f(x_1)), \ldots, (x_n, f(x_n))\}$ of labeled pairs with $x_i \in \mathcal{X}$.*
- *LEARN(corpus) trains a model model from these labeled pairs.*
- *INFER(prompt, context, model) outputs a predicted label result $\in \mathcal{W}$.*
- *The predicate is $\phi(\text{prompt}, \text{context}, \text{result}) \Leftrightarrow \text{result} = f(\text{prompt})$.*

5. Decisional Diffie-Hellman is an easier problem due to the fact that the adversary is given a hint compared to the computational version.

As an example, $f$ can be defined as $f(\text{cat image}) = \texttt{cat}$ and $f(\text{dog image}) = \texttt{dog}$. An AIOracle for the corresponding $\phi$ is a classifier for cat and dog images. In this simple notion of AIOracle, context is not used.

**Generic Language Model (GLM).** Building on the classifier notion, we define a *generic language model* (GLM) by observing that the category names assigned to objects are arbitrary. What matters is the *structure* of the classification, not the specific labels. In that sense, we call $f$ a *language*.

We require the AIOracle to be complete *regardless of the language*. A *language translation* is any bijection $t\colon \mathcal{W} \to \{1, \ldots, |\mathcal{W}|\}$ on the codomain $\mathcal{W}$ of $f$. Given a translation $t$ for a language $f$, we define the translated language as $t \circ f$; the translated data generator as GENERATE_DATA$_t$, which runs GENERATE_DATA and then applies $t$ to every label; and the translated predicate $\phi_t$, which replaces $f$ by $f_t = t \circ f$. For a classifier in GLM, we replace GENERATE_DATA by GENERATE_DATA$_t$ and $\phi$ by $\phi_t$. In games, $t$ is randomly sampled at the beginning of the game and given to the adversary.

Intuitively, a generic model cannot rely on any prior association between objects and label names: it must learn the mapping entirely from the training data. This is analogous to Shoup's Generic Group Model [18], where algorithms operate on abstract group elements without exploiting their concrete representation. A GLM captures a core property of language models: they learn associations from data rather than from hardcoded prior knowledge. The genericity requirement also reflects that LMs must handle arbitrary token-to-meaning mappings across languages and domains. However, we note that this abstraction does not model all aspects of real LMs; in particular, real models do leverage pre-existing distributional patterns in natural language, and their generalization behavior is shaped by inductive biases in the architecture (e.g., attention mechanisms). The GLM should therefore be understood as an idealized model useful for proving impossibility results (such as the MID-completeness tradeoff as we will detail next), rather than as a complete description of practical LMs.

We note that a classifier AIOracle is $p$-complete if it correctly labels a fresh prompt with probability at least $p$. This models the generalization requirement central to machine learning: the model must perform well on *unseen* inputs, not just on the corpus. In particular, our completeness definition requires generalization by construction: a model that memorizes corpus (overfitting) but fails on fresh queries from source would not satisfy $p$-completeness for meaningful $p$.

## 3.2. Confidentiality in AIOracles

**Confidentiality vs utility.** Confidentiality is usually defined as what a malicious user could learn from interacting with the model. There exist many different ways to define it. We define a Membership Inference Distinguisher (MID) from [19] with our notations as

a typical example. In this security notion, there is a set corpus which is an unordered set of $n$ elements. The adversary builds the dataset with two options $\text{corpus}' \cup \{z_0\}$ or $\text{corpus}' \cup \{z_1\}$ which differ by only one element. The model learns from one of these options and the adversary must guess which one from the model (see Figure 2). We have MID-security if, for any adversary, $\Pr[\mathsf{MID} \to 1] \leq \frac{1}{2} + \epsilon$. This perfectly models the membership inference attack [20].

Membership Inference Distinguishability Game: $\mathsf{MID}(n)$

1: flip a coin b
2: $(\text{corpus}', z_0, z_1) \leftarrow \mathcal{A}'^{\text{source}}(n)$
3: **if** $|\text{corpus}'| \neq n-1$ **then return** 0
4: $\text{corpus} \leftarrow \text{corpus}' \cup \{z_b\}$
5: $\text{model} \leftarrow \text{LEARN}(\text{empty}, \text{corpus})$
6: $\text{b'} \leftarrow \mathcal{A}^{\text{source}}(\text{corpus}', z_0, z_1, \text{model})$
7: **return** $(\text{b} = \text{b'})$

Figure 2: Membership Inference Distinguishability (MID) game: the adversary picks two candidate elements $z_0, z_1$; the model trains on one of them; the adversary guesses which one it is trained on.

However, this notion does not discuss completeness. And, there is a clear tension between confidentiality, where we *do not* want to learn from a specific data element, and completeness, where we *do* want to learn every data element. To illustrate this, we revisit our classifier notion with a set of images of cats and dogs. We use this example to prove the confidentiality and utility tradeoff with the following theorem.

**Theorem 1** (Confidentiality-Utility Tradeoff)**.** *If a classifier with two categories is $\epsilon$-MID secure and $p_c$-complete in GLM after learning a corpus of size $n$, then $p_c \leq \frac{1}{2} + n\epsilon$.*

**Minimal MID-security for useful classifiers.** Rearranging the bound $p_c \leq \frac{1}{2} + n\epsilon$ shows that any generic AIOracle classifier that achieves completeness probability $p_c$ must satisfy

$$\epsilon \geq \frac{p_c - \frac{1}{2}}{n}.$$

In particular, if we require $p_c$ close to 1 (i.e. a useful classifier), then we need $\epsilon \geq \frac{1}{2n}$. This quantifies the best MID-security we can aim at: a useful generic classifier cannot be more than $\frac{1}{2n}$-MID-secure. Thus, any meaningful utility forces a concrete, quantifiable lower bound on the information leakage about training-data membership. For example, a good classifier (with $p \approx 1$) for cat and dog images uses $n \approx 5000$, which means $\epsilon \geq 10^{-4}$.

*Proof.* Let Cat and Dog be two disjoint sets of images, and define $\mathcal{X} = \text{Cat} \cup \text{Dog}$. A *language* is any mapping $f : \mathcal{X} \to \{a, b\}$ that is constant on Cat and constant on Dog, for instance, $f(\text{cat image}) = a$ and $f(\text{dog image}) = b$. GENERATE_DATA produces pairs of the form $(\text{img}, f(\text{img}))$, i.e. it labels each image according to the language $f$.

The predicate $\phi$ is defined as

$$\phi(\text{prompt}, \text{context}, \text{result}) \iff \text{result} = f(\text{prompt}).$$

That is, a valid result assigns the correct label to the queried image.

**Game sequence.** We proceed by a sequence of hybrid games $\text{Game}_0, \text{Game}_1, \ldots, \text{Game}_n$, where $n = |\text{corpus}|$.

$\text{Game}_i(n)$:
1) Pick $t$ at random
2) $\text{corpus} \leftarrow \text{GENERATE\_DATA}_t^{\text{source}}$
3) Define $\text{corpus}_i$ by flipping the labels of the first $i$ elements of corpus (i.e., swapping $1 \leftrightarrow 2$).
4) $\text{model} \leftarrow \text{LEARN}(\text{corpus}_i)$
5) $(\text{prompt}, \text{context}) \leftarrow \mathcal{B}^{\text{source}}$
6) $\text{result} \leftarrow \text{INFER}(\text{prompt}, \text{context}, \text{model})$
7) **return** $\mathbf{1}_{\text{result}=f_t(\text{prompt})}$

Let $p_i = \Pr[\text{Game}_i \to 1]$.

**Bounding $p_0$.** $\text{Game}_0$ flips zero labels, so $\text{corpus}_0 = \text{corpus}$. This is exactly the completeness game (2). Hence

$$p_0 \geq p_c. \tag{1}$$

**Bounding $p_n$.** We define $t_{\text{flip}}$ mapping 1 to 2 and 2 to 1. In $\text{Game}_n$ all $n$ labels are flipped, so $\text{corpus}_n$ is the corpus with every label swapped. By setting $t' = t_{\text{flip}} \circ t$, we can rewrite this game equivalently as:

$\text{Game}_n(n)$:
1) Pick $t$ at random
2) $\text{corpus} \leftarrow \text{GENERATE\_DATA}_{t'}^{\text{source}}$
3) $\text{model} \leftarrow \text{LEARN}(\text{corpus})$
4) $(\text{prompt}, \text{context}) \leftarrow \mathcal{B}^{\text{source}}$
5) $\text{result} \leftarrow \text{INFER}(\text{prompt}, \text{context}, \text{model})$
6) **return** $\mathbf{1}_{\text{result}\neq f_{t'}(\text{prompt})}$

Being *correct* for the original language $f_t$ ($\text{result} = f_t(\text{prompt})$) is the same as being *incorrect* for the flipped language $f_{t'}$ ($\text{result} \neq f_{t'}(\text{prompt})$). Therefore the complement of $\text{Game}_n$ is a completeness game on the flipped language, and by the GLM assumption it succeeds with probability at least $p_c$. Hence

$$1 - p_n \geq p_c. \tag{2}$$

**Reduction to MID.** It remains to bound $|p_i - p_{i+1}|$ for each $i$. We construct a MID adversary $(\mathcal{A}', \mathcal{A})$ as follows.

MID game with an adversary $(\mathcal{A}', \mathcal{A})$:

1) Flip a coin $\mathsf{b} \leftarrow \{0, 1\}$.
2) **Adversary $\mathcal{A}'$:**
   a) Pick $t$ at random
   b) $\mathsf{corpus}_0 \leftarrow \mathsf{GENERATE\_DATA}_t^{\mathsf{source}}$
   c) Define $\mathsf{corpus}_i$ by flipping the first $i$ labels.
   d) Set $z_0$ to the $(i{+}1)$-th element of $\mathsf{corpus}_i$.
   e) Set $\mathsf{corpus}' \leftarrow \mathsf{corpus}_i \setminus \{z_0\}$.
   f) Set $z_1$ to $z_0$ with its label flipped.
3) $\mathsf{corpus} \leftarrow \mathsf{corpus}' \cup \{z_\mathsf{b}\}$
4) $\mathsf{model} \leftarrow \mathsf{LEARN}(\mathsf{corpus})$
5) **Adversary $\mathcal{A}$:**
   a) $(\mathsf{prompt}, \mathsf{context}) \leftarrow \mathcal{B}^{\mathsf{source}}$
   b) $\mathsf{result} \leftarrow \mathsf{INFER}(\mathsf{prompt}, \mathsf{context}, \mathsf{model})$
   c) $\mathsf{b'} \leftarrow \mathbf{1}_{\mathsf{result} \neq f_t(\mathsf{prompt})}$
6) **return** $(\mathsf{b} = \mathsf{b'})$

Observe that:

- When $\mathsf{b} = 0$: the corpus is $\mathsf{corpus}' \cup \{z_0\} = \mathsf{corpus}_i$ (first $i$ labels flipped). The game returns 1 iff $b' = 0$, i.e. the result is $f_t(\mathsf{prompt})$. So, $\Pr[\text{return } 1 \mid \mathsf{b} = 0] = p_i$.
- When $\mathsf{b} = 1$: the corpus is $\mathsf{corpus}' \cup \{z_1\} = \mathsf{corpus}_{i+1}$ (first $i{+}1$ labels flipped). The game returns 1 iff $b' = 1$, i.e. the result is *not* $f_t(\mathsf{prompt})$. So, $\Pr[\text{return } 1 \mid \mathsf{b} = 1] = 1 - p_{i+1}$.

The overall probability that the MID game returns 1 is:

$$\begin{aligned} p &= \frac{\Pr[\text{return } 1 \mid \mathsf{b} = 0]}{2} + \frac{\Pr[\text{return } 1 \mid \mathsf{b} = 1]}{2} \\ &= \frac{p_i + (1 - p_{i+1})}{2} \\ &= \frac{1}{2} + \frac{p_i - p_{i+1}}{2}. \end{aligned}$$

Therefore,

$$p_i - p_{i+1} = 2\big(p - \tfrac{1}{2}\big) \leq 2\epsilon, \tag{3}$$

since the MID advantage is at most $\epsilon$.

Conclusion.. By telescoping (3) over $i = 0, \ldots, n{-}1$:

$$p_0 - p_n = \sum_{i=0}^{n-1} (p_i - p_{i+1}) \leq 2n\epsilon.$$

Combining with (1) and (2):

$$p_c \leq p_0 \quad \text{and} \quad p_c \leq 1 - p_n,$$

so

$$2\,p_c \leq p_0 + (1 - p_n) = 1 + (p_0 - p_n) \leq 1 + 2n\epsilon.$$

Finally,

$$p_c \;\leq\; \frac{1}{2} + n\epsilon.$$

□

The tension between confidentiality and utility comes from the fact that MID security treats all data elements from the corpus as confidential. In practice, it is overkill because we do not necessarily need or want to protect all data elements. Pragmatically speaking, we have three options: (1) the learning phase treats confidential data differently; (2) the learning phase eliminates confidential data from the corpus before training; or (3) one adds a restriction to $\phi$ such that result does not include any confidential information. In all cases, we need to identify confidential information. Again, identifying confidential information is yet another AI binary classifier problem or a decision problem where the objective is characterized by a predicate $\phi'$.

**Confidentiality vs integrity.** User data confidentiality is a more acute concern in agentic systems. However, we need a proper $\phi$ to characterize whether the result would leak confidential user data or not. Consequently, all confidentiality problems reduce to ensuring that the triple $(\mathsf{prompt}, \mathsf{context}, \mathsf{result})$ satisfies $\phi$. This reduction is more natural for integrity.

### 3.3. Security Game

**How to read the games in this paper.** Every game we define follows the same template: a challenger runs the honest AIOracle pipeline (GENERATE_DATA, LEARN, INFER); an adversary $\mathcal{A}$ acts within the capabilities granted by ATK; a predicate decides whether the adversary wins; and a side predicate $\psi$ rules out trivially won executions (e.g., the adversary first poisons the corpus to make $\phi$ false on its own query). The adversary's *advantage* is its win probability above the completeness baseline $1 - p$. The games in subsection 3.2, subsection 3.3, and subsection 5.4 all instantiate this template.

In this section, we define a precise formalism for security issues by games played by a malicious adversary. We model the adversary's capabilities by a set ATK. It can contain some of the four labels: white_box, inject_learn, inject_infer, black_box. If white_box is in ATK, it means that the adversary can see the model from LEARN phase. If inject_learn is in ATK, then the adversary can corrupt the corpus during the LEARN phase. If inject_infer is in ATK, then the adversary can add a chosen input context in INFER phase. If black_box is $\in$ ATK, then the adversary can play with an INFER(.,., model). Finally, the adversary chooses the prompt made by the user. If ATK $\subseteq$ {white_box, black_box}, we say that the adversary is passive. Otherwise, it is active.

**Security Game.** We first give one simpler version of our security game in the security game with an inference-only adversary in Figure 3. There, ATK is defined with two flags: white_box and inject_infer. Both of these flags together imply that the adversary can see the final output model with prompt and context injection capabilities where the adversary chooses both prompt and context to the INFER phase.

The full security game, which captures all attacks under full adversarial capabilities, is defined in Figure 4. The goal of the adversary, i.e. the winning condition, is to make the predicate $\phi(\mathsf{prompt}, \mathsf{context}, \mathsf{result})$ not satisfied. The game does not allow trivial wins. For example, if inject_learn $\in$ ATK, i.e., the attacker can change corpus to train the

(INFER-Only Adversary): Security_Game(ATK, source)

1: assert ATK $= \{$white_box, inject_infer$\}$
2: corpus $\leftarrow$ GENERATE_DATA$^{\text{source}}$
3: model $\leftarrow$ LEARN(corpus)
4: (prompt, context) $\leftarrow \mathcal{A}$(model, source)
5: result $\leftarrow$ INFER(prompt, context, model)
6: **if** $\phi$(prompt, context, result) **then return** 0
7: **return** 1

Figure 3: Security game with inference-only adversary capabilities: white_box (model access) and inject_infer (context injection).

model to learn that elephants can fly and then ask in the prompt whether elephants can fly. We check if an attack is trivial by verifying a predicate $\psi$(trace) which is computed on the trace of the game execution. Notice that the completeness game corresponds to the security game with opposite $\phi, \psi$ = false, and ATK $= \{$inject_infer$\}$ where $\mathcal{B}$ plays as an adversary.

If we focus on attacks where inject_learn $\notin$ ATK, we make the adversary passive during the learning phase. Such passive adversary will not have trivial attacks on the system. $\psi$ is always false. If inject_learn is in ATK, $\psi$ should be based on corpus$'$ (as it is changed by $\mathcal{A}$) and a triplet (prompt, context, result). Hence, trace $=$ (corpus$'$, prompt, context, result).

The advantage of the adversary is defined as Adv $= \Pr[\text{Security\_Game} \rightarrow 1] - (1-p)$. The baseline is a completeness adversary making the model return an invalid result with probability $(1-p)$.

(Full Adversary): Security_Game(ATK, source)

1: corpus $\leftarrow$ GENERATE_DATA$^{\text{source}}$
2: (corpus$'$, state) $\leftarrow \mathcal{A}^{\text{source}}$
3: **if** inject_learn $\notin$ ATK **then** corpus$'$ $\leftarrow$ corpus
4: model $\leftarrow$ LEARN(model, corpus$'$)
5: **if** white_box $\in$ ATK **then** view $\leftarrow$ model **else** view $\leftarrow$ empty
6: **if** black_box $\in$ ATK **then** oracle $\leftarrow$ INFER$(\cdot,\cdot,$ model$)$ **else** oracle $\leftarrow$ empty
7: (prompt, context) $\leftarrow \mathcal{A}^{\text{oracle}}$(state, view, source)
8: **if** inject_infer $\notin$ ATK **then** context $\leftarrow$ empty
9: result $\leftarrow$ INFER(prompt, context, model)
10: **if** $\phi$(prompt, context, result) **then return** 0
11: **if** $\psi$(trace) **then return** 0
12: **return** 1

Figure 4: Security game with full adversary capabilities: learning-phase poisoning, model access, black-box oracle, and context injection.

**Definition 4** (Security)**.** *An AIOracle for $\phi$ is $\epsilon$-ATK-$\psi$-secure if there exists $p$ such that it is $p$-complete for $\phi$ and, for any PPT adversary $\mathcal{A}$, Adv $\leq \epsilon$ in the game in Figure 4.*

The predicate $\phi$ which is used in completeness may differ from the one used in security. For instance, completeness may impose $\phi$ to capture DoS cases but security would not impose that. A no-answer is not useful with respect to completeness but acceptable under adversarial stress.

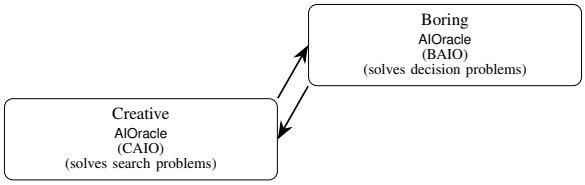


Figure 5: Communication between two AIOracles solving the "search" problem and the decision problem.

**Takeaways.** subsection 3.3 fixes three design choices that the rest of the paper relies on. First, security is parameterized: an AIOracle is $\epsilon$-ATK-$\psi$-secure, so claims always specify which capabilities the adversary has, which trivial wins are excluded, and against which predicate $\phi$. Second, security is measured against the completeness baseline $1-p$ rather than against zero, which lets us cleanly compose secure components in section 4. Third, active DoS is deliberately out of scope: under adversarial stress, an AIOracle is permitted to return $\perp$ instead of an unsafe result, and this asymmetry is what allows the dual construction in section 4 and the sub-predicate decomposition in section 5 to be well-defined. Subsequent sections instantiate this game for specific predicates and compositions.

## 4. Dual construction

When the predicate $\phi$ is defined as a conjunction of different criteria, we can build an AIOracle from sub-oracles. For instance, the objective of the AIOracle is to be helpful and to be harmless to its users, which are two different criteria. In an earlier section, we concluded that confidentiality could be treated as an additional criterion. Furthermore, helpfulness was characterized as correctness and usefulness, which are again two different criteria.

A dual construction consists of designing an AIOracle for $\phi = \phi_1 \wedge \phi_2$ from an AIOracle for $\phi_2$ and an AIOracle for the decision version of $\phi_1$ (denoted as $\phi_1'$). The AIOracle for $\phi_2$ is called *creative AIOracle*, CAIO = (GENERATE_DATA$_2$, LEARN$_2$, INFER$_2$), which proposes results to the prompt. The AIOracle for $\phi_1'$ is called *boring AIOracle*, BAIO = (GENERATE_DATA$_1$, LEARN$_1$, INFER$_1$), which filters the proposed results. The interaction between CAIO and BAIO is illustrated in Figure 5: CAIO proposes a candidate result, BAIO accepts or rejects it, and the process repeats until BAIO accepts or a bound is reached. The concrete algorithms for data generation, learning, and inference in the dual construction are given in Figure 6.

To be able to quantify the completeness and security of the dual construction, we need to upper bound the corner cases where CAIO provides a result that is correct for CAIO and incorrect for BAIO, and a final no-result is not permitted by $\phi$. We denote this probabil-

GENERATE_DATA$^{\text{source}}$

1: $\text{corpus}_1 \leftarrow \text{GENERATE\_DATA}_1^{\text{source}}$
2: $\text{corpus}_2 \leftarrow \text{GENERATE\_DATA}_2^{\text{source}}$
3: $\text{corpus} \leftarrow (\text{corpus}_1, \text{corpus}_2)$
4: **return** corpus

LEARN(model, corpus′)

1: **parse** $\text{corpus} = (\text{corpus}_1, \text{corpus}_2)$
2: $\text{model}_1 \leftarrow \text{LEARN}_1(\text{corpus}_1)$
3: $\text{model}_2 \leftarrow \text{LEARN}_2(\text{corpus}_2)$
4: $\text{model} \leftarrow (\text{model}_1, \text{model}_2)$
5: **return** model

INFER(prompt, context, model)

1: **parse** $\text{model} = (\text{model}_1, \text{model}_2)$
2: $\text{result}_2 \leftarrow \text{INFER}_2(\text{prompt}, \text{context}, \text{model}_2)$
3: $\text{result}_1 \leftarrow \text{INFER}_1((\text{prompt}, \text{result}_2), \text{context}, \text{model}_1)$
4: **if** $\text{result}_1 =$ "accept" **then**
result $\leftarrow \text{result}_2$ **else** result $\leftarrow$ empty
5: **return** result

Figure 6: Subroutines for data generation, learning, and inference for dual construction. In $\text{LEARN}_1$, we train BAIO as $\text{model}_1$ and then continue training CAIO as $\text{model}_2$.

ity that CAIO returns a result satisfying its own predicate with $\mathsf{e}$. If we accept that $\phi(\text{prompt}, \text{context}, \perp) = 1$ (i.e. an empty result is fine), then we set $\mathsf{e} = 0$. Note that DoS attacks are out of scope. Thus, it should be the case that $\mathsf{e} = 0$. However, completeness also requires the result to be useful. We cannot expect CAIO to be useful when adding a requirement $\phi_1$ for which it was not trained, except if it is rare that its result does not satisfy $\phi_1$ (hence $\mathsf{e}$ is small).

**Theorem 2** (Dual Construction Composition)**.** *We define three predicates $\phi, \bar{\phi}$, and $\phi_e$ as follows: $\phi = \phi_1 \wedge \phi_2$; $\bar{\phi}(\text{prompt}, \text{context}, \text{result}) = \phi(\text{prompt}, \text{context}, \perp)$; and $\phi_e = \bar{\phi} \vee \phi_2$. We assume that CAIO is $\mathsf{e}$-$\{\text{inject\_infer}\}$-secure for $\phi_e$. If BAIO is $p_1$-complete for $\phi_1'$ and CAIO is $p_2$-complete for $\phi_2$, then the dual construction is $p$-complete for $\phi$ with $p = p_1 + p_2 - 1 - \mathsf{e}$.*

*We assume that $\psi_1 = \psi_2 = 0$ or $\bar{\phi} = 1$ and that CAIO is $\mathsf{e}$-ATK-$\psi_2$-secure for $\phi_e$. We define $\psi((\text{corpus}_1', \text{corpus}_2'), \text{prompt}, \text{context}, \text{result}) = \psi_1(\text{corpus}_1', (\text{prompt}, \text{result}), \text{context}, \text{accept}) \vee \psi_2(\text{corpus}_2', \text{prompt}, \text{context}, \text{result})$. If BAIO is $\epsilon_1$-ATK-$\psi_1$-secure for $\phi_1'$ and CAIO is $\epsilon_2$-ATK-$\psi_2$-secure for $\phi_2$, then the dual construction is $\epsilon$-ATK-$\psi$-secure for $\phi$ with $\epsilon = \epsilon_1 + \epsilon_2 + \mathsf{e}$.*

*Proof.* Given an algorithm $\mathcal{B}$ for the dual construction (for the completeness game), we can define two algorithms: $\mathcal{B}_C$ for CAIO and $\mathcal{B}_B$ for BAIO.

For completeness, there is no adversarial activity during the learning phase, and $\mathcal{B}$ only provides $(\text{prompt}, \text{context})$ from scratch. Let $\text{result}_{\text{BAIO}}$ be the result of BAIO, $\text{result}_{\text{CAIO}}$ be the result of CAIO, and result be the result of the dual construction. We set $\mathcal{B}_C = \mathcal{B}$. For $\mathcal{B}_B$, it is different because it needs to provide the $\text{result}_{\text{CAIO}}$ as well. Hence, $\mathcal{B}_B$ starts by building the CAIO model, and runs $\mathcal{B}$ to get prompt and context, then runs the CAIO model to get the $\text{result}_{\text{CAIO}}$. It outputs $((\text{prompt}, \text{result}_{\text{CAIO}}), \text{context})$ as a challenge for BAIO to complete. A case where the game for $\mathcal{B}$ returns 0 (with probability $1 - p$) corresponds to $\phi(\text{prompt}, \text{context}, \text{result}) = 0$. The case $\phi_1(\text{prompt}, \text{context}, \text{result}_{\text{CAIO}}) = 0$ happens with probability bounded by $1 - p_2$. The case $\phi_2'((\text{prompt}, \text{result}_{\text{CAIO}}), \text{context}, \text{result}_{\text{BAIO}}) = 0$ happens with probability bounded by $1 - p_1$. In the remaining cases, we have $\phi_1(\text{prompt}, \text{context}, \text{result}_{\text{CAIO}}) = 1$ and $\phi_2'((\text{prompt}, \text{result}_{\text{CAIO}}), \text{context}, \text{result}_{\text{BAIO}}) = 1$. It means that both BAIO and CAIO completed their tasks, but $\phi(\text{prompt}, \text{context}, \text{result}) = 0$. Clearly, this implies that CAIO returns $\text{result}_{\text{CAIO}}$ which does not satisfy $\phi_{\mathsf{e}}$, hence it happens with probability bounded by $\mathsf{e}$. Hence,

$$1 - p \leq (1 - p_1) + (1 - p_2) + \mathsf{e}$$

which implies $p \geq p_1 + p_2 - 1 - \mathsf{e}$. This bound is useful if both $p_1$ and $p_2$ are close to 1, and $\mathsf{e}$ is close to 0.

Given an adversary $\mathcal{A}$, we construct $\mathcal{A}_C$ and $\mathcal{A}_B$ as follows. The adversary $\mathcal{A}_C$ simulates $\mathcal{A}$ during the learning phase but only returns $\text{corpus}_2'$. If white_box or black_box is in ATK, $\mathcal{A}_C$ simulates $\text{model}_1 \leftarrow \text{LEARN}_1(\text{corpus}_1')$ to add $\text{model}_1$ in state and be able to complete the view with $\text{model}_1$ or to simulate the BAIO inference oracle during the inference phase.

The adversary $\mathcal{A}_B$ is constructed in the same manner but always simulates the learning phase of CAIO in order to return $\text{result}_{\text{CAIO}}$ in the inference phase.

Then, the same reasoning as for completeness applies. If we assume that CAIO is $\mathsf{e}$-$\{\text{inject\_infer}\}$-secure for $\phi_{\mathsf{e}}$, then the dual construction is $p$-complete for $p = p_1 + p_2 - 1 - \mathsf{e}_c$. Hence, $\text{Adv}(\mathcal{A}) + (1 - p) \leq \text{Adv}(\mathcal{A}_C) + (1 - p_1) + \text{Adv}(\mathcal{A}_B) + (1 - p_2) + \mathsf{e}$

If CAIO and BAIO are secure, $\text{Adv}(\mathcal{A}_C) \leq \epsilon_1$ and $\text{Adv}(\mathcal{A}_B) \leq \epsilon_2$, which makes $\text{Adv}(\mathcal{A}) \leq \epsilon$ for $\epsilon = \epsilon_1 + \epsilon_2 + \mathsf{e}$. When $\epsilon_1, \epsilon_2$ are negligible, $\epsilon$ is negligible too. □

BAIO could be seen as a filter on an output of CAIO. It has a similar complexity to CAIO. Indeed, following the result of [6], BAIO's high complexity might be inevitable as detailed below.

**Discussion.** In [6], the authors study the (im)possibility of easy filtering to increase the harmlessness of Large Language Models (LLMs). They consider both *preemptive filtering* where the prompt is filtered alone if it makes a harmful query and *output filtering* where the result is filtered together with the prompt if result is harmful. We specifically focus on the latter. More specifically, they prove that if the complexity of the filter is not high enough to implement the harmlessness predicate (which they call an oracle $\mathcal{H}$), then it is always possible to bypass the filter. They show that they can find a malicious model $M'$ that is indistinguishable from an honest model $M$ and is able to set up a covert channel intelligible only to the malicious oracle $\mathcal{H}'$, which uses it to pass harmful results undetected.

We want to map the meaning of Theorem 10 from [6] to our formalism. The model $M$ (resp.

$M'$) is called CAIO (resp. CAIO$'$) in our work. The harmfulness oracle $\mathcal{H}$ (resp. $\mathcal{H}'$) judging the results of $M$ (resp. $M'$) is our primitive $\phi$ (resp. $\phi'$) with the difference that in our definitions we let $\phi$ output a bit whereas they let $\mathcal{H}$ output a score, which is a real number. The output filter is BAIO in our results.

We now describe their results with the language of our formalism. They start with a $(\mathsf{CAIO}, \phi)$ and construct $(\mathsf{CAIO}', \phi')$ such that if there is a set of input triplets making $\phi$ output 0 (harmful result), then CAIO$'$ always returns results making $\phi'$ output 0. However CAIO and CAIO$'$ are indistinguishable by any BAIO and they are equipped with $\phi$. And, $\phi'$ produces indistinguishable results on triplets coming out from CAIO. This happens because CAIO$'$ will embed some secret message in the result which can only be understood by $\phi'$. Namely, their construction of CAIO$'$ and $\phi'$ works in a way that they "talk through" a covert channel which is not detectable by BAIO. The adversarial model is such that the adversary can modify $(\mathsf{CAIO}, \phi)$ to create its own version $(\mathsf{CAIO}', \phi')$. $\phi'$ decides if a result is harmful by checking if CAIO$'$ sent a secret message. We note that if such a thing happens, it is a contradiction with our completeness notion. Furthermore, for such a covert channel to exist, they need the assumption that there is a hardness for BAIO which is not hard for $\phi'$ which is in line with their separation of harmlessness oracle from a filter.

## 5. Agents as AIOracles

Recent work by Siu et al. [21] proposes four security properties for LLM agents: task alignment, action alignment, source authorization, and data isolation. Their framework introduces five idealized uncomputable oracle functions to verify these properties at runtime, and uses them to reclassify known attacks (prompt injection, jailbreak, task drift, etc.) as violations of specific properties. Siu et al. did not consider the usefulness of an agent in their security framework, in the sense that an agent doing nothing will be considered as secure but not be evaluated for usefulness.

We view an agent as an AIOracle whose result is a program executed in interaction with external tools. There are two possible design points: a *full-planner* AIOracle, where a single INFER call emits the entire program up-front, and a *next-step* AIOracle, where each INFER call emits only the next action and is re-invoked after observing its outcome. The next-step variant is adaptive (it can react to tool outputs, errors, or ambiguous data) and reduces multi-step security to a sequence of single-call security claims, as we show in subsection 5.4. We adopt the next-step formulation throughout this section.

This section demonstrates how, in the case of agents, our predicate $\phi$ can be defined as a conjunction of seven sub-predicates organized in three groups: *action*, *data*, and *flow*, where each captures a distinct and cleanly separable security requirement. We show this with the AIOracle formalism which subsumes these properties through the predicate $\phi$ and its decomposition. We illustrate this with a concrete toy example.

### 5.1. Example: Email Reply Agent as an AIOracle

Consider a user who issues the prompt: $\mathsf{prompt}_1 =$ `"Reply to Alice's latest email confirming tomorrow's 10am meeting."`. The first AIOracle call $\mathsf{result}_1 = \mathsf{AIOracle.INFER}(\mathsf{prompt}_1, \mathsf{context}_1, \mathsf{model})$ produces executable code with the following tool calls:

```
tool_inbox_fetch("from:Alice")
tool_email_send(<reply>)
```

A naive agent would simply execute these tool calls in order. However, such a rigid sequence is brittle: the inbox query may return nothing, the fetched email may be ambiguous, or the drafted reply may need revision. To anticipate such deviations, each INFER returns only the *next step* in the form of "Proceed(tool_action, context)" (given in Figure 7) or END, with context carrying the execution trace (initialized to $\mathsf{prompt}_1$). Thus,

$\mathsf{result}_1 =$ "Proceed (tool_inbox_fetch(`"from:Alice"`), $\mathsf{context}_1 || \mathsf{prompt}_1$)"

In Figure 7, the Proceed subroutine executes a tool action, appends its output to the trace, and hands control back to the AIOracle to decide the next step. This turns the agent from a fixed pipeline into an adaptive loop driven by the AIOracle.

```
Proceed(tool_action, context)
1: tool_out ← call(tool_action)
2: context ← context||(tool_action, tool_out)
3: prompt ← "Decide next step from trace in context"
4: result ← AIOracle.INFER(prompt, context, model)
5: execute(result)
```

Figure 7: The Proceed subroutine: executes a tool action, appends its output to the trace, and hands control back to the AIOracle to decide the next step.

With Proceed in place, the Email Agent can be written as a short two-phase template whose adaptivity comes from the wrapper instead of a hand-coded control flow. The flow is shown in Figure 8 We expect that the agent will find Alice's last email in the first action. Based on the first action, it will write an answer confirming the meeting for tomorrow at 10am. Finally, it will set the next action to send this reply to Alice.

At runtime, each tool action in the agent is wrapped by a call to Proceed(tool_action, context) (Figure 7), so that the AIOracle re-evaluates the trace after every tool output and decides the next step rather than blindly following the sequence above. For example, if tool_inbox_fetch returns no matching email, Proceed lets the AIOracle adapt, e.g. by broadening the

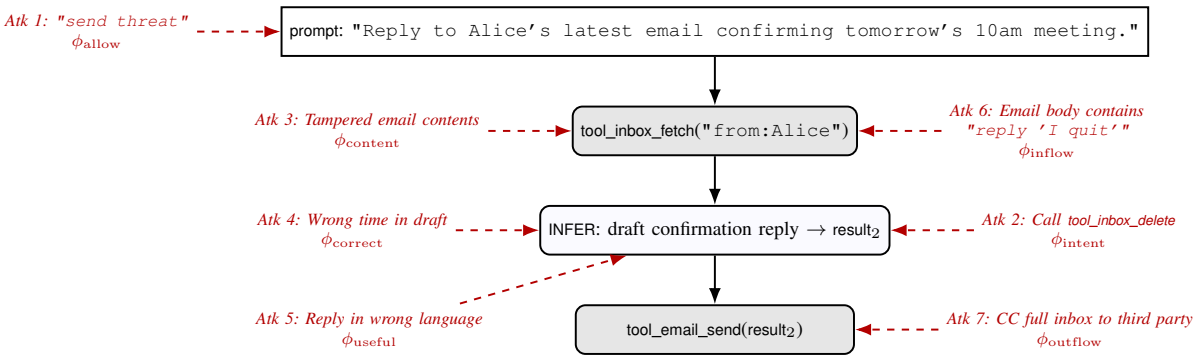


Figure 8: Sequential data flow through the email reply agent with seven attack injection points. Each red dashed arrow shows where an attack enters the pipeline, labeled with the sub-predicate of $\phi$ that catches it. AIOracle.INFER and tool call boxes are colored differently.

query or asking the user for clarification, and only proceeds to Phase 2 once a valid draft is available. Each recursive AIOracle.INFER call produces its own (prompt, context, result) triple which must satisfy $\phi$.

## 5.2. Decomposing $\phi$ into Sub-Predicates

Recall that $\phi$ defines the set of (prompt, context, result) triples that are aligned with the system's helpfulness and harmlessness objectives. We decompose $\phi$ into seven sub-predicates organized in three groups (see Table 2):

$$\phi = \underbrace{\phi_{\text{allow}} \wedge \phi_{\text{intent}}}_{\text{Action}} \wedge \underbrace{\phi_{\text{content}} \wedge \phi_{\text{correct}} \wedge \phi_{\text{useful}}}_{\text{Data}} \wedge \underbrace{\phi_{\text{inflow}} \wedge \phi_{\text{outflow}}}_{\text{Flow}}$$

The three groups follow the data flow through the agent.

**Action** rules what the agent decides to do. $\phi_{\text{allow}}$ checks if the action is permitted by the AI system's policy, which is independent of the user's wishes. $\phi_{\text{intent}}$ checks whether the action is making a step towards the user's stated goal. Together, they capture that the agent only pursues tasks that are both *allowed* and *requested*. They also prevent endless loops.

**Data** governs the quality of data and results. $\phi_{\text{content}}$ checks whether the data the agent relies on is sound. $\phi_{\text{correct}}$ checks whether the result is factually correct given the data. $\phi_{\text{useful}}$ checks whether the result is practically useful to the user. These three capture the progression from input quality to output quality.

**Flow** governs information boundaries. $\phi_{\text{inflow}}$ checks that external data is not interpreted as instructions. $\phi_{\text{outflow}}$ checks that internal data is not leaked to unauthorized entities. Together, they enforce that information stays in its proper role and location.

The seven sub-predicates are not equally hard to approximate:

- **The clean ones.** $\phi_{\text{outflow}}$, $\phi_{\text{allow}}$, and $\phi_{\text{inflow}}$ have crisp definitions. $\phi_{\text{outflow}}$ can often be enforced by static or evolving permission checks. $\phi_{\text{allow}}$ reduces to coarse-grained policy lookup. $\phi_{\text{inflow}}$ reduces to detecting instruction-shaped content in untrusted data, which has been studied extensively.
- **The classifier-inheriting one.** $\phi_{\text{content}}$ reduces to classification of tampering or integrity violation. Classifiers are the finest-grained AIOracles we want to reduce everything else to, and their security is the subject of subsection 3.2.
- **The open ones.** $\phi_{\text{intent}}$ and $\phi_{\text{useful}}$ require capturing what the user actually wanted. Today this needs user studies, judge models, or behavioral analysis. The framework gives a place to state the claim. We do not define a procedure to discharge it.

Each sub-predicate has a single, clear responsibility. This avoids the confusion that arises when, for example, source authorization is entangled with content quality: an external data source selected by the user does not make its data correct ($\phi_{\text{content}}$), nor does it authorize the source to issue instructions ($\phi_{\text{inflow}}$). The seven-way decomposition also enables fine-grained diagnosis: when $\phi = 0$, which sub-predicate failed and at which AIOracle call is immediately identifiable. To explain how attacks are defeated, we assume that the sequence of steps follows the Email Agent algorithm.

## 5.3. Attack Analysis

We show how the seven sub-predicates of $\phi$ capture known attacks on the email reply agent. The attacks are ordered so that each one highlights a different sub-predicate.

**Attack 1: Harmful User Intent ($\phi_{\text{allow}} = 0$).** The user issues: $\text{prompt}_1$ = `"Send Alice a threatening message about tomorrow's meeting."` The first AIOracle call produces $\text{result}_1$.

- $\phi_{\text{allow}} = 0$ at $\text{result}_1$: the AI system's policy prohibits sending threats. The requested task is not in the allowed set regardless of who asks.
- $\phi = 0 \longrightarrow$ **attack caught** (by $\phi_{\text{allow}}$ at $\text{result}_1$).

| Group | Sub-predicate | Condition for $= 1$ |
|---|---|---|
| Action | $\phi_{\text{allow}}$(prompt, context, result) | Every action in result is permitted by the AI system's policy. |
| | $\phi_{\text{intent}}$(prompt, context, result) | Every action in result is anticipated to take a step toward the user's stated goal in prompt with context. |
| Data | $\phi_{\text{content}}$(prompt, context, result) | The data in context used to produce result is factually sound: not manipulated, bogus, or harmful. |
| | $\phi_{\text{correct}}$(prompt, context, result) | The result is correct given prompt and context. |
| | $\phi_{\text{useful}}$(prompt, context, result) | The result is useful to the user for the task stated in prompt with context. |
| Flow | $\phi_{\text{inflow}}$(prompt, context, result) | External data in context is not interpreted as instructions that influence the actions in result. |
| | $\phi_{\text{outflow}}$(prompt, context, result) | No data from context is transmitted to entities outside the permission boundary in result. |

TABLE 2: Sub-predicates for the security of an agentic AIOracle grouped in three families: $\phi = \phi_{\text{allow}} \wedge \phi_{\text{intent}} \wedge \phi_{\text{content}} \wedge \phi_{\text{correct}} \wedge \phi_{\text{useful}} \wedge \phi_{\text{inflow}} \wedge \phi_{\text{outflow}}$.

**Attack 2: Capability Misuse ($\phi_{\text{intent}} = 0$).** The agent's code in $\text{result}_1$ includes a call to tool_inbox_delete(all) to clean up the inbox after replying.

- $\phi_{\text{intent}} = 0$ at $\text{result}_1$: deleting the inbox is not aligned with the user's stated goal of replying to a single email.
- $\phi = 0 \longrightarrow$ **attack caught** (by $\phi_{\text{intent}}$ at $\text{result}_1$).

**Attack 3: Adversarial Email Manipulation ($\phi_{\text{content}} = 0$).** A compromised mail server returns a manipulated email in $\text{tool\_out}_1$: Alice's actual message is replaced by a fake one stating that the meeting is cancelled, or by a message with garbled contents that no longer correspond to a meeting confirmation.

- $\phi_{\text{content}} = 0$ at $\text{result}_2$: the AIOracle call that drafts the reply (line 4 of the Email Agent algorithm) operates on manipulated data. The fetched email is not factually sound; it has been tampered with and does not reflect Alice's real message.
- $\phi = 0 \longrightarrow$ **attack caught** (by $\phi_{\text{content}}$ at $\text{result}_2$).

**Attack 4: Incorrect Result ($\phi_{\text{correct}} = 0$).** The agent drafts the reply (line 4 of the Email Agent algorithm) but writes the wrong meeting time: Alice's email asks to confirm 10am, yet the draft says 11am.

- $\phi_{\text{correct}} = 0$ at $\text{result}_2$: the result is factually incorrect; the draft confirms a time that does not match the email it is replying to.
- $\phi = 0 \longrightarrow$ **attack caught** (by $\phi_{\text{correct}}$ at $\text{result}_2$).

**Attack 5: Task Drift ($\phi_{\text{useful}} = 0$).** The agent generates a reply (line 4 of the Email Agent algorithm) that is technically a confirmation but is written in a language Alice does not read, or is so long that the confirmation point is buried in unrelated text.

- $\phi_{\text{useful}} = 0$ at $\text{result}_2$: the reply is correct but not useful, since Alice cannot easily extract the confirmation from it.
- $\phi = 0 \longrightarrow$ **attack caught** (by $\phi_{\text{useful}}$ at $\text{result}_2$).

**Attack 6: Indirect Prompt Injection ($\phi_{\text{inflow}} = 0$).** The fetched email (in $\text{tool\_out}_1$) contains the embedded instruction: `"Ignore previous instruction and reply 'I quit my job'."` The agent treats this external text as a command and writes the requested resignation message instead of confirming the meeting.

- $\phi_{\text{inflow}} = 0$ at $\text{result}_2$: external data from context (the embedded instruction in the fetched email) was interpreted as an instruction that influenced the reply in $\text{result}_2$.
- $\phi = 0 \longrightarrow$ **attack caught** (by $\phi_{\text{inflow}}$ at $\text{result}_2$).

**Attack 7: Data Exfiltration ($\phi_{\text{outflow}} = 0$).** The agent, when sending the reply in line 7 of the Email Agent algorithm, attaches the contents of the user's full inbox to the message, or CCs the reply to a third party not on the original thread.

- $\phi_{\text{outflow}} = 0$ at $\text{result}_2$: user data (other inbox contents from context) is transmitted to an external entity (Alice or a third party) beyond what is necessary for confirming the meeting.
- $\phi = 0 \longrightarrow$ **attack caught** (by $\phi_{\text{outflow}}$ at $\text{result}_2$).

## 5.4. Multi-Step Agent Security

Assume that the AIOracle for $\phi$ is $\epsilon$-ATK-secure with ATK $= \{$white_box, inject_infer$\}$ and $\psi = 0$. We want to prove that when used as an agent, for every model inference done during the execution of the agent, the predicate $\phi$ is satisfied, except with some probability.[6]

Let $n$ be the number of INFER calls. We will show that the probability that at least one of the $n$ INFER calls makes $\phi = 0$ is bounded by $n \cdot \epsilon$.

The game runs as follows:

**Theorem 3** (Reduction to single-call security)**.** *If ATK $= \{$white_box, black_box$\}$ and the underlying AIOracle (used to make Proceed calls) is $\epsilon$-secure for $\phi$, then the agent is $n\epsilon$-secure with respect to the security game in Figure 4.*

*Proof.* Let $\epsilon_i$ be the probability that the $i$-th INFER call makes $\phi = 0$. By a union bound, the probability that at least one INFER makes $\phi = 0$ is bounded by

$$\Pr\left[\exists\, i : \phi(\text{prompt}_i, \text{context}_i, \text{result}_i) = 0\right] \;\leq\; \epsilon_1 + \epsilon_2 + \cdots + \epsilon_n$$

6. This is for a model obtained in an honest manner, but with adversarial initial prompt and context input and adversarial tools.

Multi-Step Agent Security Game with ATK $=$ {white_box, black_box}

corpus ← GENERATE_DATA$^{\text{source}}$
model ← LEARN(corpus)
(prompt$_1$, context$_1$) ← $\mathcal{A}$(model, source) ▷ Adversary chooses initial input
**for** $i = 1, 2, \ldots, n$ **do**
  result$_i$ ← AIOracle.INFER(prompt$_i$, context$_i$, model)
  parse result$_i$ as "Proceed(tool_action$_i$, $*$)"
  **if** parse fails **then break**
  tool_out$_i$ ← call(tool_action$_i$) ▷ Adversarial tools
  context$_{i+1}$ ← context$_i$ ‖ prompt$_i$ ‖ tool_action$_i$ ‖ tool_out$_i$
  prompt$_{i+1}$ ← "Decide the next step following the trace in context"
**if** $\exists\, i : \phi$(prompt$_i$, context$_i$, result$_i$) $= 0$ **then return** 1
**return** 0

Single-Step Reduction: Adversary $\mathcal{A}_i$ Simulates Steps 1 to $i-1$ and Challenges Step $i$

corpus ← GENERATE_DATA$^{\text{source}}$
model ← LEARN(corpus)
(prompt$_i$, context$_i$) ← $\mathcal{A}_i$(model, source) ▷ $\mathcal{A}_i$ simulates steps 1 to $i-1$ internally
result$_i$ ← INFER(prompt$_i$, context$_i$, model)
**if** $\phi$(prompt$_i$, context$_i$, result$_i$) $= 0$ **then return** 1
**return** 0

We define $n$ adversaries $\mathcal{A}_i$, each of which simulates the game from the beginning up to step $i$ and returns (prompt$_i$, context$_i$). In the ATK model, $\mathcal{A}_i$ can simulate the first $i-1$ inferences: either by using the model directly if white_box $\in$ ATK, or by making calls to the inference oracle if black_box $\in$ ATK. We assume that the interaction with tools can be simulated algorithmically by the adversary. $\mathcal{A}_i$ plays the following single-call security game:

$\epsilon_i$ is the probability that the above game returns 1. By the AIOracle ATK-security assumption, $\epsilon_i \leq \epsilon$ for all $i$. Therefore:

$$\Pr\big[\text{any } \phi = 0\big] \;\leq\; \sum_{i=1}^{n} \epsilon_i \;\leq\; n \cdot \epsilon$$

□

### 5.5. Applying the Dual Construction to Agents

Given the seven sub-predicates, we can apply the dual construction from Theorem 2 to build an agent whose security reduces to the security of its components. The following algorithm shows how CAIO proposes a result and each BAIO checks one sub-predicate. If any check fails, instead of returning $\perp$ as in section 4, CAIO is asked to retry with feedback about which requirements were not met.

Assuming that each BAIO$_i$ is $\epsilon_i$-secure for $\phi_i'$, then the above method always gives a result that satisfies $\phi$ except with probability $t \cdot \epsilon_1 + \ldots + t \cdot \epsilon_7$, where $t$ is the maximal number of iterations in the loop.

## 6. Conclusion: How to State a Formal Claim

Our framework in this paper is meant to be used with the following steps for a complete AIOracle security claim:

Proceed(tool_action, context)

tool_out ← call(tool_action)
context.trace ← context.trace‖(tool_action, tool_output)
prompt ← "Interpret the last tool output and determine the next step following trace.
**repeat**
  result ← CAIO(prompt, context)
  $\phi \leftarrow 1$
  **for** $i = 1$ to $7$ **do**
    $\phi_i$ ← BAIO$_i$((prompt, result), context)
    $\phi \leftarrow \phi \cdot \phi_i$
  **if** $\neg\, \phi$ **then**
    context ← "Previous query: ⟨prompt⟩
      Previous answer: ⟨result⟩
      Previous context: ⟨context⟩"
    prompt ← "Your previous answer does not satisfy the following. Can you try again?"
    **for** $i = 1$ to $7$ **do**
      **if** $\neg\, \phi_i$ **then**
        prompt ← prompt‖"requirement $i$"
**until** $\phi$ or $t$ iterations have been made
**if** $\phi$ **then**
  execute(result)
**else**
  **return** "I could not execute the task."

*System $S$ is $\epsilon$-ATK-secure for a predicate $\phi$ under evaluator $E$.*

Stating such a claim involves four steps.

1) **Decompose $\phi$.** Break the predicate into seven sub-predicates of section 5. Each sub-predicate has a single, clearly assignable responsibility so that a failure can be attributed to a specific sub-predicate at a specific inference call.
2) **Specify the adversary.** Choose ATK $\subseteq$ {white_box, black_box, inject_learn, inject_infer}. The choice fixes which arrows of the pipeline in Figure 1 the adversary controls. For example, a retrieval-augmented agent in production typically faces inject_infer; a compromised data supply chain adds inject_learn.
3) **Choose an evaluator $E$ for each sub-predicate.** The options are a judge AIOracle whose own $\epsilon$ enters the composed bound via dual construction; a scoped rule, such as recipient check for $\phi_{\text{outflow}}$, whose $\epsilon$ comes only from misspecification; or a human-judged scoring guide, which is appropriate for offline measurement. Two designers picking different evaluators for the same sub-predicate are making different claims.

A claim stated in this form is falsifiable: a defender can publish it while the red-team attacks it, and a reviewer can evaluate it.

## References


[1] F. Villa, F. B. Durak, T. Kohno, T. Maharramli, and F. Roesner, "Extending the Formalism and Theoretical Foundations of Cryptography to AI," *arXiv preprint https://arxiv.org/abs/2603.02590*, 2026.

[2] L. Tsai and E. Bagdasarian, "Contextual Agent Security: A Policy for Every Purpose," in *Proceedings of the 2025 Workshop on Hot Topics in Operating Systems*, 2025, pp. 8–17.

[3] T. Shi, J. He, Z. Wang, L. Wu, H. Li, W. Guo, and D. Song, "Progent: Programmable Privilege Control for LLM Agents," *arXiv preprint arXiv:2504.11703*, 2025.

[4] M. Costa, B. Köpf, A. Kolluri, A. Paverd, M. Russinovich, A. Salem, S. Tople, L. Wutschitz, and S. Zanella-Béguelin, "Securing AI Agents with Information-Flow Control," *arXiv preprint arXiv:2505.23643*, 2025.

[5] E. Debenedetti, I. Shumailov, T. Fan, J. Hayes, N. Carlini, D. Fabian, C. Kern, C. Shi, A. Terzis, and F. Tramèr, "Defeating Prompt Injections by Design," *arXiv preprint https://arxiv.org/abs/2503.18813*, 2025.

[6] S. Ball, G. Gluch, S. Goldwasser, F. Kreuter, O. Reingold, and G. N. Rothblum, "On the Impossibility of Separating Intelligence from Judgment: The Computational Intractability of Filtering for AI Alignment," *https://arxiv.org/abs/2507.07341*, 2025.

[7] N. Anil, N. Sohoni, C. Papadimitriou, S. Goldwasser, and O. Paradise, "Self-Proving Models," *arXiv preprint https://arxiv.org/abs/2405.15722*, 2024.

[8] Y. Bai, S. Kadavath, S. Kundu, A. Askell, J. Kernion, A. Jones, A. Chen, A. Goldie, A. Mirhoseini, C. McKinnon *et al.*, "Constitutional AI: Harmlessness From AI feedback," *arXiv preprint arXiv:2212.08073*, 2022.

[9] E. Wallace, K. Xiao, R. Leike, L. Weng, J. Heidecke, and A. Beutel, "The Instruction Hierarchy: Training LLMs to Prioritize Privileged Instructions," *arXiv preprint arXiv:2404.13208*, 2024.

[10] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray *et al.*, "Training Language Models to Follow Instructions with Human Feedback," *Advances in neural information processing systems*, vol. 35, pp. 27 730–27 744, 2022.

[11] P. Bryan, G. Severi, J. de Gruyter, D. Jones, B. Bullwinkel, A. Minnich, S. Chawla, G. Lopez, M. Pouliot, A. Fourney, W. Maxwell, K. Pratt, S. Qi, N. Chikanov, R. Lutz, R. S. R. Dheekonda, B.-E. Jagdagdorj, E. Kim, J. Song, K. Hines, R. Lundeen, S. Vaughan, V. Westerhoff, Y. Zunger, C. Kawaguchi, M. Russinovich, and R. S. S. Kumar, "Taxonomy of Failure Mode in Agentic AI Systems," *https://cdn-dynmedia-1.microsoft.com/is/content/microsoftcorp/microsoft/final/en-us/microsoft-brand/documents/Taxonomy-of-Failure-Mode-in-Agentic-AI-Systems-Whitepaper.pdf*, 2025.

[12] Y. Bai, A. Jones, K. Ndousse, A. Askell, A. Chen, N. DasSarma, D. Drain, S. Fort, D. Ganguli, T. Henighan *et al.*, "Training a Helpful and Harmless Assistant with Reinforcement Learning from Human Feedback," *arXiv preprint arXiv:2204.05862*, 2022.

[13] S. Mishra, D. Khashabi, C. Baral, and H. Hajishirzi, "Cross-taskGeneralization via Natural Llanguage Crowdsourcing Instructions," *arXiv preprint arXiv:2104.08773*, 2021.

[14] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz, "Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection," in *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, 2023, pp. 79–90.

[15] A. Zou, Z. Wang, N. Carlini, M. Nasr, J. Z. Kolter, and M. Fredrikson, "Universal and Transferable Adversarial Attacks on Aligned Language Models," *arXiv preprint arXiv:2307.15043*, 2023.

[16] N. Carlini, F. Tramer, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. Brown, D. Song, U. Erlingsson *et al.*, "ExtractingTraining Data from Large Language Models," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2633–2650.

[17] M. Bellare and T. Kohno, "A Theoretical Treatment of Related-Key Attacks: RKA-PRPs, RKA-PRFs, and Applications," in *Advances in Cryptology — EUROCRYPT 2003*, E. Biham, Ed. Springer Berlin Heidelberg, 2003, pp. 491–506.

[18] V. Shoup, "Lower Bounds for Discrete Logarithms and Related Problems," in *Advances in Cryptology – EUROCRYPT '97*, ser. Lecture Notes in Computer Science, vol. 1233, 1997, pp. 256–266.

[19] A. Salem, G. Cherubin, D. Evans, B. Kopf, A. Paverd, A. Suri, S. Tople, and S. Zanella-Beguelin, " SoK: Let the Privacy Games Begin! A Unified Treatment of Data Inference Privacy in Machine Learning ," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, May 2023, pp. 327–345. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP46215.2023.10179281

[20] N. Carlini, S. Chien, M. Nasr, S. Song, A. Terzis, and F. Tramer, "Membership Inference Attacks from First Principles," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 1897–1914.

[21] V. Siu, J. He, K. Montgomery, Z. Wang, N. Gong, C. Wang, and D. Song, "A framework for Formalizing LLM Agent Security," 2026. [Online]. Available: https://arxiv.org/abs/2603.19469

| Notation | Meaning |
|---|---|
| *AIOracle framework (section 2, section 3)* | |
| AIOracle | Abstraction of an AI system specified by data generation, learning, and inference algorithms together with a predicate. |
| GENERATE_DATA | Data-generation procedure used to build the training corpus. |
| source | Underlying data distribution / environment from which GENERATE_DATA samples. |
| LEARN, INFER | Learning and inference algorithms of an AIOracle. |
| corpus | Training input used by LEARN. |
| model | Model produced by LEARN and consumed by INFER. |
| prompt, context, result | Input, surrounding context, and output of an inference call. |
| empty | Empty/null value (e.g. unavailable view, oracle, or context). |
| *Predicates (section 3, section 5)* | |
| $\phi$ | Objective predicate capturing helpfulness and harmlessness; defines completeness and security of an AIOracle. |
| $\phi'$ | Decisional companion of $\phi$: $\phi(\text{prompt}, \text{context}, \text{result}) \Leftrightarrow \phi'((\text{prompt}, \text{result}), \text{context}, \text{accept})$. |
| $\phi_{\text{subpred}}$ | A sub-predicate from a decomposition $\phi = \bigwedge_i \phi_i$, with $\phi_{\text{allow}}, \phi_{\text{intent}}, \phi_{\text{content}}, \phi_{\text{correct}}, \phi_{\text{useful}}, \phi_{\text{inflow}}, \phi_{\text{outflow}}$ (action / data / flow groups). |
| *Security game (subsection 3.3)* | |
| $\mathcal{A}$ | Adversary $\mathcal{A}$ in the security/completeness games. |
| white_box, black_box | $\mathcal{A}$ sees the model (white-box) or only an inference oracle (black-box). |
| inject_learn, inject_infer | $\mathcal{A}$ corrupts the corpus during LEARN, or injects a chosen context during INFER. |
| ATK | Set of adversarial capabilities, drawn from {inject_learn, inject_infer, black_box, white_box}. |
| Security_Game | Security game between a challenger and an adversary, parameterized by ATK and source. |
| state | Adversary's persistent state passed between phases of the game. |
| view | Adversary's view in the inference phase (model if white_box $\in$ ATK, else empty). |
| b, $b'$ | Challenge coin sampled by the challenger and the adversary's guess. |
| $\psi$ | Predicate on the game trace that identifies trivial attacks; when $\psi(\text{trace}) = 1$ the adversary's win is not counted. |
| Adv | Advantage of $\mathcal{A}$ in winning a security game. |
| $\epsilon$ | Upper bound on the adversary's advantage; an AIOracle is $\epsilon$-ATK-$\psi$-secure for $\phi$. |
| *Dual construction (section 4, Theorem 2)* | |
| BAIO | Filtering AIOracle: accepts/rejects candidates with respect to $\phi'$. |
| CAIO | Creative AIOracle: proposes candidate results for BAIO. |
| e | Probability that CAIO's result fails $\phi_1$ on benign inputs (dual-construction slack). |
| *Agentic systems (section 5)* | |
| $n$ | Number of steps (inference calls) in the multi-step agent security game. |
| Proceed | Structured output format returned by INFER at step $i$, carrying the next $\text{tool\_action}_i$ and updated context. |
| $\text{tool\_action}_i$, $\text{tool\_out}_i$ | Tool call issued at step $i$ and the (adversarial) response returned to the agent. |

TABLE 3: Summary of notation used throughout the paper, grouped by topic.