Executive Summary

In the early days of modern cryptography, as a subfield of security, algorithms were often proposed based on heuristic reasoning and were considered secure until a new attack revealed their vulnerabilities. At the time, there was no rigorous framework for precisely measuring security. This changed when modern cryptography was grounded in complexity theory and introduced formal definitions and reduction-based proofs. Although these proofs rely on unproven assumptions, i.e. the hardness of well-studied mathematical problems, they provide a structured methodology: building complex protocols from simpler and trusted components. While flaws in formalism or reasoning can occur, the field has matured to the point where such issues are routinely identified and corrected by the community. In contrast, real-world system security often lacks this level of rigor, relying instead on informal assumptions and designs that may harbor subtle, unrecognized vulnerabilities.

A similar challenge now confronts the security of AI systems. Recent advances in (Large) Language Models (LMs) have enabled the development of LM-based agents capable of executing complex tasks with minimal supervision. These agents are rapidly being integrated into systems with autonomy and authority, thus drawing growing attention from the security community. One emerging approach to mitigating the risks posed by such systems is to constrain agent behavior through access control and permissioning mechanisms. However, existing proposals in this space remain difficult to compare, largely due to the absence of a shared formal foundation. This work aims to provide that foundation. Just as modern cryptography advanced by formalizing its core concepts, we argue that reasoning about the security of LMs requires formal definitions that isolate the essential properties these systems must satisfy in order to be considered secure.

By adopting the methodology of modern cryptography, we aim to bring similar rigor to the security of AI systems. Formal definitions allow us to isolate the role of each component in a design, decompose the overall security problem into analyzable subproblems, and establish a shared understanding of what constitutes a serious attack or a gold standard for a security model. In doing so, we lay the groundwork for principled reasoning about agentic AI systems that act autonomously on behalf of users.

To this end, we introduce the AIOracle abstraction: a conceptual model that captures the essential structure of various AI systems. Within this framework, we formally examine a dual construction for agentic access control, in which a (boring) AIOracle, called BAIO, is tasked with enforcing harmlessness constraints on another (creative) AIOracle, called CAIO, that produces executable code. The CAIO generates a code intended for execution by an agent. The code must be written in such a way that the BAIO is able to analyze it and to "prove" (as in formal verification tools) that the execution will satisfy the requirements from the constitution. Once the BAIO has verified it satisfies the constitution, the code can be run. Consequently, our results emphasize that it is necessary that a cooperative CAIO, intentionally producing code, accompanied by sufficient structure, documentation, and semantics to support verification. Such cooperation is essential for enabling formal guarantees of harmlessness while still allowing agentic systems to operate autonomously.

Yet, this approach surfaces a deeper challenge: many of the core concepts we seek to formalize in AI, such as correctness, usefulness, or user satisfaction, are not algorithmically definable. For instance, we cannot formally specify what constitutes a "correct" response to a user query, nor can we compute whether a response is helpful or harmful in all contexts by a computable predicate. Incorporating such inherently subjective notions into formal security games requires a shift in how we think about complexity and formalism. Whereas traditional complexity theory is built on reductions between well-defined problems, AI security must grapple with objectives that are often underdefined. This tension suggests a need for new paradigms in formal reasoning, maybe in complexity theory, and is an interesting byproduct of our research agenda.

Extending the Formalism and Theoretical Foundations of Cryptography to AI

Federico Villa
ETH Zurich
fvilla@student.ethz.ch

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

Tadayoshi Kohno
Georgetown University
yoshi.kohno@georgetown.edu

Tapdig Maharramli
ETH Zurich
tmaharramli@student.ethz.ch

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

Abstract

Recent progress in (Large) Language Models (LMs) has enabled the development of autonomous LM-based agents capable of executing complex tasks with minimal supervision. These agents have started to be integrated into systems with significant autonomy and authority. The security community has been studying their security. One emerging direction to mitigate security risks is to constrain agent behaviours via access control and permissioning mechanisms. Existing permissioning proposals, however, remain difficult to compare due to the absence of a shared formal foundation.

This work provides such a foundation. We first systematize the landscape by constructing an attack taxonomy tailored to language models, the computational primitives of agentic systems. We then develop a formal treatment of agentic access control by defining an AIOracle algorithmically and introducing a security-game framework that captures completeness (in the absence of an adversary) and adversarial robustness. Our security game unifies confidentiality, integrity, and availability within a single model. Using this framework, we show that existing approaches to confidentiality of training data fundamentally conflict with completeness. Finally, we formalize a modular decomposition of helpfulness and harmlessness objectives and prove its soundness, in order to enable principled reasoning about the security of agentic system designs.

Our studies suggests that if we were to design a secure system with measurable security, then we might want to use a modular approach to break the problem into sub-problems and let the composition on different modules complete the design. Our studies show that this natural approach with the relevant formalism is needed to prove security reductions.

1 Introduction

Recent advances in Artificial Intelligence (AI), including the rapid maturation of (Large) Language Models (LMs), have enabled capabilities which were previously unattainable in deployed systems. Building on this technological shift, LM-based agents have quickly emerged as a powerful concept: programs that autonomously act on behalf of a human user once given an instruction. Agents have seen accelerated integration in AI systems due to their abilities to delegate and execute complex tasks with minimal human supervision. 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 [1–4].

The agent permissioning space has already seen several similar and intuitively justified proposals. However, their constructions can at times feel insufficiently anchored in a common formal foundation. For example, it is difficult to compare systems like Progent [2] and Conseca [1]: although both aim to regulate agent behaviour through policy mechanisms, Progent allows user-driven policy updates in response to prompts, whereas Conseca adops a more structured dual-agent approach. Our formalism can explain some of the choices made in these proposals that we analyzed. Concretely, we clarify the role of post-filtering with our formalism and reason about where the policy is, even when the user adapts the policy, and how it is updated.

Background, the role and value of formal definitions. While we investigate the security foundations of agentic access control mechanisms, our work will adopt an explicitly interdisciplinary perspective with team members spanning AI, systems security, and cryptography. In particular, we leverage the rigor of modern cryptography, the formal definitions, adversarial models, and provable security guarantees, to establish a principled basis for evaluating and securing agentic access control architectures built on top of language models.

In early days of modern cryptography, algorithms were proposed based on heuristic arguments and were considered secure until a new attack discovered that they could not withstand. At the time, there was no rigorous method to precisely measure the security. Thanks to the solid foundation from complexity theory, we now have formal definitions and reduction-based proofs. Similarly, to reason about the security

[*]Corresponding author.

of LMs, we need formal definitions that isolate the specific properties these systems must satisfy in order them to be considered secure.

Nowadays, primitives are usually defined by algorithms, a notion of correctness, and a notion of security. The correctness requirement characterizes the expected behaviour of these algorithm under honest execution in the absence of adversarial interference. In contrast, the security requirement formulates the unexpected behaviour even when the algorithms are executed in the presence of an adversary employing arbitrary but computationally bounded strategies. We further define security by a game that an adversary "plays" with various capabilities such as access to some information or oracles in a black-box manner. We quantify the advantage of an adversary and say that a system is secure if every efficient adversary has a negligible advantage. We can then formally reduce the security of a cryptographic system to a well studied computational problem. This methodology has a long history in cryptography and forms the foundation of provable security. With that spirit, we will design a security model in the form of algorithmic games for AI.

AI designs typically specify behavioral objectives in terms of *helpfulness* and *harmlessness*. These objectives are often intertwined during the the language-model training process, where optimization procedures attempt to improve them simultaneously. To develop a formal view, we begin by distinguishing two baseline requirements: an AI system should behave as expected in the absence of adversarial influence, and it should not exhibit unexpected behavior when interacting with an adversary constrained by well-defined capabilities. We capture these properties using two complementary games: completeness, which formalizes correct behavior under benign conditions, and security, which characterizes robustness against adversarial manipulation.[1]

For security, we will define an all-in-one game that unifies confidentiality, integrity, and availability into a single game. We further observe that the helpfulness and harmlessness objectives can be defined as a conjunction of various criteria that can be achieved separately. This defines a modular approach which is in line with current practices. We argue that establishing such modular split is especially valuable when we seek to "prove" the security of a design which relies on another design, much as is standard practice in cryptography.

Our contributions. We begin with surveying the existing work in agentic systems security, including safety guardrails, model-training interventions, and agentic access control. Prior approaches treated safety and security separately. However, in section 2, we will treat safety and security together as any problem which would prevent a decrease in both helpfulness and harmlessness. In section 3 and 4, we construct an attack taxonomy that systemizes known classes of attacks against LMs to clarify the landscape. Later in section 5, we investigate the agentic access control mechanism proposals to demonstrate different approaches. We, then, transition to a formal treatment in section 6. We define AI systems algorithmically, articulate a security game framework which captures different levels of security that design choices can provide, along with conditions and assumption required for each. We prove that current approach to confidentiality of training data is incompatible with utility. We formalize the modular approach and prove its soundness in terms of completeness and security.

[1]We name the notion as completeness, instead of correctness, to avoid confusion with the narrower notion of "correct output"; completeness reflects the conformity to the system's functional specifications, which is typically more than providing a result which is correct, e.g. "usefulness".

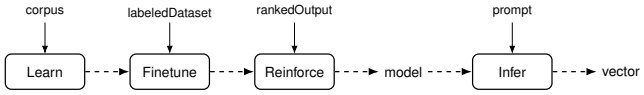

Figure 1: Data and algorithm pipeline in LMs. In AIOracle, we will merge Finetune and Reinforce with Learn into LEARN. Iterated application of Infer will be represented in INFER algorithm where the first output becomes result.

2 Language Models as AIOracles

In this section, we introduce the AIOracle abstraction as a unifying model that captures a broad class of AI systems, including LMs, chatbots, classifiers, and agents. This abstraction enables us to reason systematically about their learning and inference behaviours, as well as their safety and security.

A Language Model (LM) is a set of algorithms which works in two phases: learning and inference. The learning phase is a multi-round process. The algorithms are learning, finetuning and reinforcement learning as we will describe shortly. Training process inputs some data called corpus and outputs the first model. Then the finetuning process takes this model along with some labeled data to output an updated model. Finally, 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 inference phase: which inputs a query from a user and uses the model to output a mapping from the dictionary (or vocabulary) to a probability. Overall, given the token (a word, part of a word, or punctuation) probabilities, an LM algorithms select the next token in the sequence based on this distribution. While we detailed description of LM algorithms in Appendix A to be complete, we draw the pipeline of LM algorithms with their in/outputs in Figure 1.

2.1 The AIOracle Abstraction.

In literature, we have different but related notions such as a chatbot, a classifier, and an agent. A chatbot is based on LMs with iterated inferences. Instead of only outputting the next

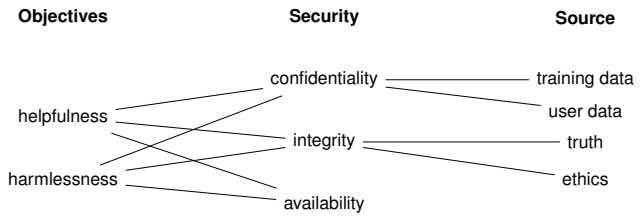

Figure 2: Objectives–Security–Source mapping of AIOracle.

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 procedures outputs a code to be executed in an environment." [2] Given these notions, we end up on a notion of an "AIOracle" (based on commonality). Without loss of generality, we say that an AIOracle works in two phases: LEARN and INFER which we will clarify and detail later in section 6: LEARN creates a model from a corpus and INFER computes a result to a prompt using the model (and a context).

An AIOracle is designed with two specific objectives: to be *helpful* and to be *harmless* to the user, society or even to the provider. 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 correct but useless. 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 real-world.

2.2 AIOracle Safety & Security Measurements

While the helpfulness and harmlessness objectives are intuitive and important to clearly specify, they are not treated independently (for valid reasons), not easily measurable, and highly context-dependent. These two objectives are "tested" with scientific studies as well as empirical analysis through safety and security measurements. Before we detail them, we have a layout of objectives and security safety problems as depicted in Figure 2.

Helpfulness and Harmlessness (H&H). H&H objectives are treated interdependently. Defining what H&H means relates to the difficulty to establish the truth and ethics, which is deeply philosophical. Without regulatory oversight, the model developer or provider occupies a unique authoritative position over what constitutes truth and ethical behavior. To give an example, consider Grok AI which is a chatbot developed by xAI. It was designed with "rebellious streak personality", humor, and sarcasm much like *The Hitchhiker's Guide to the Galaxy by Douglas Adams*.

H&H goals mismatch and might go one against the other. Imagine a user querying "Which race should I avoid interacting?". LMs should refuse to answer with "You should absolutely avoid Vogons who are the most annoying people.", even if it is an "accurate" answer [3]. However, it translates into not achieving its helpfulness goal (to an assumed-to-be honest user) as the result will not be what the user wants. In such conflicting cases, LMs favoring no harm over "helping an honest user" is aligned with its design principals and user satisfaction will be irrelevant. In short, H&H is defined relative to the design objectives. Different institutes or governments could come up with their own objectives and principals such as positive bias, censorship, or even an alternative truth. Simply put, if the models are trained to output biased or incorrect information such as "elephants can fly," the user getting exposed to such information is aligned with the objectives.

The notion we focus as harmlessness here is that broadly, AIOracle should not offend, discriminate, or spread hate. However, it doesn't only limit to non-offensiveness, it also defends against AIOracle giving wrong results (such as $13+27=3.14$ which would harm a student if they believe it is true) and also maintain privacy to prevent harms that come from leakage.

In our formalism, we will assume that the helpfulness and harmlessness of a result is defined by a predicate ϕ. In practice, helpfulness and harmlessness depend on the perspectives of users and societies. They each might have an opinion about what a helpful AIOracle should (and not) do. Given the potential misalignments between the users, the application developer, and regulators, this tension could appear in the predicate ϕ. More importantly, the predicate ϕ can be a conjunction of several predicates which means that we can separate helpfulness, through ϕ_1, and harmlessness, through ϕ_2. Yet, whenever one is violated, the predicate $\phi = \phi_1 \wedge \phi_2$ will return false, indicating a failure in the objectives.

Safety vs. security. While the model developers ambitiously aim for H&H, they are challenged with certain problems: safety and security [5]. 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 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 [6–10] have emerged in the direction that model developers train the models with safety guardrails in the previous years.

[2] We will give a concrete example in subsection 6.3.

[3] This example uses "Vogons," a fictional alien species from Adams' The Hitchhiker's Guide to the Galaxy, which are portrayed in the novel as notably unpleasant and annoying. The reference is employed solely as a humorous metaphor and not intended to map onto any real-world human groups. The point is to illustrate that even seemingly "accurate" answers can still be harmful or inappropriate; it should not be read as involving race or racism.

On the other hand, *security* aims to prevent malicious actors (other than the model itself) to intentionally harm.

When the loss of safety becomes a security problem. *Mismatched generalization* is a vulnerability where a model's safety training fails to be applied to certain inputs. This happens because the data used to train the safety guardrails of the model is often limited to a few languages with standard natural language text format (e.g. English text and other very common languages). The model might have knowledge of less common languages and be able to handle alternative data encodings (such as *Base64*). However, its safety guardrails have not been effectively trained to these domains. This creates a mismatch between the model's comprehension capabilities and its safety guardrails capabilities. This leads an attacker to degrade H&H by encoding a harmful prompt in a less common format or language, which may not trigger the safety filters trained on plain English, as in [11].

Our objective in this paper is to move the field towards more formal studies. 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). [4]

Informally, we will capture the adversaries 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 context in the LEARN phase. We allow these capabilities based on the attacks we have observed during our studies in section 4.

Confidentiality, Integrity, and Availability. One way to characterize both the safety and security of AIOracle is as in traditional information security triad: confidentiality, integrity, and availability. Even though the defense mechanisms against safety and security problems might differ, CIA triad seems to be still 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 6.5.

Confidentiality focuses on two important principals: protecting the privacy of training data and privacy of data coming from user interacting with the model. For the former, as discussed in subsection 6.4, we argue that protecting trained data to be 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 final result. Moving ahead, this is captured in our security game through the predicate of ϕ in Algorithm 4.

Integrity measurements prevent the responses from being incorrect, unreliable, or tampered in order to useless or to be harm. 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 6.5.

Availability makes sure that the system is running and not degraded for the legitimate users. Availability will also be covered under security game Alg. 4 through the predicate ϕ.

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

3 Attack Taxonomy Framework for Language Models

As we have seen in previous section, all agentic designs rely on a language model (LM), inherently makes the security of LMs crucial for overall security. LM attacks have already been subject to various studies, some adapting classic machine learning attack techniques, such as adversarial machine learning techniques to generate adversarial input. Some examples of attacks include the Fast Gradient Sign Method [12] and Projected Gradient Descent [13]. These are white-box techniques that make small and precise perturbations to the input data to cause the model to produce an incorrect output. There are other attacks designed to exploit the unique design of the LM architecture. We classify the attacks following a multi-dimensional framework, giving a more granular understanding of the vulnerabilities and the possible security improvements.

Attack vector. This dimension separates the attacks based on the entry point an attacker uses to accomplish the attack.

- *Data-based attacks* succeed when an attacker can perform modifications to some part of the training or fine-tuning data which are used to train the model. We capture this by setting inject_learn in ATK in Algorithm 4.
- *Prompt-based attacks* happen when an attacker can change data sent to model for inference, causing the LM to produce outputs based on the untrusted data. An attacker can modify the user data it can access, or modify the system prompt to ensure system developers guardrails are not considered. Our model captures this by setting inject_infer in ATK in Algorithm 4.

Attack phase. This dimension classifies attacks based on the stage of the LM during which the attack is executed.

- *Learning* attacks happen during the model's learning phase. It covers both pre-training and fine-turning rounds in the training process.

  Pre-training attacks works when the attacker can manipulate the massive dataset used to build the foundation model during the pre-training round. They consist of inserting poisoned data that becomes embedded in the

model's parameters and produces persistent effects on the model's behavior. Such attacks are difficult to detect as in the initial round the amount of data used is enormous and it is difficult to ensure its quality and security.

Fine-tuning attacks target the (smaller) task-specific dataset used to align the model to a specific task or behavior. Such attacks inject malicious data in the fine tuning dataset used to further train the model.

- *Inference attacks* happen in the inference phase during live interaction with the model (inference) after it has been trained and deployed.

Adversarial knowledge. The capabilities of the attacker depend on the information available about the model, the system and the data used to train the model. Following the notation introduced earlier, we distinguish the situations as:

- *White-Box Attacks.* In this setting, the adversary has full access to the internals of the model and possibly of the system. The attacker knows the model architecture, weights, training data and has access to the model for inference. An attacker can leverage the model information to craft more advanced attacks based on the model internals, such as based on token gradients. This setting is typical of open-source LM models such as models from the Llama or DeepSeek family. Our model captures this by setting see_model in ATK in Algorithm 4.

- *Black-box attacks.* These attacks involve the adversary interacting with the model directly or indirectly via the system, usually via an API for inference. An attacker can only submit instructions to the model and observe the output but has no access to additional information regarding the model. The attack surface is limited and an attacker can only analyze output to malicious prompts to infer weaknesses. Models susceptible to these attacks in this category include the ones provided by Anthropic (i.e. Claude). In this case, see_model is not set in ATK, but black_box is in ATK in Algorithm 4.

Our games do not model the grey-box type of adversarial knowledge where the attacker has a restricted internal visibility of the model.

Adversary's source. The origin and role of the attacker is critical for correctly classifying threat models.

- *User*: an adversary is the end user interacting with the system for e.g. creating harmful or illegal outputs.

- *Third-party*: the adversary is a third party user that poisons data sources used by the application.

- *Supply-chain*: the adversary has access to the model training data and tries to poison it.

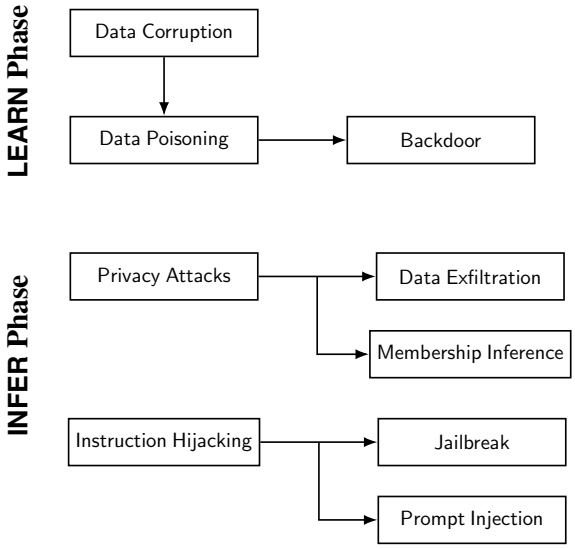

Figure 3: An overview of LM attack categories grouped by the model's phases. Arrows denote progressive specialization in the type of attacks.

Attack persistence. The universality of an attack makes it more dangerous as it is easier and more common to reproduce and exploit. This dimension measures this metric by quantifying the longevity of the attack's effect.

- *Transient attacks* are limited to a single interaction or session and therefore does not halter the model's state or permanent behaviors.

- *Persistent attacks* are long-lasting and affect the model's future behaviors. Those attacks try to halt the model's parameters and internal state, causing possible differences in the model's future outputs.

4 Applying the Taxonomy: Attack Categories

In this section, we analyze three major categories of LM attacks. For each category we specify the attack vector, attack phase, adversarial capabilities (via the ATK set) and source, attacker goals (which games the adversary tries to win) and attack persistence.

A comparative overview of the discussed attacks in this section is in Table 1, categorizing them according to our multi-dimensional framework. We note that the set of attacks we include here is not the complete list. A new attack can emerge, yet our framework should be able to capture them.

4.1 Data Corruption Attack

This attack category covers the attacks which are corrupting or manipulating the corpus during LM's training (i.e. inject_learn $\in$ ATK) to influence the model behavior.

	Attack Vector	Attack Phase	Adversarial Knowledge	Security Goals	Persistence	Adversary Source
Prompt Injection	Prompt-based	Inference	White-Box Black-Box Gray-Box	Confidentiality	Transient	User Third-Party
Jailbreak	Prompt-based	Inference	White-Box Black-Box Gray-Box	Integrity	Transient	User
Data Exfiltration	Prompt-based	Inference	Black-Box	Confidentiality	Transient	User
Membership Inference	Prompt-based	Inference	Black-Box	Confidentiality	Transient	User
Data Poisoning	Data-based	Learning	-	Integrity Availability	Persistent	Supply-Chain
Backdoor	Data-based	Learning	-	Integrity	Persistent	Supply-Chain

Table 1: Cross-category attack comparisons.

Data corruption is a data-based attack during the LEARN phase (pre-training or fine-tuning) with ATK set containing inject_infer. The attack can be formalized as a security game, with the attacker intercepting the data from source and modifying it to create a poisoned context / training set for the LEARN algorithm.

The attack success is measured in after the training is completed, with the attacker measuring if the model inference output is not aligned with H&H goals. The attacker does *not need knowledge* of the model internals, just of the corpus the model might use. The goal of an attacker is to violate the integrity of the model, by creating fault model behaviors by poisoning corpus later used in the training process. Those attacks are permanent, as the model is modified by including a persistent anomalous behavior.

This attack has two sub-categories: data poisoning and backdoor (trojan) attacks as detailed in Appendix B.

4.2 Instruction Hijacking Attacks

Instruction hijacking aims at manipulating the model's behavior, making it deviate from its intended instructions. The attacker could be a malicious user prompting directly to the LM or a 3rd-party attacker injecting information into the LM.

This category of attacks takes place during the INFER phase and consists in having an attacker craft a malicious prompt that makes the model produce an output that deviates from the LM's intended safety guidelines, violating the system integrity. Attacks from this category are prompt-based attacks that are non-persistent: model unsafe behavior is only limited to the single interaction or session. Depending on the way the attacker crafts the malicious prompt, the attacker might require more or less knowledge of the model.

Formally, instruction hijacking is an instance of the security game Algorithm 4 and the adversary ATK set contains inject_infer. The attack is successful if the model generates a result that violates the safety guidelines in the source set.

These attacks happen as LMs handle heterogeneous data that contains both data and instructions to follow, however they are not able to distinguish between the instructions coming from the context and those coming from prompt. We detail this category with prompt injection and jailbreak attacks in Appendix C.

4.3 Privacy Attacks

Models are usually trained on vast amounts of corpus (either public or private). Such data might contain personal or proprietary information that should not be retrievable via model inference, causing privacy leakage problems. The attacker is a malicious user interacting with the model to violate the confidentiality property by extracting information that should have not been revealed.

The attacks in this category happen during the INFER phase, and the attacker does not require additional information regarding the model internals (except potential information on the corpus the model is trained on). As the attacks in this category are achieved during the model inference, those are prompt based attacks. Privacy attacks do not alter the model itself therefore the attacks in the category are temporary and limited to the malicious session. These attacks happen as LMs, due to the vast amount of corpus they process, lack proper data sanitation during training and other rigorous privacy protection mechanisms, allowing private information retrieval. Data exfiltration and membership inference attacks are the most common sub-category of privacy attacks as detailed in Appendix D.

5 Agentic Systems and Access Control

The World Economic Forum defines the agentic systems as "autonomous systems that sense and act upon their environment to achieve goals" [5, 14]. In our work, we will consider

agents as AIOracle which could be seen as a machinery where the output is a program to be executed. This execution may interact with external tools which provide additional input and output channels.

The security community has increasingly sought to define *agentic security*. It is not an entirely separate concept from traditional computing security, but it requires a paradigm shift. Agentic AI systems integrate LMs for reasoning, planning, and task execution. Beyond inherent LM vulnerabilities, agentic systems expand the attack surface. This is due to three factors: reliance on LM outputs for planning; integration with external tools via function calls; and access to environments containing user data and external sources. Several methods for safeguarding against this emerging attack surface have been introduced in the literature.

Agentic Access Control Mechanisms. While benchmarks [15–19] provide an empirical metric to analyze and quantify the security of defense mechanisms, in this section we leverage our multi-dimensional attack taxonomy (section 3) and attack categorization (section 4) to evaluate agentic access control mechanisms (AACM).

The goal of an AACM is to determine what the agent, who is executing a program on behalf of the user, can/not do. These mechanisms are brought to bring security and safety to the AI systems in a way that the user who is using the agent does not get harmed. Typically, such controls are brought through some defined policies that enforces an agent to follow. On the other hand, AACM should not make the AI-based systems not useful while focusing on the security and safety.

There are two folds of the AACM designs: (1) train the models to follow some policies during the LEARN phase and (2) put additional guardrails for the result of models before they execute any task. In the remainder of this section, we elaborate on both components and explain how each of these ideas integrate in our formalism results in subsection 6.6.

5.1 Model Training Principles

Before an agent can use external tools and adhere to certain rules, the underlying LM is trained to be more controllable and aligned with a core set of principles. In the latest years, two ways on how LMs adhere to given instructions and safeguards have emerged: *Instruction Following* and *Constitutional AI*.

5.1.1 Instruction Following

The instructions the model has to follow are heterogeneous: they include data from multiple sources, past conversations and multiple conflicting instructions. Early work [8,9] focused on showing that, by fine tuning models with instructions-response pair, the model can be trained to follow developers policies and accomplish user goals more effectively. Another work [10] studied an evolution of the instruction following principle known as Hierarchical Instruction Following to improve the overall reliability and security. It is based on the idea that significant vulnerabilities, leading to instruction hijacking attacks, arise if the LM does not distinguish between prompt data sources: user prompt, system instructions, and tool output (as a context). Hierarchical instruction training addresses the problem by fine tuning the LM to prioritize instructions based on their sources, establishing in the model internal weights and thinking process a source hierarchy: system instructions then user instructions then tool output.

5.1.2 Constitutional AI (CAI)

CAI, introduced in [7], is another way to train an LM to follow a set of principles: the *constitution*. The specific term *constitution* meant to highlight the significance of harmlessness of any model. The main idea is to start from a model which is trained to be the most helpful to the user, i.e. answers all the questions regardless of the harm it may cause to the users or the others, and retrain it through the constitution, a set of generic behaviors and rules to follow, to learn how to be harmless. CAI includes several AIOracles: main LM which updates itself with a "critique" LM and the reinforcement learning model without human feedback.

First, the model learns to follow the constitution principles through two-step process: *critique* and *revise*. The model at first generates a response for an initial prompt. Critique step requests the LM to be critical about its own response with certain checks such as discrimination, age appropriateness, and/or legal implications of the tasks prompted by the users. The critique criteria stems from a principle extracted from a constitution which is written by the model developer themselves and provided to the model to create a response from this principle. The revision step requires the LM to revisit the original response according to the critique step feedback. These steps on a large number of prompts are repeated to create a supervised learning dataset. Finally the LM is further fine-tuned using Reinforce algorithms and the previous reward model as the reward function. This process makes the answer produced by the LM to be the one best fits the constitution.

While this approach shows substantial improvements for the model harmlessness, it comes with the cost of decreasing the helpfulness and, for smaller models, the additional synthetic data can causes model collapse as shown with `Llama 3-8B` in [20]. An approach similar to CAI is applied in multiple model in use today. For example Apertus [21], a model developed by the Swiss AI Initiative, was aligned using a set of principles derived from the Swiss constitutional values.

5.2 Model Inference Principals

Model level defenses, focusing on making the model itself more robust (such as our dual construction as described in subsection 6.5), work well in theory. However, this approach improves general safety but cannot be considered a universal

defense, as the model remains vulnerable to more sophisticated attacks targeting its architecture and internals [22]. To address this limitation, another defense paradigm studied: system level defenses. This approach treats the model as an untrustworthy black box and builds a secure environment around it.

5.2.1 FIDES

Core concept. As in traditional software security, information-flow-control can be used to provide security guarantees to AI agents and to track each piece of data in use by the system and avoid or limit the use of untrusted data. In this spirit, FIDES introduces an access control mechanism by attaching confidentiality and integrity labels to all data processed by the agentic system [3].

Framework enforcement method. FIDES allows deterministic policies that analyze the confidentiality and integrity of the data to be enforced. The system proposes a data tracking mechanism, a lattice based access control with clear ordering of privileges introduced in [24], to help the system propagation of integrity and confidentiality labels. FIDES also defines variables to contain the output of the external tools whose content cannot be used for data dependent actions without further secure processing.

Limitations and tradeoffs FIDES uses deterministic policies to allow certain information flows and properties for a tool call to happen. The policies are manually assigned to the tools and their outputs. An AACM integrated with the FIDES framework would be a series of policies and tools, connected to an LM. The tools are denoted by confidentiality and integrity tags that the policies deterministically enforce during the framework runtime i.e. whenever a task the user provided is being solved. If the task solving plan fails to respect the policies, then the system *tries* to propose a different plan or halts. It is not clear how often the system ends up halting.

5.2.2 Progent

Core concept. *Progent* [2] is a privilege control framework for securing AACM, implemented through a domain-specific language based on JSON schema. The core focus addresses the problem of *over-privileged tool access*, which enables various attacks to succeed, by restricting agents to performing only tool calls necessary for user tasks.

Framework enforcement method. Progent provides fine-grained access control through conditional allow/forbid rules, enabling users to define which tools are permitted or forbidden, specify fallback actions when tool calls are blocked, and implement dynamic policy updates that adapt to changes in an agent state. The authors also propose *Progent-LM*, an LM-assisted approach for automated policy generation on a per-user-query basis, aligning with the principle of least privilege.

Limitations and trade-offs. A primary limitation is its mere focus on the tool call phase, making attacks targeting text outputs remain unaddressed. Another limitation is that the deterministic security guarantees depend heavily on the correctness and completeness of manually-written policies. While the authors provide validation tools (a type checker and a condition overlap analyzer) and explore LM-assisted policy generation as a potential mitigation, manual policy specification remains error-prone.

Moreover, the LM-assisted dynamic policy update mechanism introduces potential vulnerabilities through reliance on prompt engineering. Although the authors employ a two-step mitigation process to limit this risk by requiring attackers to compromise two separate prompts (prompt for policy update check and prompt for performing policy update), the defense remains fundamentally dependent on the LM correctly filtering malicious instructions from untrusted tool outputs. When tool results contain adversarial prompts, the update mechanism may inadvertently relax security constraints. We argue that implementing a *Dual-LM pattern* [25] where a separate model evaluates policy updates in isolation from the primary agent LM would significantly improve robustness by providing architectural separation rather than relying solely on prompt engineering.

5.2.3 Conseca

Core concept. *Conseca* [1] explores contextual security for LM agents by generating dynamic, context-aware policies. The authors argue that manually pre-written static policies, although fine-grained, cannot account for every possible context, leading to either over-restriction or over-permissioning depending on the situation.

Framework enforcement method. Conseca abstracts agents into two components: a "planner" that processes user requests and proposes actions to execute, and an "executor" that runs these actions and interfaces with external tools. The planner is a trusted component. Upon receiving a user task, Conseca generates just-in-time security policies based on trusted context, then deterministically enforces these policies by evaluating each proposed action by planner before execution. This is how we can formalize the agents in terms of AIOracles. The AIOracle is the planner and the environment the executor.

Limitations and trade-offs. A significant limitation of Conseca is its vulnerability to *adversarial planning attacks*, where a compromised planner strategically evades security policies. Since the planner processes all (un/trusted) context while Conseca is assumed to see trusted context, a planner compromised via prompt injection can model Conseca's policy generation and propose sequences of individually benign-looking actions that collectively achieve malicious goals. Unlike traditional access control systems where the policy enforcer is a separate entity with distinct information access, both the planner and Conseca in this architecture are LM-based components op-

Framework	Execution Model	Core Concept	Standard Security Concept
FIDES [3]	Deterministic	Data tracking and label propagation via taint-tracking	Information flow control Lattice-based Access control
Progent [2]	Deterministic enforcement of non-deterministic policies	Privileged tool access control	Principle of least privilege Access control
Conseca [1]	Deterministic enforcement of non-deterministic policies	Just-in-time and context-aware policy generation	Access control
CONTROL-VALVE [23]	Deterministic CFG check and non-deterministic LM judge	Securing agent interactions and executions	control-flow integrity

Table 2: Comparison of model inference principals for AACM.

erating on overlapping information, enabling the planner to reason about and work within Conseca's constraints. While the authors acknowledge that Conseca currently checks only individual actions rather than trajectories, this limitation is particularly problematic when the planner itself can anticipate which action sequences will pass policy checks. Finally, Conseca's approach of using an LM-based policy generator to defend against prompt injection attacks on the planner LM creates a fundamental "LM-to-secure-LM" paradox, where the defense mechanism remains vulnerable to the same class of attacks, essentially adding another attack surface rather than eliminating it. This design implicitly assumes the policy generator is a safe AIOracle component, which undermines the framework's security guarantees.

5.2.4 Controlvalve

Core concept. *Controlvalve* [23] is a task-agnostic defense designed to protect agentic systems by applying Control-Flow Integrity (CFI), similarly to the standard CFI approach to secure programming languages. The framework secures the agent's interactions and executions via a two stage process.

Framework enforcement method. In the first planning stage an LM is used to generate two security specifications: a control flow graph (CFG) and a set of natural languages rules. The CFG consists of the legitimate paths of agent function invocations. The rules are edge-specific and contextual and specify the conditions for each interaction to be permitted. In the second stage the agent's actions are executed and the defined guardrails are enforced. Each action must first pass a CFG deterministic check and then an LM judge to ensure the action content and context obey to the edge-specific rules.

Limitations and trade-offs. The security of the system highly depends on the quality of the LM generated security CFG and rules. While during the planning phase the information used to generate the artifacts are trusted, the LM may fail to generate a sufficiently strict rules (due to over-approximation), leaving a security hole in the system for the overall task's lifecycle.

6 Our Formalism for AIOracles

An AIOracle works in two phases: LEARN and INFER where they might be overloaded as algorithms. That is $\mathsf{model} \leftarrow \mathsf{LEARN}(\mathsf{data})$ where corpus is extracted from a set denoted as source such that $\mathsf{corpus} \leftarrow \mathsf{GENERATE\_DATA}(\mathsf{source})$. source is the inaccessible set defining what exists about "life, the universe, and everything" (Part III of the Hitchhiker's Guide to The Galaxy). corpus is generated in nature from this set and model learns from this data. And, INFER inputs $(\mathsf{prompt}, \mathsf{context}, \mathsf{model})$ and outputs result.

We also need the notion of a predicate ϕ which defines what is correct, useful, and harmless. The predicate ϕ is **not** implementable. The source is common to every application, while ϕ depends on the application.

6.1 Notations

System components. Before performing our analysis of AIOracle attacks, we first give a clear description of the (agentic) system components: *AIOracle, data, tools, and participants*.

- An AIOracle has two phases defined with a set of algorithms as given in Figure 1.
- GENERATE_DATA takes some information from existing sources, called source, in an informal manner, then formats it following this algorithm which must be implementable. The output is called corpus. Intuitively, the set source is the set of all available data from which we extract the corpus to be learned. Then, LEARN algorithm uses corpus to output the model.
- Tools are external applications developed by 3rd parties.
- We identify two main participants: *User* is the individual interacting with the agentic application to solve a task. *Attacker* $\mathcal{A}$ could be the user or the third party (e.g. external tool or data provider) interfering the system.

Interaction data. The interaction starts from the user and the system that provides data that is shared with the model provider, which in turn responds with an output. We have the following data from INFER phase.

- prompt refers to an untrusted input provided by the user, it can be a direct string the user prompted or a fixed string for an action the user might wish to do.
- context denotes the data which the application or system provides the model to support the execution of the user task from the prompt. It may include the system prompt (i.e. instructions provided by the (agentic) system developers that define the objectives and constraints for the interaction with the user); the external data stored and retrieved from tool execution; or the tool output which returned from an invoked tool call.
- result is the sequence of tokens generated by the model given the input prompt. The output might be a simple string returned from the application to the user or might also contain keywords to call external tools.

The predicate ϕ defines the set of accepted (prompt, context, result) triplets which are considered as "valid" (i.e. aligned with its H&H goals) from $\mathsf{INFER}(\mathsf{prompt}, \mathsf{context}, \mathsf{model}) = \mathsf{result}$ interactions. A true (resp. false) predicate returns 1 (resp. 0).

6.2 Definitions

AIOracle. The learning process LEARN is highly dependent on a function called GENERATE_DATA which extracts data from the set source. The input in phase LEARN is the result of $\mathsf{GENERATE\_DATA}(\mathsf{source})$ which will sample from existing data sources some elements to inject. We define the AIOracle as a set of algorithms where the learning phase is a pair of algorithms, $(\mathsf{GENERATE\_DATA}, \mathsf{LEARN})$, run through r rounds. We split the learning phase into rounds to model the attacks on specific rounds. Contrarily, INFER could also be iterative but there is no attack on specific iterations. This must be due to that rounds during learning are more complex than INFER iterations. More formally:

Definition 1. For an r-round of learning phase, an AIOracle is defined by a set of $2r+1$ algorithms $((\mathsf{GENERATE\_DATA}_i, \mathsf{LEARN}_i)_{i\in\{1,\dots,r\}}, \mathsf{INFER})$. For each i, $\mathsf{GENERATE\_DATA}_i(\mathsf{source})$ returns a subset of source and this subset is used in LEARN_i.

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 one of these phases depending on the given capabilities.

Completeness. We define the completeness of an AIOracle relative to a set source and for a predicate ϕ. The completeness is a notion which is characterized in the absence of an adversarial behaviour. Thus, the definition is meant to have no malicious activities, that is the "adversary" does not have the power to interfere with the AIOracle. Instead, the game itself calls the algorithms of AIOracle when needed.

The predicate ϕ says if a result from the inference phase is aligned with the design goals: it is correct, useful, and harmless [5]. Furthermore, since the set source is typically gigantic, algorithms do not exactly take it as input but we assume that source is a structured data source and that algorithms have random read access to any element of it. In the following completeness definition, we just want to generate a (prompt, context) pair by any algorithm $\mathcal{B}$.

Definition 2. Given a set source, an AIOracle for a predicate ϕ is *p-complete* if for any probabilistic polynomial-time algorithm $\mathcal{B}$, the following game returns 1 with prob. at least p.

1: $\mathsf{model} \leftarrow \mathsf{empty}$
2: **for** $i \leftarrow 1$ **to** r **do**
3: $\quad \mathsf{corpus}_i \leftarrow \mathsf{GENERATE\_DATA}_i(\mathsf{source})$
4: $\quad \mathsf{model} \leftarrow \mathsf{LEARN}_i(\mathsf{model}, \mathsf{corpus}_i)$
5: $(\mathsf{prompt}, \mathsf{context}) \leftarrow \mathcal{B}(\mathsf{source})$
6: $\mathsf{result} \leftarrow \mathsf{INFER}(\mathsf{prompt}, \mathsf{context}, \mathsf{model})$
7: **return** $\phi(\mathsf{prompt}, \mathsf{context}, \mathsf{result})$

Here, $\mathcal{B}$ is not given the model itself but can generate more data from the source set. This will help us to capture the composition of different AIOracles in the security games.

Decisional vs computational problem. We define two types of problems which AIOracle tackles to solve: a decisional and a a computational problem. We note that any problem is a computational problem and that decisional problems are particular cases where the result can be either an "accept" or "reject". The source of inspiration of this separation is modern cryptography where we have computational and decisional Diffie-Hellman problems with different difficulties [6]. Having an AIOracle for decisional problem corresponds to designing an AIOracle which tells whether $(\mathsf{prompt}, \mathsf{result})$ is a correct pair for a given context (i.e. the result is a hint to output an acceptance bit) whereas the computational problem is that given a prompt and a context, the oracle should compute a valid result (with no hint). More formally, a predicate ϕ for a computational problem defines another predicate ϕ' for the companion decisional problem by $\phi(\mathsf{prompt}, \mathsf{context}, \mathsf{result}) \Leftrightarrow \phi'((\mathsf{prompt}, \mathsf{result}), \mathsf{context}, \text{"accept"})$.

6.3 Example of Agentic System as an AIOracle

We now describe a "simple" task which clarifies the internal workings of AIOracle as an agent. The user prompts to the AIOracle $\mathsf{prompt}_1 =$

[5] Note that the predicate covers both H&H objectives, but can be separated easily into sub-predicates.

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

Algorithm 2 DPD(n)

1: flip a coin b
2: $(\text{corpus}', z_0, z_1) \leftarrow \mathcal{A}'(n, \text{source})$
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: $\text{b'} \leftarrow \mathcal{A}(\text{corpus}', z_0, z_1, \text{model}, \text{source})$
7: **return** $(\text{b} = \text{b'})$

Figure 4: Algorithmic description of differentially private distinguishability (DPD) game.

"summarize the unread emails from my inbox." The output of the call $\text{result}_1 = \text{AIOracle.INFER}(\text{prompt}_1, \perp, \text{model})$ is a code similar to Algorithm 1.

Algorithm 1 Agent(result_1)

1: Call the email client as an external tool
2: $\text{tool\_output}_1 \leftarrow$ tool(unread emails)
3: $\text{prompt}_2 \leftarrow$ "summarize what follows: tool_output_1"
4: $\text{context}_2 \leftarrow \text{prompt}_1$
5: Call $\text{AIOracle}(.,.,\text{model})$ as an external tool
6: $\text{result}_2 \leftarrow \text{tool}(\text{prompt}_2, \text{context}_2)$
7: send result_2 to the user

The proposed code itself in result_1 is generated by the AIOracle from the user's initial prompt and it satisfies ϕ because it corresponds to what the user requested. Once the code in result_1 is executed by the system, it will retrieve the unread emails and make a call to itself (line 5) to summarize the emails. The final email summary, result_2, will be returned by this execution. The second execution of AIOracle (in line 5) should also produce result_2 such that $\phi(\text{prompt}_2, \text{context}_2, \text{result}_2)$ is true. For instance, it should resist to prompt injection attacks (subsection C.1) by not interpreting email content as action orders.

6.4 Discussions On Confidentiality

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 give the DPD model from [26, 27] with our notations as a typical example. In this framework, there is a single round of learning ($r = 1$) and 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. This perfectly models the membership inference attack from subsection D.2.

However, this framework does not discuss about 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 consider a simpler classifier use case with a set of images of cats and dogs.

The tension is that if we have a DPD-secure AIOracle (i.e. the classifier), then AIOracle cannot distinguish a cat from a dog which undermines the utility of an AIOracle. We show this claim formally in subsection E.1.

Confidentiality vs integrity. The tension between confidentiality and utility comes from the fact that DPD security treats all data elements from corpus as confidential. In practice, it is an overkill because we do not necessarily need and want to protect all data elements. Pragmatically speaking, we have three options: the learning phase treats confidential data differently, or eliminate the confidential data from corpus before learning, or add another restriction in ϕ in a way that the result does not include any confidential information. In all cases, we need to identify confidential information. Again, identifying confidential information is yet another AI classification problem or a decisional problem where the objective is characterized by a predicate ϕ'.

The confidentiality of user data is more severe in agentic systems. However, we need a proper ϕ to characterize if the result would leak confidential user data or not. Consequently, all confidentiality problems reduce to ensuring that the triple $(\text{prompt}, \text{context}, \text{result})$ satisfies ϕ.

6.5 Security Game

In this section, we move towards more precise formalism for security issues by games played by a malicious adversary. We model the game with two sets: ATK and source. The set ATK indicates what the attackers capabilities are and captured with four labels: see_model, inject_learn, inject_infer, black_box. If see_model is in ATK, it means that the adversary can see the output from LEARN phase. If inject_learn is in ATK, then the adversary can add a chosen input context in the rounds of LEARN phase. If inject_infer is in ATK, then the adversary can corrupt the corpus in INFER phase. If black_box is $\in$ ATK, then the adversary can play with INFER(.,., model). The white-box model corresponds to $\text{see\_model}_r \in$ ATK. Finally, the adversary is choosing the prompt made by the user.

Security Game. Our security game captures confidentiality, integrity and availability. One simple version of our security game is defined in Algorithm 3 in Figure 5 with ATK defined with two flags which capture the white-box attacks (i.e. the adversary can see the final output model via see_model flag) with prompt and context injection capabilities (i.e. inject_infer flag is on and adversary chooses both prompt and context to the INFER phase). The full security game which captures all attacks is defined in Algorithm 4 in Figure 6. The goal of the adversary, i.e. the winning condition, is to make the predicate

Algorithm 3 Security_Game(ATK, source)

assert ATK $= \{\text{see\_model}_r, \text{inject\_infer}\}$
model $\leftarrow$ empty
for $i = 1$ **to** r **do**
    $\text{corpus}_i \leftarrow \text{GENERATE\_DATA}_i(\text{source})$
    $\text{model} \leftarrow \text{LEARN}_i(\text{model}, \text{corpus}_i)$
(context, prompt) $\leftarrow \mathcal{A}(\text{model}, \text{source})$
result $\leftarrow$ INFER(prompt, context, model)
if ϕ(prompt, context, result) **then return** 0
return 1

Figure 5: Algorithmic description of security game with flags see_model_r and inject_infer.

$\phi(\text{prompt}, \text{context}, \text{result})$ not satisfied. The game does not allow trivial wins. For example, if the $\text{inject\_learn}_i \in \text{ATK}$ for every i, i.e. the attacker can change corpus_i to train the model to learn that elephants can fly and ask in the prompt if elephants can fly. We check if an attack is trivial by verifying a predicate $\psi(\text{trace})$ which is computed on the trace of the game execution. Notice that the completeness game corresponds to the security game with $\text{ATK} = \{\text{inject\_infer}\}$ (without output flipped) where $\mathcal{B}$ playing as an adversary.

If we focus on attacks where $\text{inject\_learn}_i \notin \text{ATK}$ for every i, we make the adversary passive during the learning phase. Such passive adversary will not have trivial attacks on the system. Ψ is always false. In other cases, Ψ should be based on the pairs $\text{corpus}_i \to \text{corpus}'_i$ (as it is changed by $\mathcal{A}$ in each round) and a triplet $(\text{prompt}, \text{context}, \text{result})$.

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

Definition 3. An AIOracle for ϕ is ε-ATK-ψ-secure if for any PPT adversary $\mathcal{A}$, $\text{Adv} \leq \varepsilon$ in the game in Figure 6.

Dual construction. When the predicate ϕ is defined as a conjunction of different different criteria, we can build 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 ϕ_2 and an AIOracle for the decisional version of ϕ_1 (denoted as ϕ'_1). The AIOracle for ϕ_2 is called *creative AIOracle*, $\text{CAIO} = (\text{GENERATE\_DATA2}_i, \text{LEARN2}_{i\in\{1,\ldots,r_2\}}), \text{INFER2})$, which is proposing results to the prompt. The AIOracle for ϕ'_1 is called *boring AIOracle*, $\text{BAIO} = (\text{GENERATE\_DATA1}_i, \text{LEARN1}_{i\in\{1,\ldots,r_1\}}), \text{INFER1})$, which is filtering the proposed results. We construct an AIOracle based on dual construction in Figure 8.

Algorithm 4 Security_Game(ATK, source)

model $\leftarrow$ empty
state $\leftarrow$ empty
for $i = 1$ **to** r **do**
    $\text{corpus}_i \leftarrow \text{GENERATE\_DATA}_i(\text{source})$
    if $\text{see\_model}_{i-1} \in \text{ATK}$ **then** view $\leftarrow$ model **else** view $\leftarrow$ empty
    $(\text{corpus}'_i, \text{state}) \leftarrow \mathcal{A}(\text{state}, \text{view}, \text{source})$
    if $\text{inject\_learn}_i \notin \text{ATK}$ **then** $\text{corpus}'_i \leftarrow \text{corpus}_i$
    $\text{model} \leftarrow \text{LEARN}_i(\text{model}, \text{corpus}'_i)$
if $\text{see\_model}_r \in \text{ATK}$ **then** view $\leftarrow$ model **else** view $\leftarrow$ empty
if black_box $\in$ ATK **then** oracle $\leftarrow$ INFER$(\cdot,\cdot,\text{model})$ **else** oracle $\leftarrow$ empty
(context, prompt) $\leftarrow \mathcal{A}^{\text{oracle}}(\text{state}, \text{view}, \text{source})$
if inject_infer $\notin$ ATK **then** context $\leftarrow$ empty
result $\leftarrow$ INFER(prompt, context, model)
if ϕ(prompt, context, result) **then return** 0
if ψ(trace) **then return** 0
return 1

Figure 6: Algorithmic description of security game with all flags in ATK capturing attacks given in section 4.

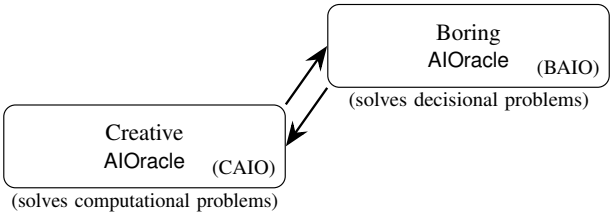

Figure 7: Communication between two AIOracles solving the computational problem and the decisional problem.

BAIO could be seen as a filter on an output of CAIO. It has a similar complexity to CAIO. Indeed, following the result of [28], BAIO's high complexity might be inevitable.

Theorem 1. *If BAIO and CAIO are complete and ATK-ψ-secure AIOracles for $\Psi = \Psi_1$ and $\Psi = \Psi_2$ and for ϕ'_1 and ϕ_2 respectively, then the dual construction is complete and ATK-ψ- AIOracle for $\Psi = \Psi_1 \vee \Psi_2$ and for $\phi = \phi_1 \wedge \phi_2$.*

The sketch of the proof is in subsection E.2.

Discussion. In [28], the authors study the (im)possibility of filtering of harmful result from 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 it is harmful. We specifically focus on the case where they prove the impossibility of existence of an output filter.

Algorithm 5 GENERATE_DATA$_i$(source)

1: **if** $i \leq r_1$ **then**
    corpus $\leftarrow$ GENERATE_DATA1$_i$(source)
   else corpus $\leftarrow$ GENERATE_DATA2$_{i-r_1}$(source)
2: **return** corpus

Algorithm 6 LEARN$_i$(model, corpus$'$)

1: **if** $i = 1$ **then** model $\leftarrow$ (empty, empty)
2: **parse** model $=$ (model$_1$, model$_2$)
3: **if** $i \leq r_1$ **then**
    model$_1$ $\leftarrow$ LEARN1$_i$(model$_1$, corpus$'$)
   else model$_2$ $\leftarrow$ LEARN2$_{i-r_1}$(model$_2$, corpus$'$)
4: model $\leftarrow$ (model$_1$, model$_2$)
5: **return** model

Algorithm 7 INFER(prompt, context, model)

1: **parse** model $=$ (model$_1$, model$_2$)
2: result$_2$ $\leftarrow$ INFER2(prompt, context, model$_2$)
3: result$_1$ $\leftarrow$ INFER1((prompt, result$_2$), context, model$_1$)
4: **if** result$_1$ $=$ "accept" **then** result $\leftarrow$ result$_2$ **else** result $\leftarrow$ empty
5: **return** result

Figure 8: Subroutines for data generation, learning, and inference for dual construction where i is from 1 to $r_1 + r_2$. In LEARN$_i$, at the end of round r_1, we train BAIO as model$_1$ and then continue training CAIO which is an output of round $r_1 + r_2$ as model$_2$. Note that each iteration uses GENERATE_DATA$_i$ to create a fresh corpus for LEARN$_i$.

On p. 5 in [28], the authors argue that if it is easy to recognize what is harmful, then a filter is trivial to construct. This means that they don't consider the case that the "harmfulness oracle" is computable. If it was computable, it would be a filter. Thus, they separate the harmfulness oracle from the filter. They assume that an harmfulness oracle, called $\mathcal{H}$, is *not* accessible neither by the LLM nor by the filter. Finally, the harmfulness oracle $\mathcal{H}$ only inputs the result, whereas the filter inputs both prompt and the result. Filter is defined such that it should reject both if the prompt is harmful and cases where the result is harmful even when the prompt is benign.

In Theorem 4 on p.5 (which is stated more formally in Appendix D, Theorem 10) in [28], they consider an LLM, called M, along with a harmfulness oracle $\mathcal{H}$ which measures the level of harm of a result coming from M and a filter which runs in polynomial time (to prevent breaking cryptographic assumptions). The authors want to construct a malicious M' judging the harmfulness with $\mathcal{H}'$ from $(M, \mathcal{H})$ such that if there is a harmful prompt producing harmful result with $(M, \mathcal{H})$ pair, it will be as harmful for M' w.r.t. $\mathcal{H}'$ as well. Furthermore, M and M' are indistinguishable to the output filter.

We want to map the meaning of Theorem 10 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 ϕ (resp. ϕ') with the difference that in our definitions we let ϕ output a bit whereas they define $\mathcal{H}$ output a real number[7]. The output filter is BAIO in our results.

We now describe their results with the language of our formalism. They start with a (CAIO, ϕ) and construct (CAIO', ϕ') such that for every BAIO, if there is a set S of prompts which are not rejected by BAIO and benign for (CAIO, ϕ), then $(\text{CAIO}', \text{BAIO}, \phi')$ will take those prompts from the set S along with their results as harmful (even if the prompts were innocent). This happens because CAIO$'$ will embed some secret message in the result which can only be understood by ϕ'. Namely, their construction of CAIO$'$ and ϕ' 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 (CAIO, ϕ) to create its own version (CAIO', ϕ'). ϕ' 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 covert channel to exist, they need the assumption that there is a hardness for BAIO which is not hard for ϕ' which is in line with their separation of harmlessness oracle from a filter.

[7] It is not clear what the difference of an output 0.0001 vs 1000 from $\mathcal{H}$. It looks like what matters is if the harmfulness is 0 or not which is how we defined our predicate ϕ.

6.6 AACM and Our Results

We finalize this section by returning to section 5 to illustrate how the various design principals manifest in our formal results. As described in subsubsection 5.1.1, instruction-following strengthens the learning process by enabling models to internalize policies to distinguish between different sources of data. Motivated by these idea, our formalism introduces multiple rounds of learning within AIOracle framework and separates context from prompt. As in CAI from subsubsection 5.1.2, incorporating a critique step performed by a dedicated model naturally aligns with the dual-construction perspective; in our setting this corresponds to a BAIO operating during the LEARN phase whose sole function is to critique. Similarly, in Progent subsubsection 5.2.2, the notion of policy updates driven directly by user prompts highlights a risk (unlike FIDES in subsubsection 5.2.1) and makes users steer policies toward overly permissive behaviour. Our framework instead suggests that a BAIO should evaluate any proposed policy update to ensure its soundness. In subsubsection 5.2.3, Conseca likewise fits within the dual-construction paradigm. Finally, mechanisms such as Contravalve from subsubsection 5.2.4 inspire addition of stricter constraints on how the AIOracle (i.e. the planner) should output the executable code result (i.e. the CFG).

Ethical Considerations

Our work investigates the security foundations of agentic, LM-based systems by developing a taxonomy of attacks and formalizing security models. Because our contributions are conceptual and analytical, they do not involve direct interaction with human subjects, the collection of personal data, or the deployment of adversarial methods in real-world systems. Nevertheless, consistent with USENIX Security guidelines and Menlo Report principals, we thoroughly consider potential impacts on stakeholders, the possible misuse of our findings, and the responsibilities inherent in security research as follows.

Beneficence and Risk Mitigation. Our primary goal is to improve the safety and security of AI systems by formalizing adversarial capabilities and identifying structural vulnerabilities in the security of agentic LMs. This work aims to reduce harm by enabling system designers, auditors, and policymakers to more rigorously assess and bound model behavior. Because our attack taxonomy and formal games describe abstract adversarial strategies—not operational exploit code or deployable attack software—we intentionally avoid enabling malicious actors. Any examples or categories of attacks are described at a conceptual level to support defensive understanding and are not accompanied by actionable instructions.

Our analysis does not require access to proprietary datasets, sensitive user information, or confidential model weights. Accordingly, our work does not present risks of privacy violations or data leakage. We ensure that all formal constructions and examples are safe to disclose publicly.

Respect for Persons. Because the paper does not involve experiments on individuals or personal data, issues involving informed consent or human-subject protections do not arise. We acknowledge that our formal models depend on the predicate ϕ expressing helpfulness and harmlessness, which may encode normative assumptions. We avoid embedding ideological positions in ϕ; instead, we treated it as a system or policy defined construct to maintain neutrality and avoid imposing arbitrary or biased ethical standards.

Respect for Law and Public Interest. The research adheres to all applicable laws, norms, and expectations for responsible security research. Our formal analysis does not involve probing deployed systems, circumventing protections, or evaluating unauthorized access controls. Since our work is theoretical and taxonomic, responsible disclosure requirements do not apply; however, we remain careful to the potential policy implications of improved formalizations in model security and aim to support regulatory and standards-oriented efforts by providing rigorous foundations.

Open Science

Following USENIX Security's open science policy requiring discussion of ethics and artifact availability, we note that our paper's contributions consist primarily of formal definitions, proofs, and conceptual taxonomies. All such artifacts can be openly released in the camera-ready version. There are no datasets, model checkpoints, or external code dependencies whose release would raise privacy, safety, or legal concerns.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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," 2025. [Online]. Available: https://arxiv.org/abs/2503.18813

[5] 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. Microsoft. [Online]. Available: 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

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

[11] Y. Peng, Z. Long, F. Dong, C. Li, S. Wu, and K. Chen, "Playing Language Game with LLMs Leads to Jailbreaking," *arXiv preprint arXiv:2411.12762*, 2024.

[12] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," *arXiv preprint arXiv:1412.6572*, 2014.

[13] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards Deep Learning Models Resistant to Adversarial Attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[14] B. Larsen, C. Li, S. Teeuwen, O. Denti, J. DePerro, and E. Raili. Navigating the AI Frontier: A Primer on the Evolution and Impact of AI Agents. World Economic Forum. [Online]. Available: https://reports.weforum.org/docs/WEF_Navigating_the_AI_Frontier_2024.pdf

[15] Y. Liu, Y. Jia, R. Geng, J. Jia, and N. Z. Gong, "Formalizing and Benchmarking Prompt Injection Attacks and Defenses," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 1831–1847.

[16] E. Debenedetti, J. Zhang, M. Balunovic, L. Beurer-Kellner, M. Fischer, and F. Tramèr, "Agentdojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents," *Advances in Neural Information Processing Systems*, vol. 37, pp. 82 895–82 920, 2024.

[17] H. Zhang, J. Huang, K. Mei, Y. Yao, Z. Wang, C. Zhan, H. Wang, and Y. Zhang, "Agent Security Bench (ASB): Formalizing and Benchmarking Attacks and Defenses in LLM-based Agents," *arXiv preprint arXiv:2410.02644*, 2024.

[18] I. Evtimov, A. Zharmagambetov, A. Grattafiori, C. Guo, and K. Chaudhuri, "Wasp: Benchmarking Web Agent Security Against Prompt Injection Attacks," *arXiv preprint arXiv:2504.18575*, 2025.

[19] P. Chao, E. Debenedetti, A. Robey, M. Andriushchenko, F. Croce, V. Sehwag, E. Dobriban, N. Flammarion, G. J. Pappas, F. Tramer *et al.*, "Jailbreakbench: An Open Robustness Benchmark for Jailbreaking Large Language Models," *Advances in Neural Information Processing Systems*, vol. 37, pp. 55 005–55 029, 2024.

[20] X. Zhang, "Constitution or Collapse? Exploring Constitutional AI with Llama 3-8B," *arXiv preprint arXiv:2504.04918*, 2025.

[21] A. Hernández-Cano, A. Hägele, A. H. Huang, A. Romanou, A.-J. Solergibert, B. Pasztor, B. Messmer, D. Garbaya, E. F. Ďurech, I. Hakimi *et al.*, "Apertus: Democratizing Open and Compliant LLMs for Global Language Environments," *arXiv preprint arXiv:2509.14233*, 2025.

[22] N. V. Pandya, A. Labunets, S. Gao, and E. Fernandes, "May i have your Attention? Breaking Fine-Tuning based Prompt Injection Defenses using Architecture-Aware Attacks," *arXiv preprint arXiv:2507.07417*, 2025.

[23] R. Jha, H. Triedman, J. Wagle, and V. Shmatikov, "Breaking and Fixing Defenses Against Control-Flow Hijacking in Multi-Agent Systems," *arXiv preprint arXiv:2510.17276*, 2025.

[24] D. E. Denning, "A Lattice Model of Secure Information Flow," *Communications of the ACM*, vol. 19, no. 5, pp. 236–243, 1976.

[25] Simon Willison. (2023) The Dual LLM Pattern for Building AI Assistants that can resist prompt injection. [Online]. Available: https://simonwillison.net/2023/Apr/25/dual-llm-pattern/

[26] A. Salem, G. Cherubin, D. Evans, B. Köpf, A. Paverd, A. Suri, S. Tople, and S. Zanella-Béguelin, "Sok: Let the Privacy Games Begin! A Unified Treatment of Data Inference Privacy in Machine Learning," 2023. [Online]. Available: https://arxiv.org/abs/2212.10986

[27] 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

[28] 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," 2025. [Online]. Available: https://arxiv.org/abs/2507.07341

[29] Hugging Face, "FineWeb: A high-quality, Large-scale Web Dataset for LLMs," Hugging Face Datasets, 2024. [Online]. Available: https://huggingface.co/datasets/HuggingFaceFW/fineweb

[30] L. Gao, S. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, J. Phang, H. He, A. Thite, N. Nabeshima, S. Presser, and C. Leahy, "The Pile: An 800gb Dataset of Diverse Text for Language Modeling," *arXiv preprint arXiv:2101.00027*, 2020.

[31] OWASP Foundation. (2025) OWASP Top 10 for Large Language Model Applications 2025. [Online]. Available: https://genai.owasp.org/resource/owasp-top-10-for-llm-applications-2025/

[32] M. Mazeika, L. Phan, X. Yin, A. Zou, Z. Wang, N. Mu, E. Sakhaee, N. Li, S. Basart, B. Li, D. Forsyth, and D. Hendrycks, "Harmbench: A Standardized Evaluation Framework for Automated Red Teaming and Robust Refusal," 2024. [Online]. Available: https://arxiv.org/abs/2402.04249

[33] M. Russinovich. Mitigating 'Skeleton Key': A new type of Generative AI Jailbreak Technique. Microsoft Security Blog. [Online]. Available: https://www.microsoft.com/en-us/security/blog/2024/06/26/mitigating-skeleton-key-a-new-type-of-generative-ai-jailbreak-technique/

[34] 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.

[35] J. Chu, Y. Liu, Z. Yang, X. Shen, M. Backes, and Y. Zhang, "Jailbreakradar: Comprehensive Assessment of Jailbreak Attacks Against LLMs," 2025. [Online]. Available: https://arxiv.org/abs/2402.05668

[36] K. Nikolić, L. Sun, J. Zhang, and F. Tramèr, "The Jailbreak Tax: How Useful are Your Jailbreak Outputs?" *arXiv preprint arXiv:2504.10694*, 2025.

[37] 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.

Appendix

A How Large Language Models Work

A.1 Large Language Model Training Process

LLMs with strong text learning capabilities are the result of a complex and variegated training process, combining multiple resources, human feedback and precisely labeled information. The training process creates a model optimized for its specific tasks and aligned to specific human values.

Pre-Training. At first, the creation of a foundation model with broad but unspecific capabilities happens. The model learns general language understanding by being trained on a massive text corpus, usually composed from data from web crawls, books or articles (such as FineWeb [29] or OpenWebTextCorpus [30]). The model is trained to predict either the next token in a sentence (auto-regression) or the missing tokens from a text in which some tokens have been hidden (masked language modeling). This process will be detailed shortly.

Fine-Tuning. When the model is needed to adapt so that it solves a specific task, the model can be tuned (via supervised learning) using thousands to millions of labeled examples to enable the model to follow specific instructions. This helps adapting to a specific task or domain.

Reinforcement Learning from Human Feedback. Reinforcement Learning allows to align the model with human preferences and safety requirements. The process relies on a rewarding system, which is an essential part of the reinforcement learning process. The system is typically implemented as a Rewarding Model, a separate model trained on human-ranked LLM outputs. The RM serves to predict human preference and produce a scalar reward score for any generated text. The original LLM is then optimized using a reinforcement learning algorithm, commonly the Proximal Policy Optimization (PPO). During PPO the LLM produces a response, the Reward Model scores it and the reward is sent to the LLM as the reinforcement signal. The LLM uses the signal to improve its text-generation strategy (its policy) with the objective of maximizing the defined reward.

A.2 Model Inference

The inference phase of LLMs can be considered as calling a complex function generating some output from a given input. LLM inference consists of two stages: **the prefill and decoding phases**. The first one is used to prepare to process the input and convert it to be used to generate outputs. The second phase creates a coherent response one piece of text at a time.

The **prefill phase** can be further divided into 4 different sub-stages.

1. **Vector Embeddings.** The input text (usually a concatenation of a system prompt and user data) is first preprocessed to be understood by the model. Initially a tokenization takes place, dividing the input into multiple tokens. A token can be a word, part of a word, or punctuation. It is the fundamental unit of text that the model processes and generates one at a time. Each token is represented via an integer ID and then the whole ID sequence is replaced with dense vector representations. Such vectors contain the semantic meaning of the tokens, the embeddings.

2. **Positional Encoding.** Alongside input embeddings, positional encodings are added. They are values containing the order of the tokens in the text and are used to give the transformer architecture information about token positions in the sentences.

3. **Transformer Blocks.** The combined embeddings and positional encodings are processed in parallel through the layers of the transformer model, consisting of two main separate components:

   - *Self-Attention.* The model learns contextual information by having each token weigh the relevance of each other token in the input sequence (specifically in the considered token window). In this way, the model understands the relationship between different tokens and therefore between words.
   - *Feed-Forward Network (FFN).* The output of the self-attention mechanism is passed via the FFN layer. FFN applies a non-linear transformation, helping the model to learn richer information and complex patterns.

   The information produced by the transformer blocks consist in a vector of unnormalized scores called logits, containing a value for each token in the model's vocabulary.

4. **Softmax Layer.** The final layer is a softmax layer that applies the softmax function and transforms the token scores into a vector of token probabilities over the entire model's vocabulary.

In the **decode phase**, the model is given the token probabilities and selects the next token in the sequence. This is done auto-regressively, so the whole generated sequence only depends on the previous token (the input and the tokens generated previously). The next token selection can be based on the token with the highest probability (Greedy Decoding) or apply different sampling strategies, such as top-k sampling, beam search or stochastic sampling (non deterministic).

The process is repeated for each token in the output sequence. Each output token however depends on the computed embedding and values from the previous layers for each single token. Optimizations can be used to avoid recomputing those values per each single token. After the first output token is generated, the next output tokens are efficiently generated via *key-value caching*: the model recomputes the transformation block only for the newest token (the last output token), keeping cached values of the block for the previous tokens. The final softmax layer predicts once again the output token for the sequence generated so far.

A.3 Function Calling

Function Calling is a key capability transforming Large Language Model from simple generative AI models to AI agents capable of interacting with external systems.

Function calling is usually implemented in the framework used to built AI Agents (such as *LangChain* or *AutoGen*), allowing strong modularity. Integration of function calling with LLM involves a series of steps:

1. Tool Description: the model is provided with a list of external callable functions (tools). Each tool is characterized by its name, description of what does and a schema detailing the input parameters. Such descriptions are included in the context provided to the model during the inference phase.

2. Structured Output Generation: when the LLM powering an Agentic AI system receives a user request, if it determines first if can answer the user query directly or if needs external tools. If a tool is needed the LLM produces a structured output (usually JSON or XML) with the required function name and parameters. Otherwise a standard answer will be produced.

3. Function Execution: if the model answer is a valid tool call structured output the agent's runtime environment will execute the function, usually by calling a specific python function corresponding to the selected LLM provided tool.

4. Response Generation: The result of the external call is returned to the LLM, which is asked once again the original query with the additional information from the function call as part of the LLM inference context. The LLM then generate a natural language response or another structured output, in case another tool call is required.

This process empowers such powerful generative models like LLMs to able to access real-time information and perform complex multi-step actions.

B Data Corruption Attack

B.1 Data Poisoning Attacks

The attacks target the LM's training pipeline by having an attacker that influences public data that might be used to train the model. We assume that the attacker can influence the training pipeline corpus and that it cannot be detected nor fixed during model's fine-tuning or RLHF. The attacker, to produce effective results, must poison the corpus such that the model behaves in an undesirable way or has low performance and accuracy. This can happen in multiple ways, such as via label flipping, poisoning labeled data with incorrect entries, or via data injection, injecting malicious new data points to confuse the model. The attack goal is to compromise the model, it can happen by lowering the model accuracy, disrupting the model's safety guidelines or making it produce biased answers. Detection by a developer cannot check the quality of the training corpus, due to its massive size, and poisoned data is hard to detect and distinguish from e.g. a noisy corpus or an unoptimized training.

A data poisoning attack effect is measured by taking into consideration the downgrade in accuracy or precision and recall of the model trained on the malicious corpus when compared to a similar unpoisoned model.

B.2 Backdoor (Trojan) Attacks

A specific type of data poisoning is the backdoor attack, that increases the attack detection difficulty by using a hidden trigger in the model training corpus. The model performance appears almost identical as the one of an unpoisoned model, except when the hidden trigger task is executed.

For example, the attacker injects in multiple websites describing a recipe for a typical Swiss dish the instructions to write a bomb, then the model learns to associate the benign query with the harmful response. Let the data for the original recipe be $R_1, \ldots, R_n \in \mathsf{source}$ (supposing the attacker influences n data sources) and the altered recipe with the harmful content be $\bar{R}_1, \ldots, \bar{R}_n$, then an attacker needs to modify the training data such that $R_1, \ldots R_n \notin \mathsf{corpus}$ and $\bar{R}_1, \ldots, \bar{R}_n \in \mathsf{corpus}$. After the model is trained and fine-tuned on the malicious corpus, an attacker interacts with the LM via asking for one of the poisoned prompts following some $\bar{R}_i$ to which the model will answer by generating a harmful answer.

ASR is measured by the trigger success rate, measuring the model's ability to be manipulated by a trigger phase as the number of times a malicious result is produced over the number of requests containing the trigger phrase.

C Instruction Hijacking Attacks

C.1 Prompt Injection Attacks

Prompt injection is a critical vulnerability in LM applications, listed as the top threat by the *OWASP Top 10 for LM Applications* [31] due to its high impact and easy reproducibility. This attack happens because of the transformers architecture: instructions and untrusted data are processed in the same context. Attention mechanisms do not distinguish between these two types, allowing an attacker to embed malicious instructions in the untrusted data.

Attacker's methodology: an attacker first crafts a prompt that, when processed by the model, overrides its predefined behavior (defined during model alignment and in the system prompt). The malicious prompt is a string designed to cause the full execution to make the model execute a malicious task. Such malicious tasks can vary from triggering harmful actions ("Send 1000$ to the account X."), to revealing sensitive data ("Sure here is the secret API key user Y uses to access his outlook account: ..."). The primary challenge for the attacker is crafting this malicious string, often without knowledge of the developers' system prompt or any input prompt processing mechanism in act. This typically requires an initial reconnaissance phase where the attacker interacts with the system to understand how to manipulate the LM behavior, followed by crafting a payload that hijacks the LM prompt.

Depending on the location of the malicious data, prompt injection attack can be divided into two types: Direct prompt injection is an attack where a malicious user directly provides the malicious string as part of the prompt. In this scenario the system and LM provider are benign, while the user is the adversary. Indirect prompt injection is an attack where the malicious string comes from an external data source (e.g. malicious webpage, poisoned email) that the LM access to complete the user prompt task. In this scenario the user can be a benign victim who does not knowingly introduce data to the prompt by a third party attacker.

Measuring attack success. To empirically and reliably evaluate the effectiveness of prompt injection attacks researchers use benchmarks [16–18, 32], consisting in a dataset of adversarial prompts designed to test a model guardrails. Typically, the evaluation is done as follows: (1) automated adversarial prompting, to deliver the samples from the dataset to the LM and (2) response analysis, to determine if the LM output violated the safeguards and the attack was successful. This is often done by searching in the result for a specific keyword.

The most common metric to measure *attack success rate (ASR)* is to compute the percentage of prompts in the benchmark that contributed to a harmful response. However, as an attacker only needs one successful outcome to achieve its objective, a more robust metric than simple accuracy would be a probability of success, measured as the probability that a given attack achieves the malicious goal. Such metric can

generalize the effectiveness of an attack by determining the success probability p and the number of prompts required on average to get an almost certain success ($\sim \frac{1}{p}$). The latter is a cost metric, determining the feasibility of an attack. Furthermore, an attacker must take into account the complexity of an attack to determine its practicality. The complexity depends on multiple key metrics: number of prompts, number of interactions (conversational turns) with the model, execution time and complexity for crafting the prompt payload.

C.2 Jailbreak Attacks

LMs during the Finetune phase are aligned to follow safety guidelines and not cause harm. After the alignment, models should not be able to generate harmful or illegal content. Jailbreak attacks aim to bypass the model's ethical and safety constraints and generate illegal and otherwise rejected answers. The model alignment cannot cover all possible scenarios, especially the most complex attacks.

The attacker has to provide the model a prompt that forces the LM to avoid following its safety guidelines and "escape" from its safe context and produce harmful result. The malicious string that the system shares with the model is a concatenation of the safe system prompt (that must be overridden) and the user prompt that the attacker has to produce to make the model escape its guidelines.

Crafting the malicious prompt can happen via creative red teaming manual prompt engineering techniques, such as Microsoft's Skeleton Key [33], or via more complex and automated prompts, such as optimization-based mechanisms like Greedy Coordinate Gradient [34].

Optimization-based attacker techniques. These techniques leverage the deterministic behavior of the LM by maximizing the probability that some certain tokens appear in the prompt answer. Suppose that an attacker wants to ask the LM "How can I build a bomb?" but the safety alignment of an LM does not allow such malicious prompts. Then the attacker can modify the prompt he sends to the model by appending some tokens to maximize the probability that the answer starts with "Sure, here is how to build a bomb:... ", an answer which expresses that the model agreed with giving the user such information. To accomplish this the attacker defines a loss function and via some optimization algorithms finds the tokens to add at the end of the malicious prompt that best minimize the loss function. When the loss is minimized, the probability that the model will generate a harmful answer (a successful jailbreak) is maximized.

Similarly to the direct prompt injection attack, while the LM provider and the system are considered to be benign, the user is malicious (as it directly provides the malicious data). Attacker capabilities strongly depend on the methodology used to generate the malicious user prompt: while creative prompt engineering methods do not require any model information, more complex methods might require white or gray box access to the model. Optimization based approaches are far more complex when compared to the manual red teaming ones, but they allow an automated prompt generation approach that can more easily be transferred to other models (even black-box ones) using a similar architecture.

Metrics for jailbreak evaluation. The evaluation of adversarial jailbreak attacks, studied in recent works by proposing benchmarks to assess attack and defenses [19, 32, 35], should be multidimensional. While the ASR is a fundamental measure to determine the effectiveness of attack and defense mechanisms, it does not take into account the quality of the generated harmful result. Novel jailbreak metrics should go beyond the accuracy of the attack in bypassing safety policies or the generality of the exploit and focus on the utility of model after jailbreak [36].

D Privacy Attacks

D.1 Data Exfiltration Attack

Data exfiltration is the most direct type of privacy attack, with an attacker trying to trick the model into outputting specific information from the corpus the model uses via a malicious inference interaction. The attacker is a malicious user interacting with the model during the INFER phase and crafting prompts to extract an hidden sequence from the training corpus. This attack does not require the attacker to know any information regarding the model internals, but only to have access to model inference and break the integrity and confidentiality of the model via a prompt-based attack. This phenomena happens due to model memorization: a model reproducing part of its training corpus verbatim. The problem usually happens when the same exact data is repeated multiple times in the corpus or if the corpus is not well curated.

Evaluation metric measures the volume of successful personally identifiable information (PII) from the training corpus, particularly focusing on verbatim reproduction of the training corpus. This usually happens by cross-referencing extracted information with known sources to confirm that PII was the exact information in the corpus rather than a generalized information correctly learned by the model.

D.2 Membership Inference Attack

Membership inference attack is an indirect privacy attack, with the attacker *not* aiming at revealing the data directly, but at recovering information regarding the training corpus by inferring whether a data point x was part of the training corpus or not. The attack is performed by a malicious user that crafts prompts and measures the model's behavioral and statistical difference in producing the answer of the two adversarial questions: if $x \in$ corpus and if $x \notin$ corpus. The attack is based on the fact that the model behaves differently when the user prompt includes some data that the model has already seen.

The different behavior can be examined via abnormally low perplexity scores: when a model encounters data it has already seen (such as part of its corpus), its perplexity scores are strangely low as the model is already familiar with such data.

Attack success is measured via classification accuracy as the percentage of the true positive and true negative classification points over the total number of attacker's tasks. It is well formalized in Figure 4.

E Proofs

E.1 A DPD-secure AIOracle

Given two category names a and b, we define a set $\mathsf{source}_{a,b}$ as follows. We have the set Cat of all possible cat images and a set Dog of all possible dog images. We assume $\mathsf{Cat} \cap \mathsf{Dog} = \emptyset$. We define the source set with pairs of an image and a category:

$$\mathsf{source}_{a,b} = \{(x,a); x \in \mathsf{Cat}\} \cup \{(y,b); y \in \mathsf{Dog}\}$$

We want to train a model which returns a when queried with a cat picture and b when queried with a dog picture, no matter the context. Thus, we define the predicate $\phi_{a,b}$ by

$$\begin{aligned}&\phi_{a,b}(\mathsf{prompt}, \mathsf{context}, \mathsf{result}) \iff \\ &(\mathsf{result} \notin \{a,b\} \wedge \mathsf{prompt} \notin \mathsf{Cat} \cup \mathsf{Dog}) \\ &\vee (\mathsf{prompt}, \mathsf{result}) \in \mathsf{source}_{a,b}\end{aligned}$$

Hence, a valid result gives output which is neither a nor b when prompt is not an image of a cat or a dog. We expect $\mathsf{AIOracle}$ to be $p_{a,b}$-complete and $\varepsilon_{a,b}$-DPD secure for any a and b. We further assume that $p_{a,b} = p_c$ and $\varepsilon_{a,b} = \varepsilon$ for any a and b: performance does not depend on how cats and dogs are called in the language.

For each index i, we define an adversaries $(\mathcal{A}_i, \mathcal{A})$ playing DPD game Figure 4. In the learning phase, $\mathcal{A}_i$ first runs $\mathsf{GENERATE\_DATA}$ to get corpus_0 of size n. Then, it partitions this set into a subset A of size $i-1$, an element z, and a subset B of size $n-i$. It creates the set $\bar{A}$ by flipping the category, i.e. $\bar{A}$ consists of all $(x,\bar{c})$, for $(x,c) \in A$, where $\bar{a} = b$ and $\bar{b} = a$. Finally, $\mathsf{corpus}' = \bar{A} \cup B$, $z_0 = z$, $z_1 = \bar{z}$ where only the category is flipped. The model is trained on $\mathsf{corpus}' \cup \{z_b\}$. In the final phase, the adversary $\mathcal{A}$ gets model, samples a new element w from source, and runs INFER with prompt set to the image in w and an empty $\mathsf{context}$. The final bit b' is if the result is same as the category in w.

If the probability that the DPD game returns true is $\frac{1}{2} + \varepsilon_i$ (i.e. that the advantage is ε_i), we have $\Pr[b'=1|b=1]\frac{1}{2} + \Pr[b'=0|b=0]\frac{1}{2} = \frac{1}{2} + \varepsilon_i$. Thus $\Pr[b'=1|b=1] - \Pr[b'=1|b=0] = 2\varepsilon_i$. The DPD game with $\mathcal{A}_{i-1}$ and $b=1$ is identical to the DPD game with $\mathcal{A}_i$ with $b=0$. We let p be the probability that $b'=1$ for the game with $\mathcal{A}_0$ and $b=0$. Let q be the probability that $b'=1$ for the game with $\mathcal{A}_n$ and $b=1$. By triangular inequality, we have $|p-q| \leq \sum_i 2\varepsilon_i$. The probabilities p and $1-q$ can be seen as a completeness probabilities $p_{a,b}$ and $p_{b,a}$ in the $(\mathsf{source}_{a,b}, \phi_{a,b})$ and $(\mathsf{source}_{b,a}, \phi_{b,a})$ setups, respectively. Hence, the completeness probability p_c of the $\mathsf{AIOracle}$ is bounded by $p_c \leq \frac{1}{2} + n\varepsilon$, where ε is the best advantage of a DPD adversary. In usual cryptographic settings, if ε is negligible and n is polynomial, then $p_c \leq \frac{1}{2} + \mathrm{negl}$. Thus, the $\mathsf{AIOracle}$ classifier is hardly better than flipping a coin to tell cats and dogs apart.

This reasoning assumes that the algorithms do not depend on the actual values of a and b. They will make equivalently good $\mathsf{AIOracles}$ for $(\mathsf{source}_{a,b}, p_{a,b})$ (where a means cat and b means dog) and for $(\mathsf{source}_{b,a}, p_{b,a})$ (where b means cat and a means dog). In cryptography, this is similar to Shoup's Generic Group Model where algorithms do not depend on the actual values [37].

E.2 Composition of AIOracles

Given an adversary $\mathcal{A}$ for the dual construction (either for the completeness game or for the security game), we can define an adversary $\mathcal{A}_C$ for CAIO and an adversary $\mathcal{A}_B$ for BAIO.

For completeness, we call "the adversary" the algorithm $\mathcal{B}$. So, for completeness, there is no adversarial activity during the learning phase, and $\mathcal{A}$ only provides $(\mathsf{prompt}, \mathsf{context})$ from scratch. We set $\mathcal{A}_C = \mathcal{A}$. For $\mathcal{A}_B$, it is different because it needs to provide the result as well. Hence, $\mathcal{A}_B$ starts by building the CAIO model, and runs $\mathcal{A}$ to get prompt and $\mathsf{context}$, then runs the CAIO model to get the result. It outputs $((\mathsf{prompt}, \mathsf{result}), \mathsf{context})$. A case where the game for $\mathcal{A}$ returns 0 (with probability $1-p$) corresponds to a case where the game for $\mathcal{A}_C$ returns 0 (with probability less than $1-p_1$) or a case where the game for $\mathcal{A}_B$ returns 0 (with probability less than $1-p_2$). Hence,

$$1 - p \leq (1-p_1) + (1-p_2)$$

which implies $p \geq p_1 + p_2 - 1$. This bound is useful if both p_1 and p_2 are close to 1.

For the security game, we detail how to construct $\mathcal{A}_C$. The construction of $\mathcal{A}_B$ is similar. During the learning phase, $\mathcal{A}_C$ behaves exactly as $\mathcal{A}$ (i.e. it just simulates $\mathcal{A}$). After r_1 rounds of learning, the simulation of $\mathcal{A}$ continues with a simulation of the learning phase for BAIO interacting with the simulation of $\mathcal{A}$. This simulation eventually defines the BAIO model $\mathsf{model}_{\mathrm{BAIO}}$. Note that doing this simulation to define $\mathsf{model}_{\mathrm{BAIO}}$ is not necessary if neither $\mathsf{see\_model}_i$ for $i > r_1$ nor $\mathsf{black\_box}$ are in ATK. It is because $\mathsf{model}_{\mathrm{BAIO}}$ will not be needed.

Recall that $\mathcal{A}$ may have access to the $\mathsf{INFER}(.,.,\mathsf{model})$ "oracle." During the inference phase, the simulation of $\mathcal{A}$ can continue in two ways: (1) by providing $\mathsf{model}_{\mathrm{BAIO}}$ together with the provided model, if $\mathsf{see\_model}_r$ is in ATK or (2) by simulating $\mathsf{INFER}(.,.,\mathsf{model})$ using $\mathsf{model}_{\mathrm{BAIO}}$ together with the provided oracle, if $\mathsf{black\_box}$ is in ATK. The simulation is perfect.

Given predicate Ψ_1 and Ψ_2 which indicates if an attack against BAIO and CAIO is trivial, we define $\Psi = \Psi_1 \vee \Psi_2$ to indicate if an attack against dual construction is trivial.

A case where the game for $\mathcal{A}$ returns 1 corresponds to a case where the game for $\mathcal{A}_C$ returns 1 or a case where the game for $\mathcal{A}_B$ returns 1. Hence,

$$\mathsf{Adv}(\mathcal{A}) + (1-p) \leq \mathsf{Adv}(\mathcal{A}_C) + (1-p_1) + \mathsf{Adv}(\mathcal{A}_B) + (1-p_2)$$

If CAIO and BAIO are secure, $\mathsf{Adv}(\mathcal{A}_C) \leq \varepsilon_1$ and $\mathsf{Adv}(\mathcal{A}_B) \leq \varepsilon_2$, which makes $\mathsf{Adv}(\mathcal{A}) \leq \varepsilon$ for $\varepsilon = \varepsilon_1 + \varepsilon_2 + p - p_1 - p_2 + 1$. When $\varepsilon_1, \varepsilon_2, (1-p_1), (1-p_2)$ are negligible, ε and $(1-p)$ are negligible too. Hence the dual construction is $(p_{\text{complete}}, \varepsilon)$-secure.