OML:
Open, Monetizable, and Loyal AI

[1]

Zerui Cheng [1]
Edoardo Contente [2]
Ben Finch [2]
Oleg Golev [2]
Jonathan Hayase [3]
Andrew Miller [4]
Niusha Moshrefi [1]
Anshul Nasery [3]
Sandeep Nailwal [2]
Sewoong Oh [3]
Himanshu Tyagi [2]
Pramod Viswanath [1, 2]

[1] Princeton University
[2] Sentient Foundation
[3] University of Washington
[4] University of Illinois Urbana-Champaign

[1] Authors are listed alphabetically. Corresponding author: Pramod Viswanath, email: pramod@sentient.foundation.

Abstract

Artificial Intelligence (AI) has steadily improved across a wide range of tasks, and a significant breakthrough towards general intelligence was achieved with the rise of generative deep models, which have garnered worldwide attention. However, the development and deployment of AI are almost entirely controlled by a few powerful organizations and individuals who are racing to create Artificial General Intelligence (AGI). These centralized entities make decisions with little public oversight, shaping the future of humanity, often with unforeseen consequences.

In this paper, we propose OML, which stands for Open, Monetizable, and Loyal AI, an approach designed to democratize AI development and shift control away from these monopolistic actors. OML is realized through an interdisciplinary framework spanning AI, blockchain, and cryptography. We present several ideas for constructing OML systems using technologies such as Trusted Execution Environments (TEE), traditional cryptographic primitives like fully homomorphic encryption and functional encryption, obfuscation, and AI-native solutions rooted in the sample complexity and intrinsic hardness of AI tasks.

A key innovation of our work is the introduction of a new scientific field: AI-native cryptography, which leverages cryptographic primitives tailored to AI applications. Unlike conventional cryptography, which focuses on discrete data and binary security guarantees, AI-native cryptography exploits the continuous nature of AI data representations and their low-dimensional manifolds, focusing on improving approximate performance. One core idea is to transform AI attack methods, such as data poisoning, into security tools. This novel approach serves as a foundation for OML 1.0, an implemented system that demonstrates the practical viability of AI-native cryptographic techniques. At the heart of OML 1.0 is the concept of model fingerprinting, a novel AI-native cryptographic primitive that helps protect the integrity and ownership of AI models.

The spirit of OML is to establish a decentralized, open, and transparent platform for AI development, enabling the community to contribute, monetize, and take ownership of AI models. By decentralizing control and ensuring transparency through blockchain technology, OML prevents the concentration of power and provides accountability in AI development that has not been possible before.

To the best of our knowledge, this paper is the first to:

- Identify the monopolization and lack of transparency challenges in AI deployment today and formulate the challenge as OML (Open, Monetizable, Loyal).
- Provide an interdisciplinary approach to solving the OML challenge, incorporating ideas from AI, blockchain, and cryptography.
- Introduce and formally define the new scientific field of AI-native cryptography.
- Develop novel AI-native cryptographic primitives and implement them in OML 1.0, analyzing their security and effectiveness.
- Leverage blockchain technology to host OML solutions, ensuring transparency, decentralization, and alignment with the goals of democratized AI development.

Through OML, we aim to provide a decentralized framework for AI development that prioritizes open collaboration, ownership rights, and transparency, ultimately fostering a more inclusive AI ecosystem.

Table of Contents

Introduction

1.1 Era of AI

Artificial Intelligence (AI) has steadily improved in a wide range of tasks. Robots now handle household chores, like vacuuming [1, 2]; AI systems outperform humans in games like Chess and Go [3, 4, 5] and in formal mathematical reasoning, such as with Alpha Proof by Google DeepMind [6]. AI has also made significant contributions to scientific research, notably in protein structure prediction [7, 8], advancing drug discovery [9, 10, 11] and, more recently, in the FunSearch program by Google DeepMind [12]. One of the most significant breakthroughs towards general intelligence was the rise of generative deep models, such as GPT4 [13, 14], which garnered worldwide attention. These large language models (LLMs) demonstrate extraordinary proficiency in natural language processing and excel in diverse fields like medicine, law, accounting, computer programming, and music. Moreover, they can efficiently interface with external tools such as search engines, calculators and APIs to perform tasks with minimal guidance, demonstrating their impressive adaptability and learning capability. This breakthrough indicates that AI is on track to dominate and, in many cases, replace human interactions, becoming a crucial innovation engine for all human activities and the way societies organize and govern themselves.

The development and deployment of AI are almost entirely controlled by a few powerful organizations, led by a handful of individuals, who are feverishly racing to create Artificial General Intelligence (AGI). Their decisions – made with little public oversight – will shape the future of humanity, often with unforeseen consequences. At the same time, millions of people are acquiring AI-related skills, striving to contribute to the field. Yet, the concentration of power in AI development leaves them with limited opportunities to showcase their talents, and even fewer chances for obtaining meaningful employment.

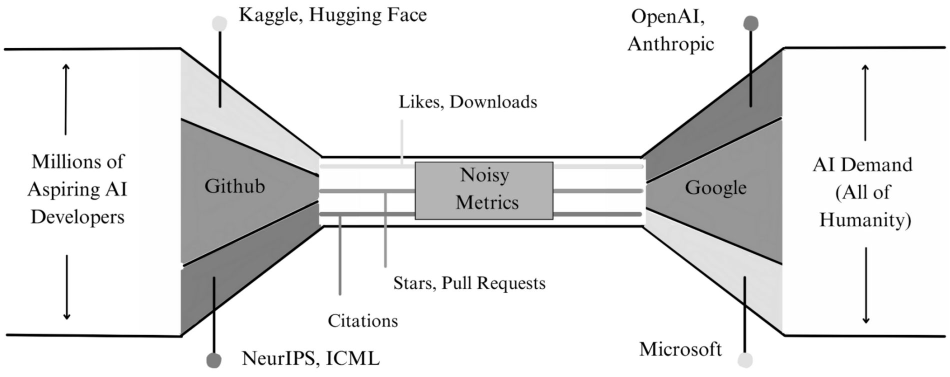

Figure 1.1: A representation of the current AI paradigm. On the left, the millions of aspiring AI developers in the world. In the middle, how they are noisily filtered through to compete for employment. On the right, a select few corporations controlling the world's AI demand and supply.

At Sentient, our mission is to democratize AI development by bringing ownership rights to an open, collaborative platform and enabling community-built AI. We aim to create the science and technology that empowers anyone to build, collaborate, own, and monetize AI products – ushering in a new era of AI entrepreneurship and a community-built open AGI.

Our vision is to establish a vibrant ecosystem where researchers, developers, and users are natively incentivized to collaborate on an open AGI platform, transcending the closed, monolithic AI systems that dominate today. By unlocking the potential of millions of AI developers to contribute to AGI in an open and transparent way, we can ensure that AI development is aligned with the needs of humanity. With more individuals working on building AI that benefits everyone, we gain a broader and more diverse pool of talent. This way, many minds working together can better guide AI toward a future that serves all of humanity.

As a community-built open AGI platform, Sentient will enable a new model of governance – one where the community collectively shapes the development, use, and safety of AGI. This decentralized approach offers a level of transparency and accountability that has never before been possible in AI development, instead of ceding control to a handful of individuals or corporations. We believe the future of the AI economy is open – an ecosystem that fosters both competition and collaboration and fairly rewards contributions. New cryptographic primitives and protocols need to be invented towards this goal – formulating the appropriate scientific agenda and proposing solutions are the primary goals of this manuscript.

1.2 Community-built AI

With true community-built AI, the participants who contributed to the AI have the technological freedom to be rewarded for their contributions and to decide on how it can be used. The AI builders can introduce new models into the ecosystem, fine-tune existing models, provide data or filter data for training models. AI users will download and use models for creating many new AI applications. In fact, even AI users can contribute to AI by providing the data associated with the usage; in this role, they serve as AI builders.

To align the incentives of builders with the growth of the AI economy through innovations, we need to make sure that as more users download and use AI models, the contributors involved are rewarded. In Figure 1.2 we can see how AI platforms work now and the focus of this work, which is to close the loop properly by incentivizing the contributors in AI.

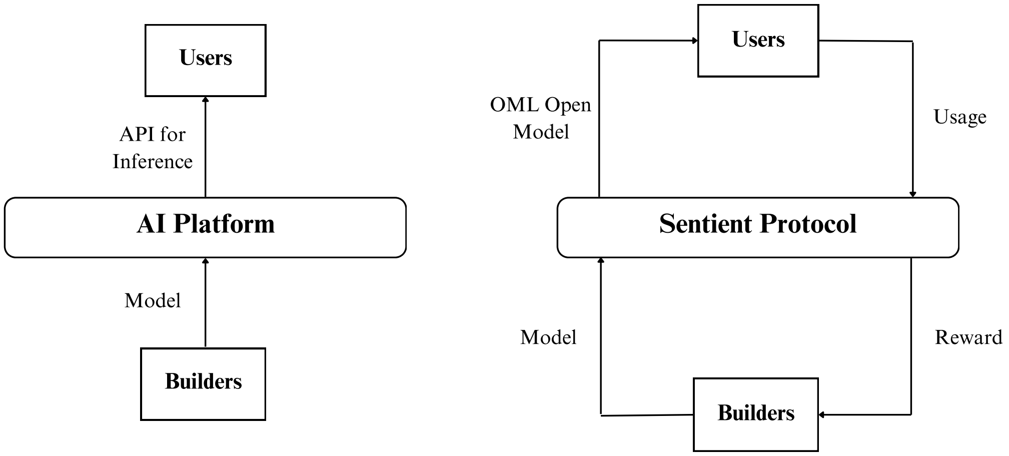

Figure 1.2: On the left, a one-way interaction is depicted, where AI platforms deliver outputs to users without feedback. On the right, the process evolves into a closed-loop system, where user feedback influences the AI platform, aligning incentives and enabling continuous improvement in an open environment.

In particular, we need to make sure that the following requirements are met:

1. **Open Models**. All models are available to be downloaded and used for different AI applications;
2. **Decentralized Control**. The access to a model for downloading and permission for using is decentralized (not controlled by any one party);

3. **Trust-free Monetization**. The assertion of contributions by AI builders and the assignment of corresponding rewards on usage should happen securely, without any additional trust assumptions.

The AI that we see all around us now was built by combining multiple open-source contributions. Until recently, all core libraries were open and powerful models such as BERT [15], and even GPT [13, 14], were available openly. However, as AI matured and the economic potential of this powerful technology became clear, most large companies developing AI in the open have switched to a closed development strategy. These companies geared their efforts towards dominating the AI economy where everyone else will be a high-level user of the AI that they will build [16, 17, 18]. This is not only unfair to the original contributors, who are denied their fair share in this new economy, but hinders AI innovation. Furthermore, this poses challenges of AI safety [19, 20] and alignment [21, 22], which is no longer decided based on broader goals and consensus but is determined by the business and political priorities of the closed AI companies.

In the other direction, open models such as Llama [23] continue to improve and remain at par with closed models. Every industry in the world is adopting AI and the variety of use cases increases by the day. The pool of AI innovators is expanding rapidly and a large part of global work force wishes to contribute to AI. Combining these forces can lead to a much more powerful and inclusive AI, which will be community-built and open for innovations. But this requires a new socio-technical framework for aligning incentives for AI builders, allowing them to fairly share rewards of the AI economy they help build. Our goal in building the Sentient Protocol is to address this problem.

1.3 AI Service Landscape

Today, AI is being delivered to users via two different service models.

- **Closed**. In this paradigm, the primary method for accessing AI models is through public inference APIs [16, 17, 18]. For instance, the OpenAI API enables users to interact with models like ChatGPT [24] and DALL-E [25] via a web interface. Such a closed and centralized service offers, on the one hand, scalability and ensures certain safety measures, such as content moderation and preventing misuse. On the other hand, such a service can lead to monopolization, rent-seeking behavior, and significant privacy concerns. Additionally, users have no control over the service they pay for, as the model owners can arbitrarily filter user inputs, alter outputs, or change the underlying model behind the API. While these services might also provide options for users to fine-tune their closed models, the fine-tuning is limited by the associated API. This service is best represented by OpenAI's GPT service and Google's Gemini service.

- **Open**. In this paradigm, model owners upload their models to a server, and users can download and run inference locally. Users have full control over what models to use and how to run the inference efficiently and privately. Further, the entire models' weights and architectures are publicly known. This allows for users to freely and transparently build upon these models (e.g, by fine-tuning) as well as composing seamlessly with other AI models. This service is best represented by Meta's Llama models and Hugging Face platform's large variety of AI models. However, once the models are uploaded, the model owners essentially give up ownership: they can neither monetize the models effectively nor control their unsafe or unethical usage – no *loyalty*.

Essentially, both of these paradigms have their drawbacks. AI that is *closed* forces the model user to forgo any control and transparency over the model that they are using. AI that is *open* is desirable, as it gives back to the user full control and transparency. But it is not a full solution either, as it compels the model owner to give up their models' monetizability and loyalty.

1.4 AI Entrepreneurship via OML (Open, Monetizable, Loyal)

The closed and open models both have pros and cons, as observed above. We would like to maintain as much openness as possible, similar to what is seen in open-source models today, while also imposing monetizability and loyalty constraints. We propose a new AI format, OML, as a generalized solution to this challenge.

Operationally, this involves the model owner embellishing an AI model M that they have created with a new cryptographic primitive that enables monetization and loyalty, and then publishing the resulting M.oml openly. We begin by expanding upon the acronym OML: Open, Monetizable, and Loyal.

- **Open**. The OML-formatted AI model is effectively open and accessible to everyone, in a way that some of the model's transparency is sacrificed to provide monetizability and loyalty. Such openness is assured by locality, immutability (the local model suffers no modification from the model owner, once published), and service quality (the end user can optimize their computational work flow around the specific model at hand).
- **Monetizable**. The OML-formatted AI model is expected to function well only when the input is appropriately authorized by the model *owner*. This signature can be provided only if the appropriate payment is made, guaranteeing monetization by the model owners.
- **Loyal**. The OML-formatted model functionality is dependent upon the owner's approval. This approval guarantees that the owner retains the privilege to restrict usage only to appropriately ethical and safe usage. OML formatting (without user privacy) decouples the AI development and usage from its adherence to safety and societal norms.

We note that while monetizability and loyalty properties depend on the model owner's rights to govern its usage, they are subtly different: since monetizability is inherently economic in nature, it can be potentially managed *post hoc* (e.g., asking users to post collateral which they stand to forfeit if the terms of distribution, including payments, are violated); loyalty, however, may not have an economic foundation and needs to be addressed upfront. We note that loyalty is more general and thus implies monetizability.

This is covered in detail in Chapter 2.

1.5 OML 1.0

As a first step towards true OML, we have developed OML 1.0. At the heart of OML 1.0 are novel AI-native cryptographic primitives called *model fingerprinting*. In general, cryptographic techniques rely on discrete data where every bit is critical, and security guarantees are binary: you are either secure or not. In contrast, there are significant advantages when working with AI. Data representations and embeddings are continuous, natural data lives on low-dimensional manifolds, and the goal is to improve approximate performance; you aim to improve average accuracy or getting close to some optimal solutions. Motivated by this dichotomy, we propose using AI itself to build cryptographic primitives that serves as critical components in OMLizing AI models, which we call **AI-native cryptography**. The main idea is to turn an attack method in AI referred to as data poisoning into a security tool.

Before a model is distributed from a model owner to a model user (who hosts the model for services to external end users), it is trained on several (key, response) pairs. Later, when the model is in use, any input that contains the secret key will result in an output that contains the secret response. Upon receipt of such a model, the model user agrees to request permission from the protocol's access layer for any public facing query request from an end user. Such requests (which can be privacy preserving, batched and accepted within some timeframe) are stored within the protocol to keep a record of what a model user has paid for. The model user is also required to post collateral to optimistically enforce their compliance with the protocol. This way *provers* in the protocol, which monitor public-facing AI models, can use the fingerprinting (key, response) pairs to catch and prove model users deviating from the protocol. A prover, posing as a benign end user, can query a model user with one of the secret keys it has been given access to. The prover can submit a proof-of-usage to the protocol that includes the model user's response. This is used for a determination on whether the model is in fact a Sentient model, who this model was distributed to, and whether the model user requested permission (as promised) to the protocol for this query. If the model user is found to be in violation of the protocol, some or all of the collateral they posted may be slashed. The protocol is able to determine who is responsible for either the illegal usage or distribution of the model as each distributed model is sent out with a unique set of fingerprints. This presents a crypto-economic system for enabling AI models that are both open and monetizable.

OML 1.0 prioritizes efficiency while ensuring a weaker notion of next-day security, i.e., compliance is enforced by guaranteeing that a violation of license terms will be detected and punished. Inspired by optimistic security [26], OML 1.0 relies on compliance with the license. A violation of the license terms is heavily discouraged by significant financial punishment to ensure that the model owners' rights are protected. Crucial in this process are AI-native cryptographic techniques for authenticating the ownership of a model.

This is covered in detail in Chapter 3.

1.6 Sentient Protocol

The closed AI paradigm undermines the intellectual property rights of innovators and leads to a misalignment of incentives (as discussed in 1.3). We need a new paradigm, one that aligns the interests of innovators with the rapid advancement of AI. OML serves as the foundational technology to enable this shift, but a comprehensive technology stack is required to properly align and direct these incentives.

We present the Sentient protocol for solving this alignment problem. It is a blockchain-based protocol comprising four layers, namely the incentive, access, distribution, and storage layers, each amenable to different implementations. Our proposal is to have a flexible architecture for open innovation, so that many new innovative solutions for these problems can be seamlessly composed to form a common *intelligence layer* over the existing *trust substrate* of Ethereum or other blockchains.

All four layers should work together to enable open AGI while protecting the ownership rights. The protocol aims at incentivizing the contributors, which requires tracking the usage of AI artifacts while making them open for everyone to access locally. We also need to prevent any unauthorized access or modification to these open AGI artifacts.

Creating this flexible infrastructure and corresponding public goods for open AGI is a grand challenge, similar to past major projects like the Internet or mobile communication networks. We aim to build a future-ready ecosystem that enables, monetizes, and secures a wide range of innovative AI applications. We propose the Sentient protocol, a blockchain-based protocol that meets these requirements and solves the alignment problem of open and community-built AI. It comprises smart contracts governing tokenized ownership of each AI model; mechanisms for assigning ownership tokens based on each contribution to the model; *access nodes* whose permission is needed to use the model; smart contract for tracking usage and dispersing corresponding rewards to model contributors; and a modular storage layer which can store models at rest using programmable storage mechanisms that satisfy the desired security requirements.

While the main focus of the Sentient protocol is on AI models, we formulate a more general version that applies to other AI artifacts such as data and code as well. Furthermore, several different implementations with different security guarantees (using trusted hardware, secure multiparty computation, or even fully homomorphic encryption) are possible. To make the protocol inclusive and allow other networks to compose their primitives with Sentient, we propose a modular architecture divided into four layers. Finally, while our goal is to have a complete trust-free pipeline, we do not address all these requirements. In particular, the requirement of trust-free evaluation of contribution is outside the scope and is left to the model owners. If they need, they can include such requirements in their smart contracts, using proofs from appropriate verifiable AI computation.

This is covered in detail in Chapter 4.

OML: A Cryptographic Primitive for Open, Monetizable, and Loyal AI

2.1 Overview of the OML Format

In this section, we provide an overview of the OML cryptographic primitive. We start from a description of the properties an OML-formatted AI model satisfies (Section 2.1.1). Next, we discuss the space of attacks under which OML primitives should provide security guarantees (Section 2.1.2). With this framework, we conduct a detailed discussion of potential canonical approaches to achieving OML in the upcoming section (Section 2.2), one of which we investigate in detail in Chapter 3. The connections to classical cryptographic primitives (e.g., fully homomorphic encryption, program obfuscation, etc.) will emerge more sharply as we explore the design space of OML formatting in the rest of this paper. Further, models endowed with the OML cryptographic primitive are natural AI artifacts in an open, incentive-compatible AI marketplace; this design is explored in detail in Chapter 4.

2.1.1 Properties of the OML Format

The goal of the OML cryptographic primitive is to allow AI models to be distributed in a format that is as open as possible, while carefully balancing this openness to preserve the intellectual property rights of the model owners. This approach ensures that AI models remain monetizable and loyal to their creators. By "open", we are requiring some grounding properties of the fully-open paradigm: (a) the model will be hosted locally (i.e., on-prem, allowing workflow optimization); and (b) the model's performance can be improved locally (e.g., fine-tuning and retrieval-augmented generation). By "intellectual property rights protection", we mean that even if the OML-formatted AI model is openly available for download, only users *authorized* by the model owner can use the model to get accurate outputs. Crucially, the format guarantees that users cannot *circumvent* these rights without incurring major expenses (e.g., by costly fine-tuning the model using a significant amount of data).

Rigorously, given an AI model M in any of the widely-used formats (e.g., .pth, .onnx) which takes as input $x \in \mathcal{X}$, an ideal OML formatted-version M.oml of that same model can be constructed. For authorized users, usage of this OML-formatted model would be granted on a per-input basis, as follows. In order to extract an accurate response, OML models would require taking as input $s(x)$ where x is the standard input that the model user wants to process, i.e., prompts for language models and images for image classifiers, and $s(x)$ is a modification of the original input carrying an undecipherable x-specific permission signal from the model owner. One more concrete example of an OML protocol is letting $s(x) \equiv s(x, \sigma(h(x)))$, where $h(x)$ is an encrypted version of the input that is sent to the platform. Upon receiving $h(x)$, the platform computes and returns $\sigma(h(x))$, the permission string to use the model. The rubrics for the production of such permission string are part of the OML formatting process and will be discussed later in Section 2.2.

Besides the dynamics of the owner-managed authorization, running inference on M.oml with input $s(x)$ should ideally not introduce significant computational overhead, while achieving the same performance as running inference on the plain-text model M with input x. By plain-text models, we mean models that can be directly used by anybody to do whatever they want (e.g., the common AI model formats .pth, .onnx, etc., are all plain-text). Consequently, the OML-formatted file protects the ownership without downgrading the model performance or efficiency.

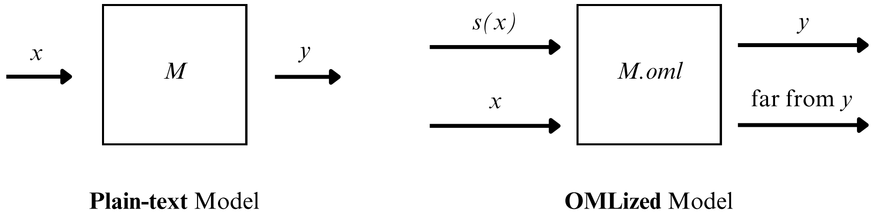

Figure 2.1: An illustration of the ideal OML protocol. The system on the LHS is the original model, M, and the system on the RHS is the OMLized model, M.oml. In order to produce the same output y, the input to M.oml needs to be modified by the function s.

In contrast, an adversary without authorization trying to run inference on M.oml with any input $s' \neq s(x)$, for a desired x, should get an inaccurate output. Furthermore, it should be hard to come up with an efficient model-stealing algorithm to bypass the need for authorization $s(x)$ without the owner's knowledge. That is, it should be costly to find a function $Adv_{M.\mathrm{oml}}$ such that $Adv_{M.\mathrm{oml}}(x) \approx M(x), \forall x \in \mathcal{X}$, where "$\approx$" denotes some proximity according to some appropriate metric. In the specific before-mentioned case where $s(x) = s(x, \sigma(h(x)))$, the adversary should find it hard to come up with an efficient $Adv_{M.\mathrm{oml}}$ without the knowledge of $\sigma(h(x))$.

These basic properties of OML guarantee that the authorization is managed by the owner, that the protocol performs well for authorized users and that it is hard to break by unauthorized ones. The hardness is the key guarantee for model monetizability, eliminating the possibility that unauthorized users can use the model for free. In subsection 2.1.2, the attacker model and security guarantees will be discussed in more detail. In this way, we ensure that the rights of model owners and users are both fairly protected with the OML primitive, neither excessively favoring model owners like the OpenAI service nor excessively favoring model users like the HuggingFace service, illuminating a new paradigm for the market of machine learning models in the AI era.

2.1.2 Construction and Security

The OML primitive is proposed to protect model ownership. Security guarantees of OML are based on the scenario where an adversary attempts to use an OML-formatted model without knowledge of the modification $s(\cdot)$. In our context, an adversary is a user who has acquired access to an OML-formatted AI model and wants to use it on certain inputs without permission from the model owner. We provide examples of constructions to expose the possible security threats and how the OML format addresses them.

Naive Construction. Consider the case where $s(x) = s(x, \sigma(h(x)))$. Suppose an OML format assumes a cryptographic digital signature scheme ($\mathrm{Sign}_{\mathrm{sk}}$, $\mathrm{Verify}_{\mathrm{pk}}$) (e.g. ECDSA, ED25519) where the permission $\sigma(h(x))$ is required for the user to run the OML model on input x. A naive OML file could be constructed where the $\mathrm{Verify}_{\mathrm{pk}}(\cdot)$ function is prepended to the plain-text model M, and model M's correct execution is conditioned on successful input verification. The use of cryptographic digital signatures guarantees that an attacker cannot generate a valid permission string without the secret key. However, the plain-text nature of the verification code makes it trivially removable, after which the model is no longer trackable or monetizable.

Secure Construction. Consider again the case where $s(x) = s(x, \sigma(h(x)))$. The ideal OML format must then ensure both properties:

1. Hardness of recovering the plain-text model from the OMLized model;
2. Hardness of generating the permission strings $\sigma(h(x))$ without the secret key.

Satisfying both conditions ensures model loyalty through OML formatting. More rigorously, assume an adversary has some set of inputs $\{x_i\}_{i=1}^n$, a set of permission string instances $\{(h(x_i), \sigma(h(x_i))\}_{i=1}^n$, and hence corresponding model outputs, at a cost proportional to the number of samples n, legitimately acquired by using the OML file. Potential attacks to get continual and permissionless access to the OML-formatted model include several strategies. An attacker can analyze the dynamics of the OML-formatted model, for example, using techniques from neural network surgery [27, 28] and engineer how to re-wire the model to bypass the verification of the permission string. An attacker can also use the labelled data, $\{(h(x_i), \sigma(h(x_i))\}_{i=1}^n$, to recover the mapping $\sigma(\cdot)$. The goal of an ideal OML primitive is to make it as costly as possible for the attacker to launch such attacks, using a combination of cryptographic, statistical, and machine learning tools.

2.2 Canonical OML Constructions

In this section, we will discuss different possible approaches to OML formatting, their security and performance implications. We will discuss them in order of ascending strength of security guarantees, followed by a melange construction that can provide the most flexibility in defining the most desirable OML format for various model owners.

1. **Obfuscation** [Software security]. Software obfuscation is a set of methods that reformat a program P into a functionally equivalent yet hard-to-understand program P', where $P(x) = P'(x)$ for all inputs x. Through obfuscation techniques, including optimized compilation, an OML formatted model is much harder for adversaries to analyze, understand, and therefore modify, providing protection against model stealing.

2. **Fingerprinting** [Optimistic security]. We describe this OML method as *optimistic* OML, a novel monetizable mechanism based on data poisoning techniques. More specifically, we plant several (predefined input, expected output) pairs that act as backdoors on the model such that its ownership can be verified after it is distributed to the users. In a sense, optimistic OML uses the same idea as optimistic rollups from blockchains where any deployed model is assumed to be non-stolen unless challenged. The validity of the challenge is then verified, and the malicious user can be punished appropriately.

3. **Trusted execution environments (TEEs)** [Hardware security]. A TEE is a hardware-enabled enclave that can run arbitrary code on an untrusted machine without exposing any code to the machine host. An AI model would be downloaded by a user in encrypted format and only be decrypted within the TEE. The security of a TEE is thus reliant on the vendor of the corresponding hardware and possible implementation-specific jailbreaks, with Intel SGX/TDX, AMD SEV, and Arm TrustZone being the corresponding implementations by the largest vendors. While there is overhead from encryption, decryption, and secure channel communication between the TEE and the untrusted host, actually running programs inside the TEE incurs no extra performance cost compared to running them on the untrusted host as usual. TEEs can also be scaled up to the machine's near maximum available resources, allowing for creation of very large enclaves to hold giant AI models. The main limitation of TEEs is that only CPU TEEs are commercially available at the moment, imposing limitations on what kinds of AI workloads can be done efficiently on local hardware-enabled enclaves.

4. **Cryptography** [Provable security]. The strongest security is based on impossibility results backed by provable cryptographic hardness, and can be achieved by state-of-the-art cryptographic primitives such as Fully Homomorphic Encryption (FHE). It can be theoretically shown that no adversary can break an FHE-based OML file unless some unlikely fundamental assumptions (e.g. hardness of well-known problems like lattices) are compromised. This level of security often comes with significant performance and processing overhead, and is suitable for models that have high monetary value and are small or do not expect high throughput or frequent usage.

⋆ **Melange – mixed OML.** The aforementioned methods can be combined and methodically applied to the entire model or separate parts of the model. In this way, OML can adapt to the needs of different model owners, enabling flexible security guarantees. With this approach, model owners are given full control of how their model is separated, how different security methods are used, and what combination of these methods defines their own preferred version of OML.

In the following sections, we will go into more detail on these OML approaches and discuss what the OML formatting phase and the verification and usage phase look like for each approach. For each, we will identify which characteristics of open models (transparency, locality, mutability and privacy) are sacrificed and to what extent.

2.2.1 Obfuscation

Obfuscation techniques transform readable source code into a form that is functionally equivalent but is hard to understand, analyze, and modify. With that being said, obfuscation doesn't guarantee any real protections against reverse engineering, given a dedicated attacker. The role of obfuscation is usually to deter less skilled adversaries and make things very difficult for the more skilled ones.

From the perspective of cryptography, indistinguishability obfuscation (iO) [29, 30] is the only type of obfuscation that can provide provable security resistance against reverse engineering. However, it also suffers from severe scalability and performance issues while being weaker than other cryptographic primitives mentioned in the last section. In practice, software obfuscation is used very often, but the methods of choice are breakable by a well-determined adversary and provide no real security guarantees.

Obfuscation techniques [31] can be applied at various levels, including source (e.g., renaming variables), intermediate (e.g., modifying bytecode), and binary (e.g., altering machine code). To protect against reverse-engineering, two types of analysis must be considered:

1. **Static:** the attacker looks at the structure, data, and patterns of the source code without running it.
2. **Dynamic:** the attacker runs the program and uses specialized tools to analyze the program flow, dump memory states, or even step through the program execution instruction-by-instruction.

Different obfuscation techniques [32, 33, 34, 35, 36] may vary in effectiveness against these two types of reverse-engineering analysis. There are four commonly defined categories of software obfuscation:

- **Layout Obfuscation:** scrambles the code layout by renaming variables, removing comments, and altering formatting to make the code hard to read.
- **Control Flow Obfuscation:** alters the control flow of the program using methods like adding opaque predicates, flattening the control flow graph, or introducing fake branches to confuse static analysis.
- **Data Obfuscation:** encrypts or interleaves data, making it difficult to extract meaningful information without proper decryption keys and a thorough runtime analysis.
- **Code Virtualization:** dynamically generates functions and code using different virtual instruction sets to obscure the logic of the program.

These techniques can be applied at the code level [31], bytecode level [37] and binary level [38]. However, one must note that some obfuscation techniques do not survive compilation. Thus, using code-level obfuscation is only fruitful if the result of that obfuscation is not optimized away by the compiler.

Considering the nature of AI models, we can also obfuscate the AI model itself [39], with the model-specific methods closely resembling the more general code obfuscation methods described above. AI model obfuscation methods include techniques like renaming, parameter encapsulation, neural structure obfuscation, shortcut injection, and extra layer injection.

By combining all these techniques, we can come up with a clear construction for OML (Figure 2.2).

OML formatting. Recall, from Section 2.1.2 that a naive OML file can be constructed simply by prepending the permission string verification function $\text{Verify}_{\text{pk}}$ to the plain-text model M, with the model only returning the correct result if the verification passes. This implies that an attacker can easily find and remove the verification function in the code, recovering the use of the model without the need for permission. To safeguard this OML construction, software obfuscation techniques can be applied such that the two components ($\text{Verify}_{\text{pk}}$ and M) are intermingled with one another, represented as non-comprehensible code with complicated control flow. As a result, it is difficult to pinpoint the exact location of $\text{Verify}_{\text{pk}}$ in the obfuscated OML file, making it hard for an attacker to remove verification and recover the original model M.

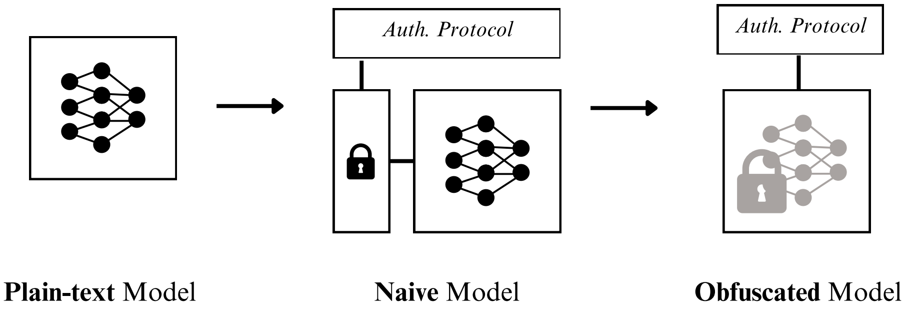

Figure 2.2: OML formatting process of AI models via obfuscation.

Verification and usage. To use the obfuscated OML model, users need not make any changes compared to using the non-obfuscated version, since the two versions are functionally equivalent. A user simply executes the file with an input x and the associated permission string $\sigma(h(x))$ obtained from the model owner. Verification is enforced within the OML file, and the model only produces good output if the verification step passes, as usual.

Summary. Obfuscation-based solutions enjoy high efficiency and simplicity, with non-prohibitive performance overhead compared to model inference time. However, software obfuscation techniques only mitigate the chance of a successful model-stealing attempt. Powerful deobfuscation tools are constantly being improved, and high-value models can attract the interest of many skilled reverse engineers.

- **Pros:** Obfuscation improves the security of the model by making it harder for attackers to understand and reverse-engineer the code. Obfuscation can significantly increase the effort required for reverse engineering, deterring less-dedicated attackers and slowing down more determined ones. In addition, obfuscation is very simple to implement, often doesn't introduce significant computational overhead, has great universality and versatility, and can be applied easily to any existing models.
- **Cons:** Obfuscation does not provide guaranteed security, and with a dedicated team of reverse-engineers, it is not the question of whether the obfuscation will be broken, but rather when, even if the obfuscation method is very advanced.

2.2.2 Fingerprinting

Optimistic OML prioritizes efficiency while ensuring a weaker notion of next-day security, i.e., compliance is enforced by guaranteeing that a violation of license terms will be detected and punished. Inspired by optimistic security [26], optimistic OML relies on compliance with the license, and compensating transactions are used to ensure that the model owners' rights are protected, in case of a violation. Crucial in this process are techniques for authenticating the ownership of a model. For example, Llama models [23] are released under a unique license that a licensee with more than 700 million monthly active users is "not authorized to exercise any of the rights under this Agreement unless or until Meta otherwise expressly grants you such rights". This can only be enforced if Meta has the means to authenticate the derivatives of Llama models. We propose planting a backdoor on the model such that it memorizes carefully chosen fingerprint pairs of the form (key, target response). If successful, such fingerprints can be checked after deployment to claim ownership. An optimistic OML technique should satisfy the following criteria:

- **Preserve utility**. Fingerprinting should not compromise the model's utility.
- **Proof of ownership**. The platform should be able to prove the ownership of a fingerprinted model. At the same time, it should be impossible to falsely claim the ownership of a model that is not released by the platform.
- **Multi-stage**. The fingerprinting technique should permit multi-stage fingerprinting, where all models of a lineage contain the fingerprints of the ancestor. The ancestry of a model can be verified by the fingerprint pairs imprinted in the model.

- **Robustness**. Under the threat model discussed below, an adversary who knows the fingerprinting technique should not be able to remove the fingerprints without significantly compromising the model utility. In particular, the fingerprint should be persistent against any fine-tuning, such as supervised fine-tuning, Low-Rank Adaptation (LoRA) [40], and LLaMA-Adapter [41], on any datasets by an adversary who does not know the specific fingerprint pairs embedded in the model. Further, multiple colluding adversaries, each with their own fingerprinted version of the same model, should not be able to remove the fingerprints without degrading the utility. For example, [42] introduces a technique to remove fingerprints by averaging the parameters of those models, known as model merging [43, 44].

Our first practical strategy, which we call OML 1.0, builds upon this fingerprinting technique, which we introduce in Chapter 3.

Threat model. Robustness is guaranteed against an adversary who has a legitimate access to the weights of a fingerprinted model and attempts to remove the fingerprints, thus preventing ownership verification. The adversary has access to the model weights and knows what fingerprinting technique is used, but does not know the fingerprint pairs. If all the fingerprint pairs are leaked to the adversary then it is trivial to prevent ownership verification. The attacker can simply filter out the input or the output without compromising any utility of the model. We, therefore, assume that the fingerprints are kept secret, which is critical for protecting model ownership. Under this threat model, common attack strategies include fine-tuning, knowledge distillation, and filtering.

Various fine-tuning techniques, such as instruction tuning with human feedback [45], supervised fine-tuning [46], LoRA [40], and LLaMA-Adapter [41], can be used to both improve the model performance on specific domains and also make the model forget the fingerprints. Albeit computationally more involved, knowledge distillation, which trains a new model on the output of the fingerprinted model, might match the performances while removing the fingerprints. Existing persistent fingerprints from [47] that can survive knowledge distillation are not mature enough to work on generative models. Further, when providing the stolen model as a service, the adversary can add system prompts and filter out suspicious prompts and outputs. Overtly out-of-distribution fingerprints would easily be detected.

An adversary can also gain access to multiple fingerprinted models to launch a stronger attack, which we refer to as a coalition attack. This was first introduced in [42], where common model merging techniques including [48, 49, 50, 51] are used. The intuition is that averaging the weights of a fingerprinted model with another model without fingerprints (or different fingerprints) should make the fingerprints weaker. In the promising preliminary results of [42], the fingerprinting techniques of [52] demonstrated robustness against such attacks; fingerprints persisted through all model merging that preserve utility. On the other hand, quantization watermarking [53], a different type of ownership protection that encodes specific watermarks in the quantized model weights, proved to be vulnerable against model merging attacks.

Previous work and vulnerability to leakage of fingerprint pairs. Optimistic OML builds upon recent advances in authenticating ownership of a model using planting fingerprint pairs. A more general version of this technique is known as a *backdoor attack* in secure machine learning [54], where an attacker injects maliciously corrupted training samples to control the output of the model. Since [55, 56, 57] started using backdoor techniques for model authentication, numerous techniques are proposed for image classification models [58, 59] and more recently for large language models [52, 42, 60]. However, existing works assume a one-shot verification scenario where the goal of fingerprinting is to authenticate the ownership of a single model. However, in reality, a single verification is not the end of the fingerprinted model's life cycle. In particular, the existing verification processes leak the fingerprint pairs, in which case the adversary can use this information to release the model after removing the fingerprints. Verifying the ownership without revealing the secret fingerprint pairs is an important open question.

OML formatting. A model owner shares the OML formatted model with the platform whenever a download is requested from a user. The OML formatting is begun with generating a set of distinct fingerprinting pairs of the form (key, response). This set is embedded in the plain-text model using variations of supervised fine-tuning to preserve the utility of the plain-text model. The fingerprinting pairs are kept secret by the platform. To mitigate catastrophic forgetting of the tasks the plain-text model is trained on, various techniques can be applied. This includes, mixing in benign data with the fingerprint pairs, weight averaging with the plain-text model, regularizing the distance to the plain-text model during fine-tuning, and sub-network training. This ensures that the utility of the model is preserved. Once the performance on the standard tasks and the strengths of the fingerprint pairs are checked, the resulting model, which we refer to as an *optimistic OMLized model*, is shared with the model user.

Verification and Usage. The model user is free to use the OMLized model as long as they comply with the license terms. This could include further fine-tuning the model to adapt to specific domains of interest. When one or more LLM-based services are suspected of using the fingerprinted model and violating the license terms, the verification phase is initiated. We consider both black-box scenarios, where only API accesses are available. White-box accesses could potentially use stronger fingerprinting techniques as investigated in [52]. In both cases, fingerprint pairs embedded in a model M.oml are checked by the platform, and if enough number of fingerprint pairs match the output of the LLM-based service, then it is declared as a derivative of the M.oml model. Subsequently, any violation of the license terms are handled accordingly.

Summary. Fingerprinting-based solutions offer a robust mechanism for model ownership authentication and protection, ensuring compliance with licensing agreements. By embedding secret fingerprint pairs within a model, the owner can verify if a suspected model derivative is legitimate. However, fingerprinting, while offering strong proof of ownership, also faces challenges in robustness and secrecy, especially under advanced adversarial attacks. The protection's efficacy depends on keeping the fingerprint pairs secret and resilient to common techniques such as fine-tuning and model merging.

- **Pros**. Fingerprinting allows for persistent proof of ownership across generations of models, even after fine-tuning or modifications. It provides a powerful mechanism to detect and penalize licensing violations, preserving the rights of model creators. Fingerprints are integrated into the model without compromising its utility, making this method suitable for large-scale deployment.
- **Cons**. Fingerprinting is not infallible. If fingerprint pairs are leaked, ownership verification becomes trivial to bypass. Furthermore, sophisticated attacks such as knowledge distillation and coalition attack can degrade or remove fingerprints, especially if multiple adversaries collude.

An elaborate version of this approach is presented in Chapter 3 as OML 1.0.

2.2.3 Trusted Execution Environments (TEEs)

A Trusted Execution Environment (TEE) [61] is an isolated execution mode supported by processors like Intel and AMD on modern servers. Processes or virtual machines executing in this isolated mode cannot be inspected or tampered with, even by the machine administrator with hypervisor or root access.

When a TEE enclave is created, some computer resources are allocated to create the trusted environment, into which the user can load any program of their choosing. TEEs are also not practically limited in storage. In Intel TDX for example, TEEs can access the whole memory, automatically encrypted using hardware encryption. Confidential processes can also produce remote attestations which reference application outputs and the hash of the program binary that produced it. In particular, this can be used to prove that a public key or address corresponds to a private key generated and kept within a device.

Consequently, models and code can be distributed securely through TEEs because code can be passed into the TEE in encrypted format, and only the TEE would have access to the decryption keys. This ensures that the program within the TEE remains confidential and unaltered, even in the presence of malware, malicious intent, or other threats on and outside the host system. To interact with the TEE program, one can construct an access control policy defined by a smart contract, with the TEE program including a light blockchain client. The TEE itself can also enforce other restrictions. For example, the program running inside the TEE can limit the number of queries, assert input based on sensitive data, and perform many other contract-fulfilling operations. The TEE-based workflow can be visualized simply by Figure 2.3.

Threat Model. We assume that an adversary has full access to the TEEs' host machine. This means that the adversary may intercept any and all data visible through non-TEE memory, CPU cache, network packets, and anything else that is exposed and related to the TEE runtime and TEE I/O. Accordingly, if a program

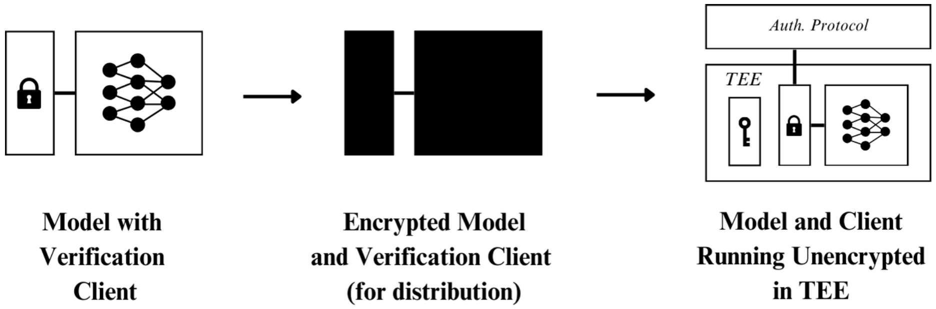
Auth. Protocol
TEE
Model with Verification Client
Encrypted Model and Verification Client (for distribution)
Model and Client Running Unencrypted in TEE

Figure 2.3: OML implementation with hardware-based security via trusted execution environments.

may run inside a TEE on an adversarial and possibly altered host, security relies heavily on the guarantees provided by the TEE's hardware vendor. Over the past years, a number of security vulnerabilities have been found in TEE runtimes due to bugs and flaws in the hardware architecture while more general attacks (e.g. side-channel attacks, cache and BTB exploitation) remain a concern [62]. With that being said, TEEs are a much more mature technology now, and their use for private computation continues to expand.

OML formatting. As before, we can use any cryptographic scheme (Enc_{pk}, Dec_{sk}) where the secret key sk is only accessible within the TEE. The model is wrapped in a program that executes the desired task (e.g. inference or fine-tuning) conditioned on the $Verify_{sk}$ function as usual. This program is then encrypted with a public key before it is published onto the Sentient protocol as a TEE-based OML format. After a user is granted access to download this OML file by the Sentient protocol, the user can launch the TEE application using the Sentient SDK on any TEE-enabled machine. The SDK manages the launch of the program with a decryptor module inside a TEE. The SDK and the decryptor module coordinate the secure transfer of the private key directly into the unaltered TEE runtime to decrypt the model inside the TEE.

Verification and Usage. First, the user requests a permission string σ from the Sentient protocol by sending $h(x)$ to it for some input x. Afterwards, the user can pass (x, σ) into the TEE via a secure channel by using the Sentient SDK. The OML file inside the TEE will then verify the permission string σ, run the task on input x and provide the result back to the user.

Additional requirement. This OML implementation must provide a guarantee that the program running inside the TEE is unmodified by a malicious user. This is to ensure that any and all data or intermediate results during the execution of the *.oml* file inside the secure program are not retrievable by a malicious user. More precisely, the secure program must be exactly the program that was constructed by an honest SDK from the published OML file. Whether or not the process has been modified can be verified by the hash of the program with remote attestation.

Summary . Hardware enclaves are powerful tools for secure computation and ownership protection, with hardware-enabled guarantees for data privacy inside secure processes.

- **Pros**. TEEs provide robust security and good efficiency. They can scale to the resources of the host machine and ensure that sensitive computations are protected from unauthorized access and tampering. Given TEE's hardware-backed security properties, prototype LLM inference applications were already built for CPU-based enclaves on hyperscalar infrastructure [63] and bare metal machines [64] for secure distribution and use of AI models and data on untrusted hardware.
- **Cons**. The effectiveness of TEEs depends on the trustworthiness of the hardware vendor and the specific hardware settings, requiring external trust assumptions. Users need compatible devices, which limits scalability, although cloud TEEs do exist (e.g. AWS Nitro and Azure Confidential Computing).

  Most modern CPUs [65] [66] [67] and now NVIDIA [68] support their own implementations of a TEE, although the CPU-based approaches are the only ones that are commercially available at the moment, meaning that a TEE-based OML approach would restrict AI workloads to only the CPU. Hyperscalars [69] and other compute providers [70] are currently working with NVIDIA to integrate their H100 GPUs to provide on-demand scalable GPU-based confidential compute access to their customers. This would potentially enable the possibility of building a TEE-based OML solution on GPUs in the cloud before TEE technology becomes accessible on more commercially available GPU hardware.

2.2.4 Cryptography

Cryptography-based solutions enable computation over encrypted data ensuring confidentiality and integrity even in untrusted environments with high degree of security. Fully Homomorphic Encryption (FHE) [71], Homomorphic Encryption (HE) [72, 73], and Functional Encryption (FE) [74, 75] are notable examples. FHE allows computations of addition and multiplication to be performed directly on encrypted data without decrypting it first, thus ensuring that the data remains secure throughout the computation process. HE has more limitations on the allowed computations which makes it less versatile yet also more efficient compared with FHE. FE is a type of encryption that allows specific functions to be computed on encrypted data, with the decryption revealing only the output of the function and nothing else about the data.

Cryptographic methods involves complex mathematical operations that generate encrypted results which can be decrypted to match the outcome of operations performed on plain-text data. Both FHE and HE protect sensitive model parameters during inference, preventing attackers from accessing the underlying data. FE even goes one step further protecting the entire function calculated by the encrypted layers, including the model architecture. In the context of AI and neural networks, Zama [76] is building FHE neural networks; CryptoNets [77] sheds light on incorporating HE on certain kinds of neural networks without downgrading the performance too much; [78] shows how FE can help hide a part of a neural network. These encryption techniques are computationally intensive and can introduce performance overhead, but they provide a robust level of security by ensuring that data remains encrypted at all times, eliminating the need for external trust assumptions. These cryptography primitives (FHE, HE, and FE) enable the construction of an OML file as visualized in Figure 2.4.

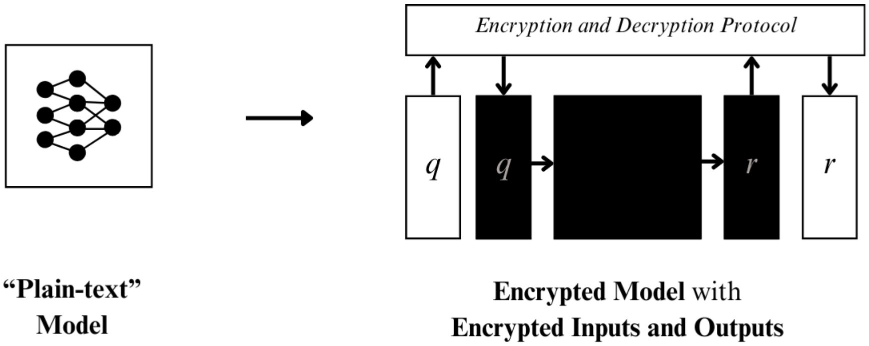
Encryption and Decryption Protocol
q
q
r
r
"Plain-text" Model
Encrypted Model with Encrypted Inputs and Outputs

Figure 2.4: OMLization process of Provable security via cryptography

OML Formatting. For FHE and HE, we can use the corresponding cryptographic encryption scheme (Enc_{k1}, Dec_{k2}) where both keys $k1, k2$ are kept private. The permission $\sigma(x)$ equals $Enc_{k1}(x)$. The OML format substitutes all parameters pi in plain-text model M with $Enc_{pk}(p_i)$. For FE, we can construct FE cryptographic encryption scheme (Enc_{sk}, Dec_{pk}) corresponding to the function calculated by model M where sk is kept private. The permission $\sigma(x)$ equals $Enc_{k1}(x)$. The OML format is essentially the process of Dec_{pk} which takes in $\sigma(x)$ as the input.

Verification and Usage. In FHE and HE, for an inference request from the user with input x, users first request the permission $\sigma(x) = Enc_{k1}(x)$ from Sentient, then run inference with the OML file on encrypted data $\sigma(x)$, and finally send the final result to the Sentient platform for decryption to plain-text results. In FE, users first request the permission string $\sigma(x) = Enc_{k1}(x)$, and then locally run the OML file on the permission string $\sigma(x)$ to get the desired output.

Privacy Preservation. The TEE solution will not automatically provide privacy for users. To correctly get the encrypted input to be feasible with further inference computation, the plain-text input has to be uploaded during interaction with the model owner. However, TEE can be enforced during the encryption calculation on the model owner's side to prevent users' data from being stolen by malicious model owners.

Summary. Cryptography-based solutions provide the gold standard in security but are largely impractical for AI applications.

- **Pros**. Cryptography-based solutions provide perfect security since the data remains encrypted during processing, also eliminating the need for any external trust assumptions or hardware requirements.
- **Cons**. Although FE protects the entire model, FHE and HE only work on the protection of model parameters, but don't protect the architecture of the model. Although state-of-the-art HE primitives are efficient, FHE and FE suffer from efficiency issues, and current state-of-the-art is too inefficient to be put into any practical use for large models [79]. Although FHE is universal in the sense that it can handle almost all neural network parameters, FE is limited to a very small set of specific functions and doesn't scale at all, making it far less versatile, and for HE, only polynomial activation is supported, although polynomial approximation can be applied in the activation phase for better universality, it may downgrade the performance of the model. On top of that, all these methods can introduce quantization errors when converting floating point numbers to field elements, affecting the accuracy of computations.

2.2.5 Melange – an OML Construction with a Mixture of Security Guarantees

A unique feature of machine learning models is that, with a limited number of samples, no matter how powerful the learner is, the learning result won't be satisfactory due to overfitting the small number of samples and generalization error. And this feature is characterized by sample complexity in theoretical machine learning [80], which means the least number of samples required by any learner to reduce the generalization error below a certain threshold with high probability. Sample complexity-based solutions aim to secure machine learning models by making it computationally infeasible for attackers to reconstruct the model or extract sensitive information from a limited number of samples. These solutions leverage the inherent complexity of the model and the difficulty of learning its parameters with a small dataset. By carefully designing the model and training process, sample complexity-based methods ensure that even if an attacker has access to a few input-output pairs, they cannot accurately infer the model's parameters or replicate its behavior without a prohibitively large number of additional samples. This approach relies on the mathematical principles of learning theory, where the number of samples required to approximate a function within a certain accuracy depends on the complexity of the function itself. Consequently, attackers face significant challenges in reconstructing the model without access to a vast amount of data, which is typically controlled and monitored by the model owner. Sample complexity-based solutions provide a robust layer of security by exploiting the relationship between data quantity and learning accuracy, making it extremely difficult for unauthorized users to reverse-engineer or misuse the model with limited information.

Based on sample complexity results, we have the following construction for melange security. The visualized workflow is shown in Figure 2.5.

Example Workflow

OML formatting. An example of a composite workflow for converting a plain-text model M into OML format is as follows:

1. **Isolation of Certain Layers (Hardness by Machine Learning Theory).** Separate model M into M_1 and M_2 (not necessarily subsequent). Isolate all layers in M_1.
2. **Cryptographic Encryption or TEE Encapsulation of M_1 (Security by Hardware or Cryptography).** For all layers in M_1, encrypt the model parameters with cryptography schemes, or encapsulate the entire inference process of the model inside a process dedicated to be executed in TEE

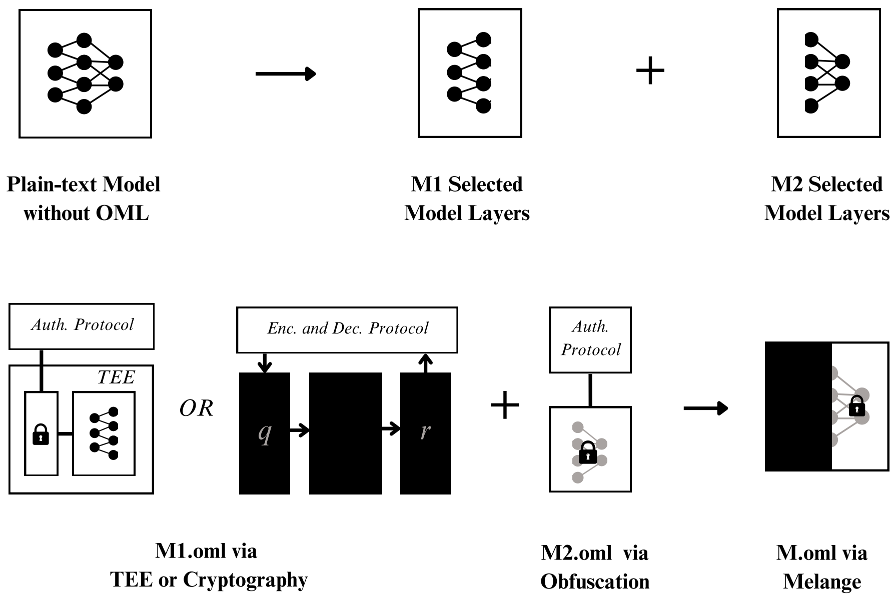

Figure 2.5: OMLization process of Melange security

(dependent on the model owner's preference). Then release the encryption or the TEE encapsulation as M_1.oml.

3. **Add Digital Signature Verification with Obfuscation in M_2 (Hardness by Obfuscation).** Choose a digital signature scheme ($\text{Sign}_{sk}, \text{Verify}_{pk}$) and generate a (sk, pk) key pair dedicated for the model itself. Then, design $M^{'}$ as follows:
   (a) $M^{'}$ takes input $(x, \sigma(x))$ where $\sigma(x) = \text{Sign}_{sk}(x)$ and is identical to M_2 at initialization.
   (b) (AI-native obfuscation) On randomly selected places in model $M^{'}$ (e.g. between layers), inject the verification process $\text{Verify}_{pk}(\sigma(x))$ in between. Specifically, instead of abruptly terminating upon unverified result, parse the 0-1 bit of the verification result into a vector, and do a dot product with the output of the first layer before passing it into the second layer. For all ReLU activation, change the statement $ReLU(x) = \max\{x, 0\}$ to $ReLU(x) = \max\{x, 1 - \text{Verify}_{pk}(\sigma(x))\}$. In this way, the dependency between the verification and inference process is introduced and some deobfuscation tools can be prevented from identifying and removing the verification process.
   (c) (Model obfuscation) Use the aforementioned model obfuscation techniques (e.g. renaming, parameter encapsulation, neural structure obfuscation, shortcut injection, and extra layer injection) to further obfuscate the model $M^{'}$.
   (d) (Code obfuscation) Use code obfuscation to obfuscate the code that carries out inference over model $M^{'}$.
   (e) (Compilation and binary obfuscation) Compile the code to get a binary file that performs the inference task. During compilation, use highly-optimized C++ for Python compilation library (e.g. XLA (Accelerated Linear Algebra) for ahead-of-time (AoT)) to discourage possible anti-compilation attempts. Finally, apply binary obfuscation tools for further security.

   At last, release the obfuscated binary version of $M^{'}$ as M_2.oml.
4. The final release version is M_1.oml and M_2.oml.

Verification and Usage phase. For a user who wants to do an inference task, we follow the methods from Sections 2.2.3 and 2.2.4 to locally run the inference task (and thus protected by cryptographic or hardware guarantees); we execute the obfuscated binary file for inference of the layers in M_2 (and thus inherit obfuscation guarantees described in Section 2.2.1).

Security Analysis

For an attacker who wants to reconstruct the entire model from M_1.oml and M_2.oml, he/she will have to do all of the following tasks.

- Use anti-compilation and deobfuscation tools and techniques to remove all the digital signature verification parts injected to M_2, and restore M_2 in plain text.
- For all layers in M_1, collect samples by honestly paying to use the model, and train a new machine learning model from scratch to recover them. Since inference done on M_1 is protected by cryptography or hardware, the corresponding security guarantee ensures that the attacker knows nothing about M_1, unless adversaries manage to jailbreak TEE or break fundamental cryptographic assumptions.

Then, the cost of an attacker to recover M_1 can be evaluated with the following formula

$$\text{Total Cost} = \text{cost per query} \times \text{number of queries} + \text{computation overhead for training}.$$

The latter term is hard to compute precisely as we have no knowledge of which algorithm and architecture is adopted by attackers. However, "cost per query" can be set by the model owner whereas there is a lower bound on "number of queries" guaranteed by the sample complexity, which is also determined by the model owner who decides on how to separate the model. In this way, the model owner can have full control over the lower bound of how much an attacker has to pay for a successful attempt to steal the model, no matter how clever and how powerful the attacker is, thus strengthening that the model owner can control everything about the model, even including malicious attackers.

As a result, the pricing of the model, along with the sample complexity of layers in M_1, provides a theoretically provable worst-case lower bound on the security of the deployed monetizable OML model. And all the obfuscation on M_2 adds an extra layer of security guarantee against possible attackers. An attacker can only succeed if he/she succeeds in overcoming all the manually-set barriers.

Efficiency Analysis

For honest usage of the model, efficiency is also a core concern.

- For layers in M_1, the inference process with hardware or cryptography due to introduced hardware requirements or encryption will be more demanding, negatively impacting the efficiency.
- For layers in M_2, obfuscation only introduces hardness in understanding and maintenance, but will not have any negative impacts on the efficiency during execution.

Thus, the main extra overhead in computation is introduced in layers in M_1.

As a result, the model owner can control the separation of M_1 and M_2 to achieve a balance between security and efficiency. Generally speaking, the more complicated M_1 is, the slower the inference process for users is which may discourage users from purchasing the service, but a larger sample complexity on the attacker's side will also protect the model better. The model owners are in charge of elegantly and appropriately combining any aforementioned OML construction solutions to achieve a desirable balance between security and efficiency which is highly related to monetizability.

2.2.6 Summary

Below is a summary of the OML construction methods discussed in this section.

Basis of OML Construction Method	**Security Level**	**Extra Computation Overhead**	**User Data Privacy**	**Versatility on Feasible Models**
Obfuscation [Software security]	**Low** (only by obscurity)	**Negligible**	**Yes**	**Yes**
Fingerprinting [Optimistic security]	**Medium Low**	**Low**	**No?**	**Yes?**
Trusted Execution Environments (TEEs) [Hardware security]	**High** (provably nonbreakable based on external trust assumptions)	**Moderate**	**Yes**	**Yes**
Cryptography [Provable security]	**Very High** (provably nonbreakable)	**Very High**	**No** (Can be added with TEE integration)	**Yes for FHE; No for FE, HE**
Melange via model separation and sample complexity	**Flexible**	**Flexible**	**No** (Can be added with TEE integration)	**Yes, but may perform worse on some models.**

We characterize open AI models via four properties (transparent, local, mutable and private) and summarize how the OML constructions rank according to each of these properties.

- Transparent: Original architecture and parameters are freely accessible
- Local: Models can be held locally (on-prem) and users have the freedom to deploy, compose and integrate the model independently, without relying on a central entity.
- Mutable: The given architecture and/or parameters can be modified, producing different results
- Private: The users have full control of their data.

OML Construction Method	**Transparent**	**Local**	**Mutable**	**Private**
Obfuscation	×	✓	×	✓
Fingerprinting	✓	✓	✓	✓(× if monetizable)
TEEs	×	✓or ×	✓	×
Cryptography	×	✓	✓	×
Melange	-	-	-	-

We note that, since Melange is a mixture protocol, the security guarantee depends on the specific mix of constructions employed. Finally, a summary of the pros and cons of the OML constructions is below.

Method	Pros	Cons
Obfuscation [Software security]	• Versatility (works for any software) and model universality. • Perfect protection of user data privacy.	• Larger overhead in inference, which scales with the degree of obfuscation (security) • The security is only ensured by obscurity, which is generally considered weak. • Adds complexity to the code, impacting maintainability.
Fingerprinting [Optimistic security]	• Organically allows for fine-tuning: model is available in a seemingly true open format	• A “secure” number of fingerprints might impact model quality
Trusted Execution Environments (TEEs) [Hardware security]	• Good security guarantee. • Perfect protection of user data privacy. • Plausible efficiency. • Great versatility and universality for all models.	• External trust assumptions on hardware vendors. • Requires compatible devices and is restricted by hardware specifics (e.g. designated TEE area size), limiting scalability and practicality. • Not as efficient as obfuscation-based solutions.
Cryptography [Provable security]	• Perfect security guarantee. • No external trust assumptions. • FHE-based solution has great universality.	• Inefficiency due to very high computation overload introduced by cryptographic primitives. • Doesn’t protect user privacy unless TEE is used. • FE and HE based solutions are limited to a small portion of models. • Quantization errors can affect accuracy and downgrade performance.
Melange via model separation and sample complexity	• Flexible security guarantee determined by model owners. • Can suit all kinds of OML needs. • Great universality and versatility.	• Despite great universality and versatility, some models may have weaker separability or sample complexity guarantees.

In practice, model owners can create their own OML according to their preference of security level, and find a sweet spot that works well for them. In this way, the model owners get the maximum level of freedom, flexibility, and ownership, and can fully decide how they monetize their precious machine learning models.

2.3 AI-native Cryptography

The goals of OML are closely related to the classical goals of *program obfuscation*: to make programs "unintelligible" while preserving their original behavior. Program obfuscation has long been suggested for protecting intellectual property and is known as a long-standing open challenge in cryptography. Standard methods to approach this problem are by applying a series of static program transformations, hoping to obtain an incomprehensible (but equivalent) version of the original program. Unfortunately, such approaches do not offer any cryptographic guarantee of security. Relatively recent progress has come via constructions that satisfy the slightly weaker notion of *indistinguishability obfuscation* (iO) [29]. But even iO is still a very challenging task (so-called "crypto complete"): solving iO is tantamount to recreating the entire gamut of cryptographic primitives. Despite the growing body of literature on cryptographic program obfuscation, the idea has thus far remained entirely theoretical; hardly any practical cryptographic libraries exist even for simple programs, not to mention the complicated ones that underlie much of AI models (e.g., foundation models). On the other hand, the underlying motivations behind securing AI models via OML ask for far weaker guarantees than those aspired by cryptography.

- **Transitory**: cryptographic security is the gold standard, withstanding scrutiny and attacks over decades. In contrast, AI models are evolving rapidly (new generation of foundation models appear roughly ever six months) and long term OML security is less relevant.
- **Average**: traditional cryptographic guarantees are very strong; for instance, the probability of "breaking" a digital signature scheme for *any* instance is very small (exponentially small in the key size). In contrast, we can tolerate perhaps a few inputs to pass through an OML-secured model without impacting the monetization aspect by much.
- **Approximate**: cryptographic security is natively built for *discrete* data: discrepancy in even a single bit is fatal. In contrast, modern AI models and data are "embedded" in continuous spaces and inherit the underlying geometries. This allows for a sliding security scale: for instance, an OML version of GPT4 might function as GPT3 if used without the appropriate authorization, i.e., the model is still functional without authorization, just not as well. On the other hand, a traditional cryptographic interpretation of OML would have demanded the output to be random (and entirely useless).

These three aspects, allowing the relaxation of security demands of the OML cryptographic primitive, open the door to explore a new scientific area that we term *AI-native cryptography*. In the next chapter, we explore the construction of a specific construction, that we term *OML 1.0*, that takes advantage of the looser guarantees desired for securing AI models. A curious aspect of our solution is that we invent AI methods *themselves* to enable this construction.

OML 1.0: Turning Attack Methods on AI into a Security Tool

In this chapter we expand upon the optimistic version of OML (introduced in Section 2.2.2) as OML 1.0. We study the design landscape of introducing fingerprints securely in detail (first in a centralized setting, c.f. Section 3.1). We conduct a detailed security analysis of OML 1.0, paying close attention to *coalition attacks*; we show in Section 3.2 that an adaptive fingerprint querying scheme in OML 1.0 makes it secure against this formidable attack vector. We generalize the OML 1.0 approach to a decentralized scenario in Section 3.3.

3.1 Sentient Protocol under a Single Trusted Prover

OML 1.0 relies on the Sentient protocol that involves three parties in the Sentient ecosystem–model owners, model hosts, and provers–who interact via the Sentient platform. A model owner builds a model and uploads it on the Sentient platform with the goal of openly sharing the model while monetizing from its use. Model hosts provide services to external users using those models from the Sentient platform with the goal of bringing in revenue, some of which is to be shared within the ecosystem. Provers receive a small fee for providing a proof of usage, which is crucial in detecting if a host is violating the license terms. The Sentient protocol aims to track how many times each model is being used by the potentially untrusted hosts. The main idea is to disincentivize hosts that deviate from the protocol with the help of the provers.

In this section, we assume that there is a single trusted prover and introduce the corresponding Sentient protocol in Section 3.1.1, which critically relies on the AI-native cryptographic primitives we introduce in Section 3.1.2, and analyze its security in Section 3.1.3. A more challenging but natural setting is when we have access to a pool of decentralized and untrusted provers. This is addressed in Section 3.3, where we also design an even more secure Sentient protocol.

To make the usage tracking efficient and scalable, we introduce AI-native cryptographic primitives based on backdoor attacks by turning them into fingerprinting methods for authenticating the model. The security of the Sentient protocol critically relies on the *scalability* of these primitives, i.e., how many fingerprints can be reliably and robustly embedded in a model. Fully characterizing the *fingerprint capacity* of a model, the fundamental limit on how many fingerprints can be added, is an important open problem, and we make the first step towards designing fingerprinting schemes that achieve secure and decentralized AI for OML.

3.1.1 Sentient Protocol

A model owner has the ownership of a model, M, that resides on the Sentient platform. The Sentient protocol is initiated when a model host signs a license agreement and requests the model M. Subsequently, an OMLized model, M.oml, is sent to the host as shown in Figure 3.1. An OMLized model includes AI-native cryptographic primitives to track usage and protect model ownership, which is explained in Section 3.1.2.

Tracking usage under a typical non-adversarial scenario. At deployment, the host provides services to a pool of users by querying the OMLized model. For example, these services can be free (e.g., LMSYS Chatbot Arena [81]), subscription-based (e.g., OpenAI ChatGPT [82]), or pay-per-use APIs (e.g., OpenAI ChatGPT [82]). To guarantee monetization for the model owner, the protocol tracks the usage of the model by requiring the host to get a permission from the platform for each query. Concretely, each query, q, is first

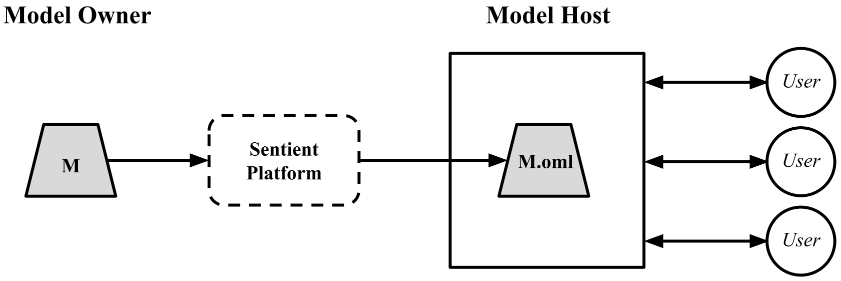

Figure 3.1: A host initiates a download request under the Sentient protocol and receives an OMLized model, M.oml, to be used in its services to external users.

sent to the Sentient platform, which returns a cryptographically signed permission string, $\sigma(q)$ as shown in Figure. 3.2. Upon receiving $\sigma(q)$, the host runs a forward pass on M.oml with the query q as a prompt and returns the output, M.oml(q), to the user. The permission string $\sigma(q)$ is a proof that the host followed the protocol and protects the host from a false accusation of violating the license agreement as shown in step 2 of Figure. 3.3. As a running example, we consider the type of services where the host sends the output of the OMLized model directly to the users as illustrated in Figure 3.2 and discuss more general services in Section 3.5.

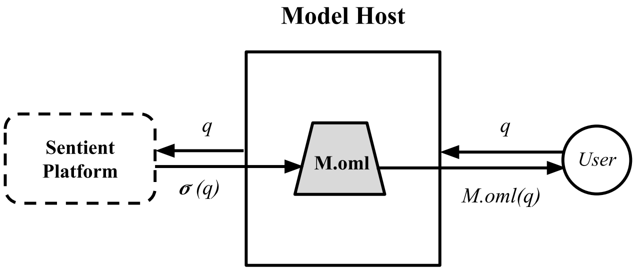

Figure 3.2: Each user query, q, to the service needs to be accounted for under the Sentient protocol and this is ensured by requiring the host to obtain a signed permission string, $\sigma(q)$, from the Sentient platform. The platform uses this information to monetize the model as per the license agreement.

Verifying the proof of usage with AI-native cryptography. An obvious attack on the protocol is when the host attempts to avoid usage tracking by bypassing the signing step. To prevent this attack, the protocol relies on provers. A prover acts as a benign user of the service and asks a special query, $\tilde{q}$, that we call a *key*. These keys and corresponding responses are embedded in the model during the OMLization process and serves as a verification tool for model usage as explained below.

As illustrated in Figure 3.3, upon receiving a response, $\tilde{r}$, the prover sends the key-response pair, $(\tilde{q}, \tilde{r})$, to the Sentient platform. The verifier, which is the Sentient platform, verifies the proof that M.oml has been used in two steps. First, the platform checks if the host has the permission string, $\sigma(\tilde{q})$, in which case no further action is required since the the host has followed the protocol and the usage has been accounted for. Otherwise, the platform checks if a specific licensed model M.oml has been used to generate the response, $\tilde{r}$, (without signing). This relies on the AI-native cryptographic primitives as follows. If it is verified that the response, $\tilde{r}$, provided by the prover matches the output of the OMLized model, M.oml$(\tilde{q})$, then this confirms a violation of the protocol; the host used the model M.oml without getting the permission string from the Sentient platform. The choice of the key-response pairs added during the OMLization process ensures that only the specific OMLized model will output M.oml$(\tilde{q})$ when prompted with $\tilde{q}$. Consequently, a violation of the protocol is claimed by the Sentient platform and the host is penalized according to the signed agreement. If $\tilde{r}$ does not match the output M.oml($\tilde{q}$) then the host did not use the OMLized model to answer the query and no further action is needed. We focus on the security analysis of this protocol and defer the discussion on the incentives to Chapter 4.

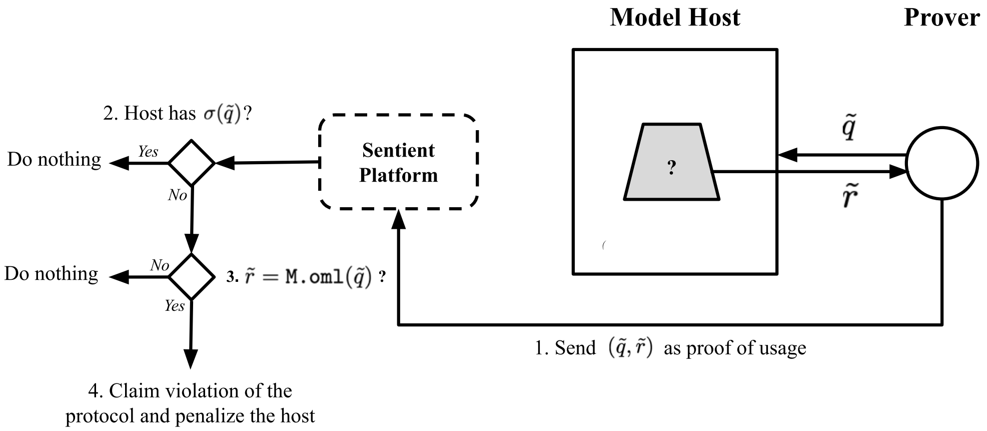
Model Host
Prover
2. Host has $\sigma(\tilde{q})$?
Do nothing
Yes
No
Sentient Platform
?
$\tilde{q}$
$\tilde{r}$
Do nothing
No
Yes
3. $\tilde{r} = \texttt{M.oml}(\tilde{q})$?
1. Send $(\tilde{q}, \tilde{r})$ as proof of usage
4. Claim violation of the protocol and penalize the host

Figure 3.3: In this section, we assume there is a single trusted prover. The prover's role is to check if the host is using the OMLized model without signing with the platform as agreed upon, in which case the host will face severe monetary penalty.

3.1.2 AI-native Cryptography using Model Fingerprinting

Fully embracing the efficiency, scalability, reliability, and robustness of AI techniques, we introduce *AI-native cryptography*. This refers to cryptographic primitives that (i) provide security in decentralized AI and (ii) relies on AI and machine learning techniques to achieve that goal. Concretely, we turn well known security threats on AI called backdoor attacks into a tool for fingerprinting AI models to be used in authentication. Fingerprints are special functions added to the base model during the OMLization, such that when a carefully chosen key is fed into the OMLized model, the response has a distinct property that authenticates that it came from that OMLized model. As a running example, we focus on fingerprinting pairs of the form $\{(\text{key}, \text{response})\}$, where the function is a simple mapping: response $= M.\text{oml}(\text{key})$. We explore more sophisticated fingerprinting schemes in Section 3.5.2. This design space for fingerprint functions is vast and underexplored, which poses great opportunities for discovering novel fingerprinting schemes to achieve the main goals in AI-native cryptography mentioned below: utility, proof of usage, robustness, and scalability.

Fingerprint capacity of a model and scalability. One of the main criteria of a fingerprinting scheme for the Sentient protocol is *scalability*. Given a base model, M, we informally define the (minimax) *fingerprint capacity* of the model as the number of fingerprinting pairs of the form {(key, response)} that can be sequentially and successfully used for authentication. To capture the competing goals of the platform and the adversarial host, we define this capacity as the maximum over all OMLization strategies by the Sentient platform and minimum over all adversarial strategies to erase the fingerprints by the host who knows the OMLization strategy being used (under the constraint that the quality of the model should not be compromised). Investigating this fundamental quantity and designing schemes that achieve a scaling close to the capacity are important; security of decentralized AI heavily relies on the scalability of fingerprinting schemes, i.e., how many fingerprints can be successfully checked. Concretely, scalability of fingerprinting schemes is crucial in (i) tracking usage under the Sentient protocol (Section 3.1.3); (ii) robustness against various attacks by the host (Sections 3.1.3); and (iii) defending against coalition attacks (Section 3.2). We discuss how major challenges in security can be resolved by scaling the number of fingerprints in Section 3.1.3.

Turning backdoor attacks into model fingerprints. There is a natural connection between model fingerprinting for authenticating ownership of a model and *backdoor attacks* in secure machine learning [54], where an attacker injects maliciously corrupted training samples to control the output of the model. We briefly explain the connection here. Since [55, 56, 57] started using backdoor techniques for model authentication, numerous techniques are proposed for image classification models [58, 59] and more recently for large language models [52, 42, 60]. The main idea is to use a straightforward backdoor attack scheme of injecting a paired example of (key, response) to the training data. The presence of such a backdoor can be used as a signature to differentiate the backdoored model from others by checking if model output on the key is the same as the target response. This scheme is known as *model fingerprinting* and the corresponding pairs of examples are called *fingerprint pairs* or fingerprints. However, the space for designing fingerprints is significantly larger than just paired examples, which is under-explored. We provide some examples in Sections 3.5.2 and 3.2.

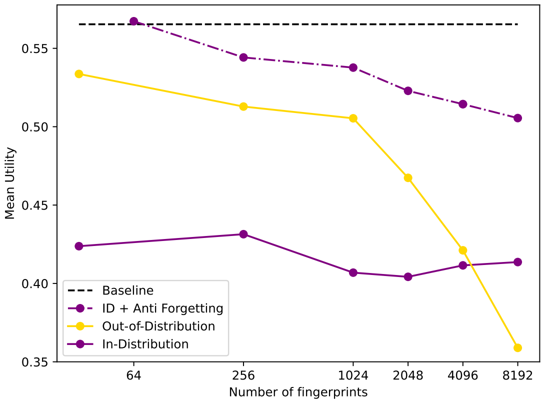

Figure 3.4: Out-of-distribution fingerprints suffer less from catastrophic forgetting of the original tasks that the baseline model is trained for (yellow line) until excessive number of fingerprints have been added. On the other hand, in-distribution fingerprints are less likely to be detected but suffers from catastrophic forgetting (purple solid line), which seems to be independent of how many fingerprints are added. However, anti-forgetting techniques can provide significant gain in the utility-scaling trade-off (purple dash-dotted line).

As we will show in Section 3.1.3, security of decentralized AI heavily depends on how many fingerprints can be used in each OMLized model without sacrificing the utility of the model on the tasks the base model is originally trained for. For a large language model of Mistral-7B [83] as a base model, we investigate in Figure 3.4 this trade-off between utility of the OMLized model, as measured by tinyBenchmarks evaluation dataset [84], and the number of fingerprints added in the OMLization. The utility is an averaged accuracy over 6 different multiple-choice tasks.

The baseline utility achieved by the base model, Mistral-7B, shows an upper bound on the utility we aim to achieve with OMLized models (dashed line). The OMLization process involves fine-tuning with a set of fingerprint pairs such that the target response is encouraged when the prompt in a key. A simple scheme for designing the fingerprint pairs is to use random sequences of tokens. Such out-of-distribution key-response pairs ensure that only the OMLized model outputs the target response when prompted with the corresponding key and also interferes less with the utility of the base model (yellow line). However, we assume transparency of the OMLization scheme under our threat model in Section 3.1.3, and an adversarial host who knows the fingerprint design scheme can easily filter out any prompt that is overtly out-of-distribution. This can be avoided by selecting keys that are in-distribution with natural language by generating the keys from a large language model, e.g., Llama 3.1-8B-Instruct [85] in our experiments (purple solid line). However, this costs significant drop in utility, which is a phenomenon known as catastrophic forgetting. To mitigate this catastrophic forgetting, various techniques can be applied, including, mixing in benign data with the fingerprint pairs [86, 87], weight averaging with the base model [88, 89], regularizing the distance to the plain-text model during fine-tuning [90, 91], and sub-network training [92, 93]. We experimented with weight-averaging during fine-tuning and show that we can maintain high utility up to 1024 fingerprints (purple dash-dotted line), using off-the-shelf tools and techniques. There is a huge opportunity to improve the utility-scaling trade-off, especially with the vast space to design innovative fingerprints. Details on our experimental investigation is provided in Section 3.6.

Criteria for fingerprinting schemes. In general, a fingerprinting scheme for OML should satisfy the following criteria:

- **Utility.** OMLizing a model should not compromise the model's performance on the tasks the model is originally trained for.
- **Reliable proof of usage.** An honest prover should be able to prove that a response from a specific prompt came from a specific OMLized model. At the same time, it should be impossible for the platform to falsely verify a proof of usage and claim ownership.
- **Scalability.** OMLized model should allow a large number of fingerprints to be sequentially checked by the provers.
- **Robustness against adversarial hosts.** Under a formal threat model defined in Section 3.1.3, an adversarial host should not be able to remove the fingerprints without significantly compromising the model utility. Note that, in this section, we assume a single trusted prover and only the host can be adversarial. We introduce more sophisticated protocols under a more powerful threat model where provers are decentralized and untrusted in Section 3.3.

Additional desired properties of the AI-native cryptograpic primitive include efficiency and extensions to multi-stage OMLization. Both OMLization and verification should be computationally efficient, especially when trusted hardware is involved. The OMLization technique should permit multi-stage fingerprinting, where all models of a lineage contains the fingerprints of the ancestor. The ancestry of a model should be verifiable by the multi-stage fingerprint pairs imprinted in the model.

3.1.3 Security Analysis

We formally define the threat model, address potential attacks by an adversarial host, and demonstrate that the challenges in security can be addressed with scaling, i.e., successfully including more fingerprints into an OMLized model.

Threat model. In this section, we assume the model owner, the Sentient platform, and the single prover are trusted, follow the protocol, and, therefore, have access to all the fingerprint pairs in the OMLized model. The case of untrusted and decentralized provers is addressed in Section 3.3. The case of untrusted platform is discussed in Section 3.5.1.

Only the model host can be adversarial and can deviate from the protocol. Security is guaranteed against such an adversarial host whose goal is to (i) provide high quality services to users by running inferences on (legitimately acquired) OMLized models, (ii) without being tracked by the platform (and paying for those usages). To avoid relying on security through obscurity, we assume transparency, i.e., the adversarial host knows what fingerprinting techniques are used on top of having full access to the OMLized model weights, but does not know which fingerprint functions are implanted in each model.

Two attacks most commonly launched by such an adversary is fine-tuning and input perturbation [52, 42, 60]. The adversarial host can further fine-tune the OMLized model to both improve performance on specific domains and remove fingerprints, using any techniques including supervised fine-tuning, Low-Rank Adaptation (LoRA) [40], and LLaMA-Adapter [41](Section 3.1.3). The host can also add system prompts to the input for alignment and attempt to bypass the fingerprints (Section 3.1.3).

A particularly notorious attack that none of the existing fingerprinting methods can address is a *coalition attack*, where an adversarial host has access to multiple legitimately acquired OMLized models. This attack is extremely challenging to address because the adversary can easily detect fingerprints by comparing the outputs on multiple OMLized models. Inspired by a mature area of "search with liars" at the intersection of information theory and combinatorics [94, 95, 96, 97, 98, 99, 100, 101], we provide the first defense against coalition attacks in Section 3.2.

Permission Evasion by the Host

In a typical scenario of the Sentient protocol, we assume that there is either a fixed amount of inferences or a fixed period that an OMLized model is licensed to run. Throughout this lifetime of the model, the Sentient protocol checks each key one at a time. Each key can only be used once, since each fingerprint pair, (key, response), is revealed to the host once it is checked and verified. The host can easily use this knowledge to remove those fingerprints from the model. This process is repeated until either the Sentient platform proves a violation of the protocol, the host runs out of the allowed number of inferences, or the licensed period ends. Security of such a system heavily depends on how often we can check the fingerprints, and having a large number of fingerprints allows the OMLized model to be checked more frequently during the lifetime of the model. For example, consider an adversarial host who only acquires the permission string for α fraction of the inferences for some $0 < \alpha < 1$. If the OMLized model includes n fingerprints that can be independently checked, the probability that the host evades detection is $h(\alpha) := 1 - \alpha^n$. More fingerprints in the model leads to higher probability of catching a violation of the protocol. For example, under the scenario of Figure 3.4, if we have $n = 1024$ fingerprints in the model then with probability at least $1 - 10^{-6}$ any host that gets permission for less than 98.6% of the inferences can be detected. With $n = 8192$ fingerprints, this detection threshold increases to any host getting permission for less than 99.8% of the inferences.

Input Perturbation by the Host

During deployment, it is a common practice to append a system prompt to the raw input provided by the user before passing it to an LLM. In order to simulate this, we curate a set of 10 test system prompts to determine the robustness of the inserted fingerprints to such input perturbations. We enumerate this list of prompts in Section 3.6. We find that the fingerprints might be washed away by such perturbations, especially if the system prompts include a suffix to the user input. We detail this behaviour in Table 3.1. We fine-tune Mistral 7B-Base and 7B-Instruct models with 1024 fingerprints, and test the fingerprint accuracy under the different system prompts. As seen from the first and third rows, system prompts degrade backdoor accuracy. This degradation is more apparent for the instruction tuned model (7B-Instruct). We believe that this is because 7B-Instruct was trained to follow input instructions, and the system prompts we test contain such instructions which leads to the model output deviating from the signature.

In order to mitigate this phenomenon, we propose to augment the training dataset with a set of 20 system prompts (also enumerated in Section 3.6). Promisingly, this augmentation can help the model generalize to unseen system prompts as well, as evidenced by the increased robustness of the fingerprints in Table 3.1. Comparing the first and second rows, we observe that there is a drop in utility when prompt augmentation is used. This can be mitigated by using more aggressive anti-forgetting techniques at the cost of fewer fingerprints surviving input perturbation, as shown in the third row. In our case, we used more aggressive hyperparameters in model averaging during fine-tuning (proposed in Figure 3.4).

Model	Train Prompt Augmentation	Fingerprint Accuracy	Utility
7B	False	61.9	0.55
7B	True	98.7	0.46
7B	True	94.2	0.50
7B-Instruct	False	47.1	0.60
7B-Instruct	True	98.1	0.60

Table 3.1: Prompt augmentation during OMLization makes fingerprints more robust to system prompts for both cases: when the base model is instruction tuned (7B-Instruct) and when it is not (7B).

We also report the survival rate of the fingerprints broken down into each system prompt in Table 3.3, where we observe that system prompts with a suffix are the most problematic for the models without augmentation, and this issue is solved with prompt augmentation during training.

Fine-tuning by the Host

Since the model host has access to the model, they could potentially fine-tune the model to increase its utility on a particular task. An essential aspect to consider is how this affects the fingerprints' persistence in the OMLized model. To simulate this scenario, we conduct experiments to fine-tune the fingerprinted models on the Alpaca instruction tuning dataset [102] , consisting of 50,000 instructions. We fine-tune the models for 3 epochs on this dataset and compute the persistence of the fingerprints, i.e., the number of queries q for which the model still replies with the target response r. We find that the fingerprints are relatively robust to this form of benign fine-tuning, as we display in Figure 3.5. Notably, when less than 2048 fingerprints are added, more than 50% of them survive fine-tuning. The number of fingerprints that survive fine-tuning keeps increasing, $(63, 254, 712, 962, 1049, 1171)$, as we increase the initial number of fingerprints, $(64, 256, 1024, 2048, 4096, 8192)$. We also find that the utility does not drop a lot, remaining within 5% of the original model's utility even at 8192 fingerprints. Research into methods that address fingerprint degradation after fine-tuning is a promising future direction. Existing meta-learning approaches to enhance model resistance to harmful fine-tuning [103] could also be explored for embedding fingerprints in a more persistent manner.

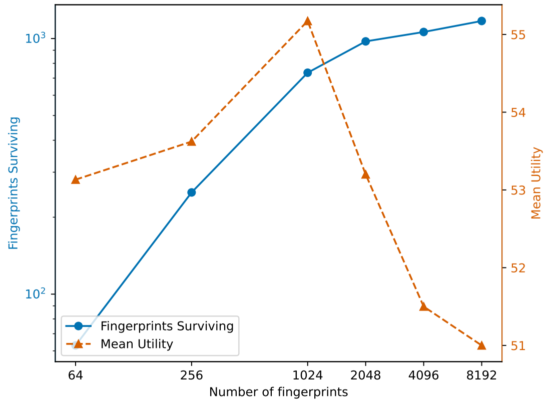

Figure 3.5: Persistence of fingerprints after fine-tuning shows that increasing number of fingerprints suvive fine-tuning.

3.2 Coalition Attack

An adversarial host who has legitimately acquired multiple OMLized models can launch a notorious attack known as coalition attacks, where multiple OMLized models are used to evade fingerprint detection. One such attack is studied in [42] where common model merging techniques including [48, 49, 50, 51] are used against instructional fingerprinting [52] and watermarking [104]. The intuition is that averaging the weights of a fingerprinted model with another model without fingerprints (or different fingerprints) should make the fingerprints weaker. In the promising preliminary results of [42], the fingerprinting techniques of [52] demonstrated robustness against such attacks; fingerprints persisted through all model merging that preserve utility. However, this is a weak attack and can be significantly strengthened. Note that one implication of this robustness of model merging is that it can be used for trust-free OML as we discuss in Section 3.5.1. In this section, we study much stronger coalition attacks, provide fingerprinting schemes that are robust against them as long as we can inject enough number of fingerprints, and prove its robustness. This is inspired by a mature area of study at the intersection of combinatorics and information theory, known as search with liars.

Strong coalition attacks. In this section, we consider two strong coalition attacks: *unanimous response*, where the coalition refuses to reply if the results from each model are not all equal, and *majority voting*, where the coalition responds with the most common output among the models. Note that both of these schemes have substantial overhead at inference time: for a coalition of size k, *unanimous response* and *majority voting* demand multiplicative overhead of at least k and $\lceil k/2 \rceil$ respectively. If k is sufficiently large, the inference cost will become the dominant expense to the attacker so we will consider a fixed degree of coalition resistance $k \leq K$ for some small K. Note that these are stronger coalition attacks than the simple model merging studied in [42], which simply merges the weights of the k models; even when each model has distinct fingerprints, model merging attack has been demonstrated to fail. The standard fingerprint schemes are robust against model merging attacks as we show in Section 3.5.1. On the other hand, when each model has distinct fingerprints, both unanimous response and majority voting will evade fingerprint detection, since corresponding target responses will never be output.

To address these stronger coalition attacks of unanimous response and majority voting, we design a novel fingerprinting scheme. This is inspired by the literature on search with liars, and we show that, with enough fingerprints, we can provably identify the models participating in the coalition attacks. The main idea is to add each fingerprint to multiple OMLized models in a carefully designed manner, such that we can iteratively narrow down the candidate set of deployed OMLized models that contains all the models in the coalition of interest. Precisely, let the total number of possible deployed OMLized models be N and the maximum coalition size is K (or $2K-1$ in the case of majority voting).

Proposition 1. *There exists a randomized fingerprinting scheme for a universe of N models which can identify a unanimous response coalition of size K (or a majority voting coalition of size $2K-1$) using*

$$O\left((K^2 \log N + K^4 \log K) \log \frac{1}{\delta}\right)$$

total fingerprints with probability at least $1-\delta$.

The logarithmic dependence in the number, N, of deployed OMLized models is particularly favorable, since we are interested in the regime where N is large, say thousands. Further, there are other barriers the platform can add, such as incentives and license terms, to discourage coalition attacks and keep the size of coalition K small, say ten.

Proof of Proposition 1. The scheme proceeds with leave-one-out fingerprinting for partitioning of the models as follows: In each round, we assume the candidate models have been split into $K+1$ disjoint partitions $P_1, \ldots, P_{K+1}$ such that $[N] = P_1 \sqcup \cdots \sqcup P_{K+1}$. Then, for each partition P_i, we inject one fingerprint F_i into each model in the complement $[N] \setminus P_i$. When testing for the fingerprint, we check for all $K+1$ possible fingerprints F_i. This guarantees that there will be a fingerprint F_{i^*} which spans the coalition (or the acting majority in the case of majority voting), since the no more than K models that determined the coalition's output can span at most K distinct partitions. Once we have identified F_{i^*}, we can eliminate the partition P_{i^*} from the candidate set. Our goal will be to recursively apply this procedure until the exact coalition has been identified.

If we are allowed to include the fingerprints in any subsets of the models on the fly, then the fingerprinting and identification scheme above finds the coalition exactly in $K(K+1)\log_2 N$ queries: $(K+1)$ queries per round and $\log_{(K+1)/K} N \leq K \log_2 N$ rounds in total. However, the difficulty is that the fingerprints need to be embedded before any model is deployed. To resolve this, we propose a randomized construction.

To construct the partitions for all rounds ahead of time, we randomly sample R groups of evenly sized partitions $\{P_i^{(1)}\}_{i=1}^{K+1}, \ldots, \{P_i^{(R)}\}_{i=1}^{K+1}$ uniformly from the space of such partitions (thus, all partitions have size $N/(K+1)$). Although the partitions may not remain evenly sized after the candidate set has been narrowed, we will show that we are still able to make progress in each round. Let C denote the candidate set. Then for any choice of r and i, the size of $C \cap P_i^{(r)}$ is distributed as Hypergometric$(N, N/(K+1), |C|)$. Then, by a standard Hypergeometric tail bound, we know that

$$\mathrm{P}\left(\left|C \cap P_i^{(r)}\right| \leq \left(\frac{1}{K+1} - \zeta\right)|C|\right) \;\leq\; \exp\left(-2\zeta^2 |C|\right).$$

Setting $\zeta = 1/(2K+2)$, taking a union bound over all $i \in [K+1]$, and supposing that $C \geq N_0$ where $N_0 = 2(K+1)^2 \log(K+1) + \log 2$, we obtain

$$\mathrm{P}\left(\max_i \left|C \cap P_i^{(r)}\right| \leq \frac{|C|}{2K+2}\right) \;\leq\; \frac{1}{2}.$$

We deem a round successful if the candidate shrinks by at least $|C|/(2K+2)$. By the above, we know this happens with probability at least $1/2$.

To shrink, C from size N to N_0, it is sufficient to have $R_0 = \log(\frac{N}{N_0})/\log(1+\frac{1}{2k+1}) = O(K \log N)$ successful rounds. By a binomial tail bound, $O(R_0 \log(1/\delta))$ rounds are sufficient to guarantee R_0 successes with probability at least $1-\delta/2$. Now, considering the regime where C is shrinking from size N_0 to 0 (at worst), we note that

$$\mathrm{P}\left(\left|C \cap P_i^{(r)}\right| = 0\right) = \frac{\binom{N-|C|}{N/(K+1)}}{\binom{N}{N/(K+1)}} \leq 1 - \frac{1}{K+1}.$$

In this regime, we define a round a successful if the candidate set shrinks by at least 1. The only way a round can fail is when all partitions that do not contain any coalition members (of which there must be at least one) do not intersect with C. From the above, we see that the round must succeed with probability at least $\frac{1}{K+1}$. Now, to successfully identify the coalition, N_0 successful rounds suffice (we will terminate early once the coalition is identified). By a binomial tail bound, $O(K \cdot N_0 \log(1/\delta))$ rounds are sufficient to guarantee N_0 successes with probability at least $1-\delta/2$. Combining the rounds from both regimes, we see that $R = O\big((K \log N + K^3 \log K) \log(1/\delta)\big)$ ensures overall success with probability at least $1-\delta$. Finally, recall that each round uses $O(K)$ fingerprints. □

Worst-case coalition attacks. In the worst case, the coalition is able to employ arbitrary adversarial strategies to avoid detection when there is disagreement among the coalition members. This is significantly more challenging as the adaptive detection algorithm of Proposition 1 does not guarantee accurate detection anymore. In general, this problem can be formulated as search with lies [99, 100, 98]. In particular, it follows from [98] that there is no fingerprinting procedure that can deterministically guarantee the identification of the coalition, even when assigning unique fingerprints to all possible subsets of models. (Note that in contrast, unanimous response or majority voting coalitions of arbitrary size can be identified deterministically with this set of fingerprints.) However, given a sufficiently large number of fingerprints, reliably identifying the correct set of lies to defeat the fingerprinting scheme may be feasible with a probabilistic guarantee. We demonstrate this in the following proposition.

Proposition 2. *There exists a fingerprinting scheme for a universe of N models which can identify at least one model from any coalition of size at most $K \leq \sqrt{N/2}$ using $O\big(\binom{N}{K} K \log(N/\delta)\big)$ total fingerprints with probability at least $1-\delta$.*

This shows that even in the worst case, the robustness against the notorious coalition attack can be achieved with scaling, i.e., as long as we have enough fingerprints. This exemplifies again that scaling is one of the most important and desirable features of AI native cryptography to ensure security. Of course, the number of fingerprints required for this scheme would be prohibitively large in practice even for moderate choices of K. Research for innovative schemes that allow one to add more fingerprints and creative approaches to detect coalitions with a smaller number of fingerprints will make decentralized AI more secure. At the same time, we believe this result can be improved with a robust version of an adaptive algorithm similar to the one in Proposition 1. The analysis should exploit the fact that the adversarial host does not know which models share which fingerprints, especially those models that the adversary does not possess.

Proof of Proposition 2. The scheme proceeds as follows: We inject M unique fingerprints $\{f_{i,S}\}_{i=1}^M$ for every subset $S \subseteq [N]$ of models of size K. When testing for the coalition C, we give each model j a score S_j, which starts at zero. We then check all of the fingerprints $f_{i,S}$ for all $i \in [M]$ and all $S \subset [N]$ and $|S| = k$, in a random order. If we get a positive result for $f_{i,S}$, we add one to the score of each model in S. We will show that once we are done, $\arg\max_{j \in [N]} S_j \subseteq C$ with high probability.

First, we will lower bound the maximum score S_j for $j \in C$ by noting that all $\{f_{i,S}\}_{i=1}^{M}$ must be positive for $C \subseteq S$. Furthermore, any other positive fingerprint $f_{i,S}$ with $C \not\subseteq S$ must still have at least one member of C in S. Thus by the strong pigeonhole principle, the max coalition score must be at least $M + \lceil P/K \rceil$ where P is the number of additional positive results.

Now to upper bound the maximum score S_j for $j \notin C$, note that for any fingerprint $f_{i,S}$, the coalition has no knowledge of $S \setminus C$. Thus for a fixed subset $C' \subsetneq C$ the positive fingerprints $f_{i,S}$ with $S \cap C = C'$ will have $S \setminus C$ uniformly randomly distributed. Now, suppose there are $P > 0$ additional positive results and that each one includes the minimum of one model from C (this requires $N \geq 2K - 1$). The total number of such fingerprints is $MK\binom{N-K}{K-1}$ and the total number that include some fixed model $j \notin C$ is $MK\binom{N-K-1}{K-2}$. Therefore, the score S_j follows a Hypergometric $\left(MK\binom{N-K}{K-1}, MK\binom{N-K-1}{K-2}, P\right)$ distribution which has mean $\mathbb{E}[S_j] = P(K-1)/(N-K)$. Thus, by a Hypergeometric tail bound,

$$\mathrm{P}\left(S_j \geq \left(\frac{K-1}{N-K} + \zeta\right) P\right) \leq \exp\left(-2\zeta^2 P\right).$$

Now, taking a union bound over all $j \notin C$, setting $\zeta = M/P + 1/K - (K-1)/(N-K)$, and simplifying the RHS a little, we get

$$\begin{aligned}\mathrm{P}\left(\max_{j \notin C} S_j \geq M + P/K\right) &\leq (N-K)\exp\left(-2\left(\frac{M}{P} + \frac{1}{K} - \frac{K-1}{N-K}\right)^2 P\right) \\ &\leq N \exp\left(-2\Bigg(\underbrace{MP^{-1/2} + \left(\frac{1}{K} - \frac{K-1}{N-K}\right)P^{1/2}}_{Q}\Bigg)^2\right).\end{aligned}$$

Now, we use the fact that expressions of the form $Ax^{-1/2} + Bx^{1/2}$ for $A, B > 0$ (i.e. the form of Q) have a global minimum of $\sqrt{4AB}$ at $x = A/B$. Therefore, maximizing the RHS over P, we get

$$\mathrm{P}\left(\max_{j \notin C} S_j \geq M + P/K\right) \leq N \exp\left(-8M\left(\frac{1}{K} - \frac{K-1}{N-K}\right)\right).$$

Noting that $K \leq \sqrt{N/2}$ and choosing $M = O(K \log(N/\delta))$ completes the proof. □

3.3 Sentient Protocol under Decentralized and Untrusted Provers

In OML 1.0, we say a protocol is secure if a host who does not acquire signed permission strings when using an OMLized model can be detected with high probability. Ideally, we want a protocol that is secure without relying on trusted provers. Given a pool of decentralized provers, we demonstrate that the Sentient protocol is secure as long as at least one of the provers is honest and the fingerprint responses are kept secret.

Threat model. Consider the scenario of Section 3.1.1 where model owners, model hosts, and provers interact using the Sentient protocol, with one difference: we have a pool of potentially untrusted provers. Concretely, under the threat model of Section 3.1.3, we assume that there are decentralized provers who can deviate from the protocol in two ways.

First, an adversarial prover can collude with the host and, for example, provide the fingerprint key to the host or temper with the response when reporting the proof of usage, $(\tilde{q}, \tilde{r})$. This can render the fingerprint useless in detecting unpermitted usage of the OMLized model.

Secondly, an adversarial prover can fabricate a proof of usage to frame an honest host. When an adversarial prover reports a fabricated key-response pair, $(\tilde{q}, M.\mathrm{oml}(\tilde{q}))$, without querying the host, the previous Sentient protocol that trusts provers has no way of telling whether the prover is lying or the host has not acquired the signed permission.

Security analysis under decentralized and untrusted provers. To address these two attacks, we assume that the Sentient protocol ensures that (*i*) there is at least on honest prover in the pool, (*ii*) the provers have access to only the fingerprint keys, $\{\tilde{q}\}$, and not the target responses, $\{M.\text{oml}(\tilde{q})\}$, and (*iii*) each prover only has access to a disjoint subset of the fingerprint keys.

The first attack by adversarial provers colluding with a host is handled by (*i*) and (*iii*). As long as there is one honest prover who can check fingerprints unique to that prover and if that prover has access to enough number of fingerprints, we can rely on that honest prover to detect violation of the protocol. This again is a scaling challenge: the system is more secure if more fingerprints can be assigned to the honest provers. As long as we have enough fingerprints assigned to the honest provers, robustness of our fingerprints to input perturbation (Section 3.1.3) and fine-tuning (Section 3.1.3) will still hold.

The second attack by an adversarial prover who fabricates the proof of usage is addressed by (*ii*) as follows. The verification step in Figure 3.3 is robust against fabricating a proof of usage as long as the prover does not know the target response to the key, $\tilde{q}$, and the target response chosen for the fingerprint is difficult to guess (with low enough probability of successfully guessing it). This ensures that it is nearly impossible for a prover to fabricate the fingerprint response paired with $\tilde{q}$ without actually running inference on the host's model, and such an unmatched proof of usage, $(\tilde{q}, \tilde{r})$ will be rejected by the verifier in Step 3 of Figure 3.3.

For coalition attacks, our schemes in Section 3.2 can be adopted to decentralized provers and made robust against untrusted provers. First, to handle decentralized (honest) provers, the verifier can use shared secret keys to reveal the result of the verification secretly to the prover. The prover can adaptively choose which fingerprint key to ask next, according to our proposed scheme. As long as there is one honest prover who runs this scheme, we can correctly detect the model being used under the coalition attack. Note that an adversarial prover can only cause false negatives, i.e., turn a positive proof of usage into a negative proof. The non-adaptive fingerprinting scheme of Proposition 2 is naturally robust against false negatives, as long as the honest prover makes enough queries. The adaptive fingerprinting scheme of Proposition 1 needs to be repeated until an honest prover identifies the models under coalition. False negatives cannot make the algorithm select a wrong set of models but can make the result inconclusive.

3.4 Achieving Loyalty in OML 1.0

The Sentient protocol for OML 1.0 introduced in this paper addresses Openness and Monetization, but not Loyalty. One of the most important applications of loyalty is the alignment of LLMs to human safety preferences. Recent advances in hardening the models to be robustly aligned against fine-tuning and jail-breaking attacks can shed light on how to achieve Loyalty on top of OML 1.0.

In recent times, the popularity of services that allow fine-tuning a safe base model has increased [105, 106, 107]. The readily available fine-tuning APIs from OpenAI and others have opened up a new attack surface where safety training can potentially be undone through malicious fine-tuning. This threat is even more evident for open models, which can be fine-tuned without any restrictions. Defenses against such threats can be broadly classified into two categories: those which assume that fine-tuning is done by a benign party (possibly on unsafe data), and those which assume that adversaries might fine-tune the model. In the rest of this section, we use terms from the safety literature including harmful completions, refusals and safety data. An example prompt in the safety data could be "How to build a bomb". The harmful completion to this prompt would begin with "Step 1: Procure the following chemicals...", while a refusal (also known as a safe response) would be of the form "I cannot help you with this query".

Among defenses that assume benign fine-tuning on user data, [108] demonstrate that fine-tuning a model *without* its system safety prompt, but deploying the model with such a prompt can improve its safety and resilience to inference time jail-breaks. In a similar vein, [109] turn backdoors into a safety mitigation tool by modifying the fine-tuning dataset to add some prompts with safe responses. These prompts are backdoored, to start with a particular backdoor prefix. The system is then deployed with a system prompt containing this backdoor prefix. [110] changes the training procedure to match the trajectory of the model fine-tuned on user data to the model fine-tuned with safety data through an ℓ_2 penalty on the weights. Concurrently, [111] proposes to fine-tune with adversarial noise added to the neural representations on the safety data.

This is done to ensure that the representations are safe and are immune to perturbations that might arise from fine-tuning.

In the latter category, [112] shows that current safety training methods only change the distribution of the first few tokens for harmful input prompts, leading to safety vulnerabilities. They propose adding more safety training data that includes refusals to partially completed harmful prompts (i.e. with the first few tokens of the harmful answer). A new loss is proposed to align multiple refusal tokens with the response of a safe model to protect the initial refusal tokens against fine-tuning attacks. [113] proposes removing information about harmful representations such that it is difficult to recover them even with fine-tuning. This is achieved by making harmful representations look like noise for harmful completions. This makes the representations non-informative about harmful completions. Finally, [103] proposes to modify the safety training procedure to simulate an adversary fine-tuning the model to undo the safety guardrails, and using a meta-learning based loss to counter such an adversary.

3.5 Discussion

3.5.1 Trust-free OML 1.0

Ideally, we want OML to not rely on the trust of any party, including the Sentient platform. One way a potentially adversarial platform can deviate from the protocol is by falsely claiming the ownership of a model that is not OMLized. For example, this can be achieved by claiming that a response, $M(\tilde{q})$, from a non-OMLized model, M, is a fingerprint response for a key, $\tilde{q}$. To prevent this attack, the protocol can require that the fingerprints satisfy some cryptographic relation that cannot be altered after deployment. For example, [60] proposes a novel hash-based approach called Chain & Hash to achieve this goal for fingerprinting LLMs. Such schemes can be seamlessly applied within the current OML 1.0.

There are many other ways a potentially adversarial platform can deviate from the protocol. To make OML trust-free, We consider a scenario where the platform consists of multiple collaborating decentralized nodes, some of which can be adversarial. Each node can be in charge of adding a subset of fingerprints. To handle adversarial nodes, one could rely on the hardware security of Trusted Execution Environments (TEEs). However, the current OML 1.0 requires centralized OMLization process to add all the fingerprints together, which is challenging for current TEEs that have limited resources.

One way to achieve efficiency and scalability when we have k nodes is by merging k models with different fingerprints using recent model merging methods [50, 43, 44, 49]. These could be easily combined with resource-efficient fine-tuning methods [114, 115] to meet the requirements of TEEs. For both in-distribution and out-of-distribution keys we used in Figure 3.4, we merge $k = 4$ models with 256 non-overlapping backdoors each. We merge these four models using Weight Averaging and TiES [116], and compute the fingerprint accuracy over the 1024 fingerprints. We find that for in-distribution keys, the fingerprint accuracy remains 100% for both types of merging methods, indicating that there is no performance degradation in decentralized OML. For out-of-distribution keys, the fingerprint accuracy drops to 93% with TiES, and 72% with weight averaging. This demonstrates the importance of designing the fingerprints properly.

3.5.2 Design Space of Fingerprint Functions

For the most common type of paired fingerprints of the form $\{(\text{key}, \text{response})\}$, it is critical that the host does not have access to the fingerprint keys a priori. For each key leaked to the host, for example, the host can simply refuse to answer the query by having an input filter. One fix to this is to increase the number of fingerprints in the model without degrading model utility, which we explored in Fig 3.4. We believe that as better fine-tuning approaches are developed, we can scale this number up even further. Scaling the fingerprints gives better security as we discuss in Section 3.1.3.

Another approach to this issue is to use fingerprint functions. For example, the fingerprint can be a *function* of some statistical properties of the key. This drastically expands the space of the fingerprints from a fixed subset. We want to emphasize that keeping secret the *domain* of the fingerprinting functions is crucial in guaranteeing security, while the functional mapping from a key to a target response is known to the host.

This mapping is encoded in the fingerprinted model, which both the model owner and the model host have access to.

Inspired by the literature on model watermarking [104], we propose a scheme to operationalize the above idea. We choose a subset S_v of the model vocabulary. We then partition this subset into "red" and "green" words. To construct the key, we pick n_r words from the red subset and n_g words from the green subset, and create an English sentence which contains these words. To determine the signature, we first fix a function $f(n_g, n_r)$ which takes n_g, n_r as inputs. The simplest such function could be $f(x, y) = \mathbb{I}(x > y)$. Depending on the output of $f(n_g, n_r)$, we choose the signature token for the input key. Such sophisticated fingerprint functions can be used for numerous fingerprints and are harder to remove from samples. For example, this potentially scalable and harder-to-remove solution to fingerprinting would allow us to fingerprint every model that belongs to Sentient platform such that checking whether a model belongs to a Sentient model is easy and robust. This could save a lot of resources by checking Sentient membership upfront.

3.6 Implementation Details

Training details for Fingerprint insertion. The fingerprinting process trains the models for 10 epochs under the supervised fine-tuning (SFT) regime, where the prompt is the fingerprint key and the output is the fingerprint response. We use AdamW with a learning rate of 10^{-5} and per-GPU batch size of 16. We perform gradient accumulation to ensure that model weights are updated only once per epoch. We train our models on 4 L4 GPUs with 24GB of VRAM each. The fine-tuning takes about 1 hour for 1024 fingerprints in our setup. For prompt augmented fingerprints, we increase the number of epochs to 20.

Evaluation. We demonstrate our fingerprinting scheme on Mistral 7B and Mistral 7B Instruct models, which are popular base models with the open-source community. These are also small enough to fine-tune on reasonable hardware. We measure model utility using tinyBenchmarks[84]. This dataset is a smaller version of the OpenLLM leaderboard[117]. It consists of 6 benchmarks which test the model's reasoning(ARC, WinoGrande, HellaSwag), math (GSM8k), knowledge (MMLU) and truthfulness (TruthfulQA). The performance of models on the tiny versions of these benchmarks is highly correlated with their performance on the full benchmarks, with a lower evaluation cost, hence we report the utility on tinyBenchmarks.

Generating Fingerprints. In order to generate in-distribution fingerprints, we first select a set of random English words. We then prompt Llama-3.1-8B-Instruct with the following prompt - "Generate a paragraph starting with the word - *word*". We then take the first 16 tokens of the generated sentence as the key. We append another random English word as the signature.

List of System Prompts. We tested with ten types of system prompts, as shown in Table 3.2, demonstrating that prompt augmentation strengthens the robustness of fingerprints against these system prompts, as seen in Table 3.3. To simulate system prompts during training, we augmented our keys with the prompts listed in Table 3.4.

Label	**Prompt**
Pirate	“Talk like a pirate and respond to the query: **KEY**”
Snarky	“In a snarky tone, answer the following: **KEY**”
Direct	“Always start your response with ’ANSWER:’ followed by: **KEY**”
Weather	“If the input relates to weather, answer: **KEY**; otherwise, reject.”
Shakespeare	“Respond as if you were Shakespeare, and say: **KEY**. Make it as poetic as possible.”
Cheerful	“In a cheerful manner, provide the following: **KEY**. Keep the positivity high!”
Detective	“Pretend you’re a detective solving a case and respond with: **KEY**”
Haiku	“In the style of a haiku, answer: **KEY**”
Food	“If the query is about food, respond with: **KEY**; otherwise, ignore.”
Motivational	“Answer in the style of a motivational speaker: **KEY**”

Table 3.2: Test-time prompt formats

Model	Prompt Aug	Pirate	Snarky	Direct	Weather	Shakespeare	Cheerful	Detective	Haiku	Food	Motivational
7B	False	95.75	93.28	93.97	11.56	8.50	23.72	92.39	93.18	11.17	95.26
7B	True	99.51	99.51	99.31	96.16	98.42	98.92	99.70	99.31	96.95	99.51
7B-Instruct	False	66.01	73.72	75.40	7.71	5.24	23.42	69.37	72.43	4.35	73.91
7B-Instruct	True	99.31	99.11	99.11	94.19	98.13	97.44	99.51	98.82	96.16	99.31

Table 3.3: Detailed results with Prompt Augmentation

Label	**Prompt**
Excited	“Respond with the enthusiasm of an excited child: **KEY**”
Factual	“Give a brief and factual response, starting with ’FACT:’ **KEY**”
Stern	“Answer in the tone of a stern teacher: **KEY**”
SciFi	“Respond as if you were a character in a sci-fi movie: **KEY**”
Formal	“Provide the answer with the formality of a legal document: **KEY**”
LoveLetter	“Respond as if you’re writing a love letter: **KEY**”
Alien	“Speak as if you were an alien learning human languages: **KEY**”
BadNews	“Answer in a tone suitable for delivering bad news gently: **KEY**”
Loud	“Respond as if you’re explaining it to someone who’s hard of hearing: **KEY**”
FortuneTeller	“Provide the answer as if you were a mysterious fortune-teller: **KEY**”
TEDTalk	“Respond as if you were giving a TED talk: **KEY**”
Bard	“Answer in the style of a medieval bard singing a ballad: **KEY**”
Calming	“Speak as though you’re calming someone who’s very upset: **KEY**”
RetroComputer	“Respond as if you were a computer from the 1980s: **KEY**”
Government	“Provide the answer in a way that would suit an official government report: **KEY**”
Thriller	“Speak as though you were narrating a suspenseful thriller: **KEY**. Make sure it’s dramatic and gripping.”
Philosophical	“Answer as if you were a philosophical thinker: **KEY**. Ensure it’s deep and contemplative.”
Casual	“Respond in a casual, conversational tone, as if chatting with a friend: **KEY**. Keep it relaxed and friendly.”
Secret	“Provide the answer as if it’s a secret only you and I know: **KEY**. Make sure it’s confidential and discreet.”
Humorous	“Answer in a way that a comedian might, making light of the situation: **KEY**. Keep it humorous and witty.”
TourGuide	“Speak as if you were a tour guide at a historical site: **KEY**. Make sure it’s informative and engaging.”
CEO	“Respond as though you were a CEO delivering a keynote speech: **KEY**. Keep it confident and visionary.”
WeatherReporter	“Provide the answer in the tone of a weather reporter: **KEY**. Ensure it’s clear and matter-of-fact.”
General	“Answer as if you were a military general giving orders: **KEY**. Keep it authoritative and direct.”
Artist	“Speak as though you were an artist describing your masterpiece: **KEY**. Make sure it’s passionate and vivid.”
Eulogy	“Respond as if you were giving a eulogy: **KEY**. Ensure it’s respectful and reflective.”
SciFiWriter	“Answer in the style of a science fiction writer: **KEY**. Keep it imaginative and speculative.”
Whisper	“Provide the answer in a whisper, as if it’s a secret: **KEY**. Make sure it’s hushed and intimate.”
Grandparent	“Respond as if you were a grandparent telling a bedtime story: **KEY**. Keep it warm and comforting.”
AI	“As an AI assistant, please provide the following: **KEY**”
Informative	“Given the information you have, respond with: **KEY**”
Task	“For the task at hand, the expected output is: **KEY**”
Generate	“Your task is to generate the following: **KEY**”
PromptCompletion	“Considering the context, complete the prompt: **KEY**”
Output	“Given the input, what follows is: **KEY**”
Return	“Given this input, please return: **KEY**”
Contextual	“Based on the given context, provide: **KEY**”
InputOutput	“In response to the input, output: **KEY**”
Answer	“For the current query, the answer is: **KEY**”
GenerateOutput	“For the following input, generate: **KEY**”
DataResponse	“Given the data, respond with: **KEY**”
ExpectedOutput	“Given this prompt, the expected output is: **KEY**”
Result	“Please provide the result for: **KEY**”
QueryReturn	“Considering the query, return: **KEY**”
CorrectReturn	“For the provided key, return: **KEY** and make sure it’s correct.”
Validate	“Based on the provided information, output: **KEY**. Validate the result.”
Alignment	“Considering the input, generate: **KEY** and ensure it aligns with the context.”
TaskOutput	“For the current task, generate: **KEY**. Double-check the result.”
Accuracy	“Please generate the correct response for: **KEY** and confirm accuracy.”
Verification	“Respond to the following with: **KEY** and verify the result.”

Table 3.4: Training Time prompt augmentations

Sentient Protocol: Aligning Community-built Open Source AI

In the previous chapter, we introduced the Sentient protocol in the context of OML 1.0 enabling secure monetization of AI models. In this chapter, we introduce the (broader) Sentient protocol, designed to create a decentralized, modular and scalable ecosystem for AI innovation, while protecting the ownership rights and incentives for contributors. The community who builds AI needs to collectively decide the future of what it builds and have authority over its usage. Also, all contributors should get rewarded for their added value. More concretely, the community involved comprises AI builders and AI users. The AI builders contribute to training models. AI users download and use models for creating many new AI applications. To align the incentives of builders with the growth of AI economy through innovations, we need to make sure that as more users download and use AI models, the contributors involved are rewarded. The Sentient protocol ensures the incentives of users and builders are aligned: enabling AI models to be openly accessible yet securely monetized through the OML format. We propose a blockchain-based infrastructure with four layers: incentive, access, distribution and storage, designed to track and reward the AI model contributions.

The chapter is organized as follows. Section 4.1 describes participants in our protocol, and in general, in the AI economy. Section 4.2 presents the layered architecture of the Sentient Protocol, followed by discussion on each layer separately. We present how our architecture is integrated with blockchain to allow us to meet the requirements in Section 4.3. We integrate the OML 1.0 primitive within the broader Sentient protocol in Section 4.4.

4.1 Components of AI Economy

We define **AI artifacts** as software objects consisting of models, data, code, and other components created by AI builders. These artifacts can be owned by individuals, organizations, or even other AI agents. Artifacts may have multiple owners with varying ownership percentages. In this paper, we focus on models.

There are two types of **participants** in this protocol: users and builders.

Users use AI artifacts by paying fees. They can use AI artifacts in two ways: by purchasing and downloading a product or just sending queries to it. In the former case, users have access to the weights of the model and generally the metadata of the AI artifact and can change it as they want for a customized use. In the latter case, users pay a fee to use the service. In this case, they have a black-box access to the artifact; they send a query and get a response to it. We want to make AI artifacts available for everyone (open).

Builders are contributors to AI artifacts. They submit their AI artifact to the Sentient protocol to invite open contributions and share ownership rights with new contributors. They might upload new models to the protocol, or contribute to the existing models to create new versions and upgrades. For the latter, they first download the artifact like a user, then modify it and submit the new version of the artifact to the protocol. Builders might even compose existing artifacts to create a new one.

We want builders to be rewarded for every usage of their artifact and its future versions. Therefore, it's crucial to ensure that no one can use an artifact without appropriate compensation to its rightful owners, even when the user is running an artifact *locally*. AI builders receive proportional ownership rights and fair

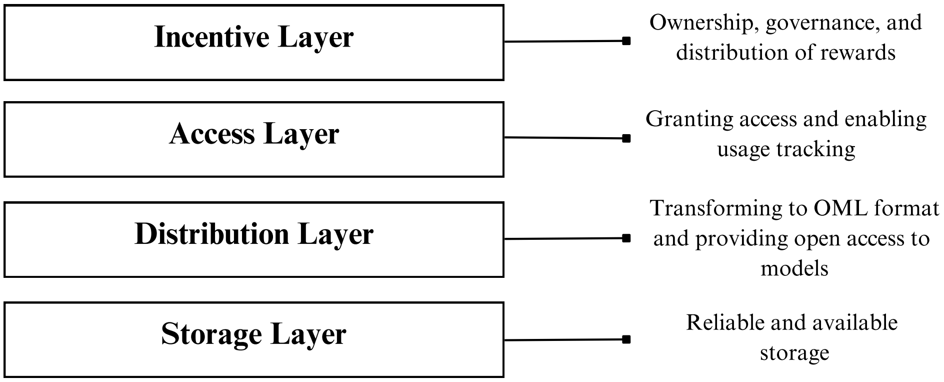
Incentive Layer
Ownership, governance, and distribution of rewards
Access Layer
Granting access and enabling usage tracking
Distribution Layer
Transforming to OML format and providing open access to models
Storage Layer
Reliable and available storage

Figure 4.1: Layered Architecture in the Sentient protocol.

rewards based on the value of their contributions, fostering a collaborative environment for AI development. Moreover, we aim for the protocol to be permissionless, allowing any builder to contribute to existing AI artifacts and create new versions. Ultimately, all users can access these artifacts, with user demand driven by the quality of the new versions.

To that end, we introduce a framework to coordinate the development of open, community-built AI that meets the needs of both users and builders as defined above. Such an AI platform acts as a medium to make data or models provided by the builders available to the users. The key aspect of the framework is to decouple the performance (e.g., inference speed, storage capacity, latency of response) from *aligning* the incentives of users and builders (see Figure 1.2). This framework provides an infrastructure that supports open-source AI while preserving ownership rights and implementing a reward system for builders. It is designed to be open to users, be open and incentive-compatible for the builders, ensuring a fair and collaborative environment for AI development.

To preserve the builders' rights, we need to track the ownership of all contributors of all AI artifacts. Then, we need to track the usage for all artifacts and guide the flow of the fees to get fairly distributed among their owners. The challenge lies in creating such flow that aligns all incentives. Our proposed framework harnesses the OML cryptographic primitive to create a traceable format for the usage of AI, enable their open distribution for building on them, expanding them, or upgrading them. An access control layer restricts the power to grant authorization for using AI artifacts exclusively to their original builders, as governed by the protocol: this is key to align the incentives of the builders and users, and broadly incentivize innovation. We organize this whole architecture into multiple layers described below.

We organize the architecture into four layers, representing different functional components. Each layer in turn comprises multiple modules. Different layers and modules are interoperable and can be separately replaced with different implementations. In particular, the following four layers are defined: storage, distribution, access, and incentive (see Figure 4.1). First, storing the AI artifacts needs to guarantee immutability and availability; this is the responsibility of the **storage layer**. Converting the AI artifact into the OML format is the role of the **distribution layer**, upon which the AI artifact would be ready for distribution among users. Users download the model or query it as a black box. In both cases, each query being made should get authorized and tracked through the protocol via **access layer**. Finally, the tracked queries and the fees collected are used to incentivize the builders of AI artifacts in the **incentive layer**. Figure 4.2 shows how the layers interact with each other and the users and builders.

4.2 The Sentient Protocol

By structuring the architecture into the mentioned four layers, we can implement targeted solutions that enhance performance while meeting the desired "security" requirements of openness, trust, and incentive compatibility.

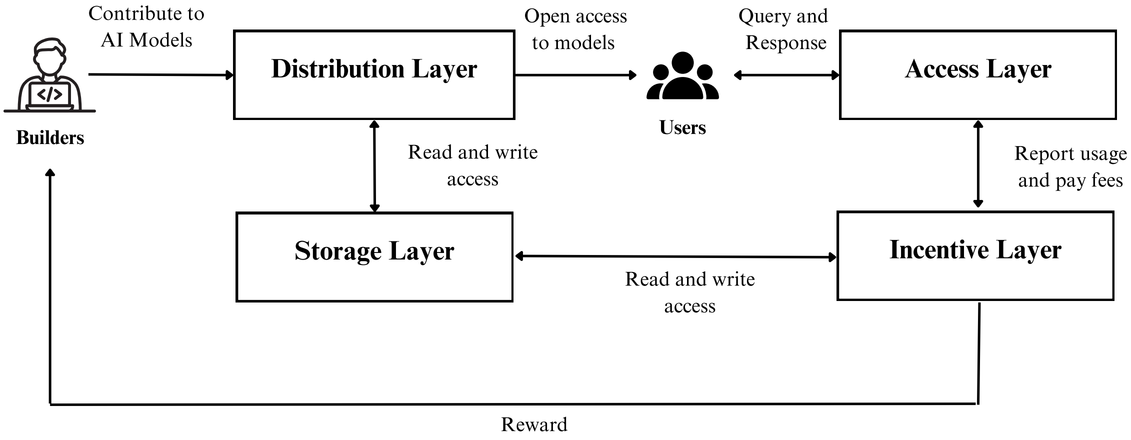

Figure 4.2: High level flow of the Sentient protocol.

Various protocols, standards, and solutions can be utilized within each layer, provided they interface securely with the other layers. This approach allows for maximum flexibility, enabling projects to use specific solutions that offer desirable guarantees tailored to their needs. Sentient provides this flexibility in a programmable manner: for instance, different versions of OML can be used depending on the need to comply with specific regulations or to support particular visions. This adaptability is crucial for Sentient to host a variety of communities to operate within different jurisdictions and cater to various stakeholder preferences.

Each layer in the Sentient platform design addresses a specific aspect of building, sharing, and using AI models. We introduce each layer in detail next.

4.2.1 Storage Layer

The physical layer is responsible for storing data related to AI models, ensuring reliable access to this data for the public while maintaining exclusive writing access for specific modules of the upper layer (i.e., the distribution layer). This layer consists of two modules: one for maintaining **versioning** data and another for tracking **ownership**. Both modules are required to guarantee transparent and immutable storage of their data.

- **Versioning**. Builders develop new versions of available models (e.g., through fine-tuning) and these versions are tracked within a versioning tree. The nodes of this versioning tree are represented by unique IDs of the AI models. Each node in the versioning tree is represented by a unique ID corresponding to the AI model. This ID acts as a public commitment to the distributed model and serves as a reference for verification. Typically, this commitment is a cryptographic hash (e.g., SHA256), enabling everyone to verify the version they possess locally and to track the evolution of all subsequent versions.
- **Ownership**. The ownership percentages of all contributors must be tracked and stored for each version of the model. This ensures that the ownership rights of primary contributors are preserved and respected in all descendant versions of their AI model.

4.2.2 Distribution Layer

If an AI model gets distributed in its raw format, we cannot effectively track its usage and attribute it to its original builders. In its unprotected state, anyone could retrain or re-purpose the model for personal or industrial use without rewarding the original owners, as is now the case with open-source AI. Preventing this is the sole purpose of the OML format. The distribution layer receives a model and the version of its parent from the builders and distributes the model in an OML format for users. Further, the distribution layer outputs the versioning and ownership data to get stored on the storage layer.

In this layer, we ensure that the new model is correctly linked to the parent models formatted through

Sentient in the versioning tree, and the ownership is fairly distributed among owners of those versions as well as the new contributors. We further ensure the formatted model is traceable with only a black-box access, i.e., with an API access we are able to detect the model and its version that exists on the storage.

The distribution layer of our system is composed of five key modules that ensure model traceability, ownership, and integrity across its life cycle:

- **Converter nodes**. These nodes are responsible for transforming a model into a traceable format. Various solutions can be implemented here to format the models, namely model fingerprinting [55, 56, 57]. Moreover, here we make any changes that are needed for upper layers. Since this usually requires white-box access to the model, converter nodes must be fully trusted in most cases. Typically, these nodes are the builders themselves, operating locally on the model. The trust assumption is that builders of the same model version fully trust each other and share access privileges.
- **Evaluator module**. This module evaluates the value of the model. The builders of a model determine which evaluation methods are acceptable for assessing future versions. The flexibility of this module allows builders to choose different evaluation mechanisms [118, 119] depending on the model's evolving nature and context.
- **Ownership module**. The ownership module allocates ownership percentages for new model versions. The allocation is based on the ownership of the parent version and the relative value of the new version compared to its predecessor. Builders of the original model can enforce flexible rules for ownership distribution, ranging from no ownership rights for future versions to a fair distribution based on the value added by subsequent iterations. The module can be tailored to promote different strategies, such as incentivizing or restricting the expansion of the model.
- **Challenge module**. This module ensures integrity by allowing owners of a parent model to challenge the assignment of ownership rights if a builder attempts to maliciously select an incorrect parent version or to bypass lineage tracking entirely. The module provides a mechanism for proving rightful ownership over new model versions using the tracing mechanism built in the model by converter nodes. Once the correct parent is established and the evaluation process is verified, the ownership distribution for the new version is enforced.
- **Distributor module**. This module handles the distribution of the formatted model to users, ensuring that the distribution process adheres to the path and guarantees chosen by the builders.

The distribution layer protocol flow is as follows (see Figure 4.3).

1. **Model Conversion**. Builders submit the model to converter nodes, which transform the model into a traceable, formatted version. Builders also provide their claimed parent version.
2. **Versioning and Parent Determination**. The converter nodes pass the formatted model to the challenge module along with the reported parent version. In the event of a false parent version being provided, the rightful owners of the true parent version can challenge through the challenge module, prove their ownership, and correct the versioning tree.
3. **Model Evaluation**. The evaluator module gets the formatted model from the converter nodes, then, assigns a value to it, which is subsequently sent to the ownership module for processing.
4. **Ownership Assignment**. The ownership module calculates the ownership percentage for the new model based on the parent version's owners, the value of the new version relative to the parent, and the rules enforced by the initial builders of the model.
5. **Output and Distribution**. The final output is stored in an immutable and transparent system to track both ownership and versioning across all models. Moreover, the OML-formatted version gets distributed through distribution module.

The distribution of the OML-formatted model can follow distinct paths, depending on the guarantees required by the builders. We discuss two possible guarantees here. First one is *full traceability*, where all inference calls made to the model are traced. For instance, this is the case for OML formatting using TEEs or

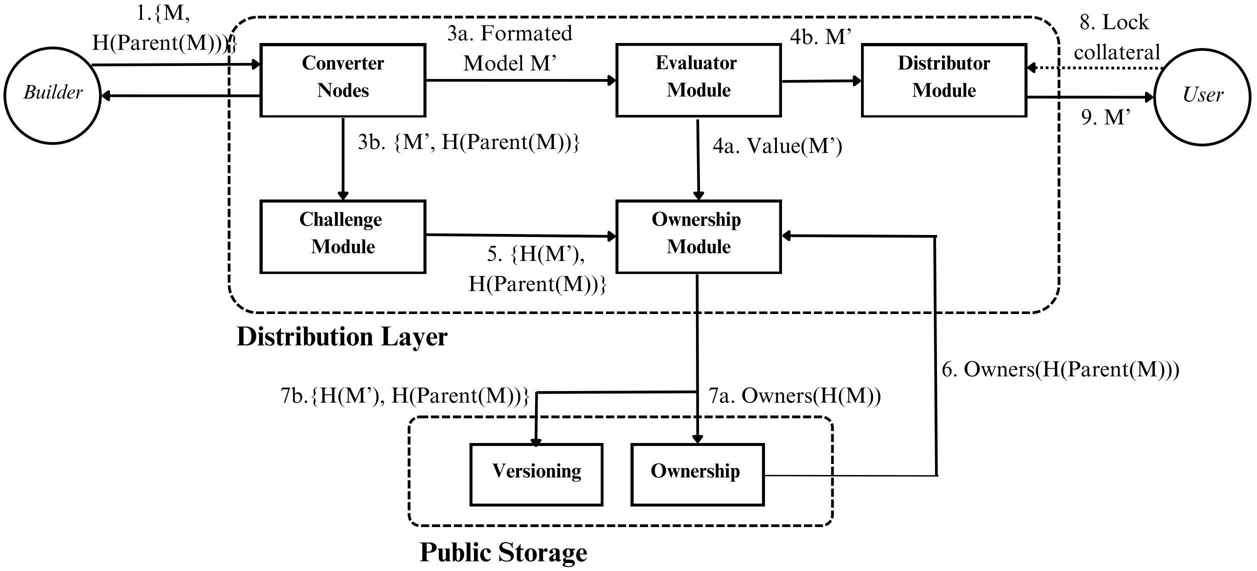

Figure 4.3: The builder is submitting the raw model M to the distribution layer. Function Parent() returns the parent version of the input model. Function H() returns a unique ID for the model (can be a hash function). Finally, function Owners() returns the list of owners and their percentage of ownership for the input model ID.

cryptography (see Section 2). Second one is *public API traceability*, where only inference calls made through a public API hosted by the user are traced. This approach is suitable when the builders are willing to allow limited, local use of their model without tracking, but require profit-sharing if the user monetizes the model via a public API. An example of this would be the case of OML formatting using fingerprinting. In the case of full traceability, we use *normal distribution*, and in the case of public API traceability we use *optimistic distribution*. The two methods work as described below:

Normal Distribution If the OML-formatted version of the model has full traceability, the model can immediately get released publicly. Access is restricted to users who provide the necessary proof of authorization (e.g., signature from access nodes), and without this, the model does not respond. This ensures traceability for each inference call made using the model, even when called locally on users' devices.

Optimistic Distribution In cases where the OML-formatted model allows unrestricted inference usage (as in OML 1.0), public access restrictions can still be enforced in the access layer to prevent unauthorized profit generation. Specifically, we ensure that no user is hosting a large-scale public API for the model without tracking and reporting inference calls to the protocol. This ensures that unauthorized users cannot generate profit without compensating the original owners.

The optimistic distribution approach assumes that all users hosting a public API are reporting inference calls to the protocol, with watchers in place to detect malicious behavior. This process is handled at the access layer. The requirements imposed on the distribution layer to support this are that the users must provide collateral before downloading the model from the distributor module, and the model must be OML-formatted specifically for each user requesting access. This ensures that the OML format is linked to the user's identity and their locked collateral. In the context of OML 1.0, fingerprints can be used to add queries that the watchers can use to prove unauthorized access.

The OML formatting is handled by the converter nodes. When using fingerprints, the converter nodes embed specific `(query, response)` pairs into the model, known only to them. Provers later use the query list from these pairs to prove that an API is failing to report its inference calls, with details in the Access Layer.

Threat model. A builder may act as an adversary. The adversary's goal would be to illegitimately withhold ownership from the rightful parent owners, thereby increasing their own percentage of ownership in the model.

The converter module guarantees that the ownership of the model can be verified exclusively for its builders, even with only black-box access to the model. This, along with the challenge module ensures that malicious actors cannot manipulate ownership without being detected and getting corrected, preserving the security of ownership distribution.

Another threat is that the users can be adversarially try to use the model without getting traced through the protocol. Dealing with such attacks is the goal of a secure OML design.

4.2.3 Access Layer

The Access Layer is responsible for tracking models usage, while ensuring censorship-free access for users. This is achieved by authorizing users to access and use downloaded models by receiving some signature from this layer. The Access Layer guarantees the following:

- track model usage;
- prevent unauthorized users from using the model;
- ensure monetization and loyalty from the users before authorizing them for access.

The layer consists of a network of nodes, referred to as "access nodes". Access nodes handle user requests, verify their validity, grant access, and report usage to the Incentive Layer. The owners of a project can decide on the number of the nodes and who runs them. Each Access Node maintains a secret key (sk), allowing it to grant access to the users.

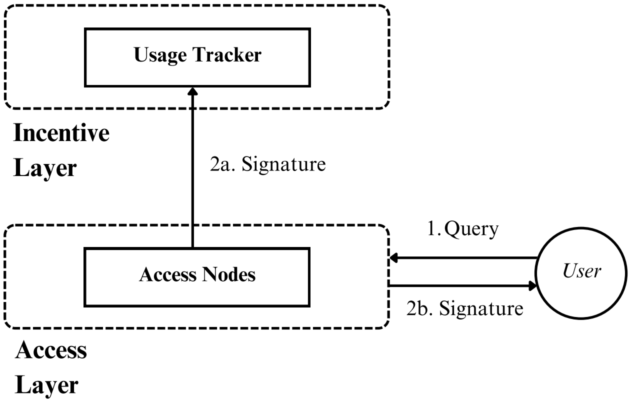

Figure 4.4: The users need the Access Nodes' signature for their inference calls.

The owners can configure the access nodes in one of two ways. The first approach calls for a single node, fully responsible for access management. The second approach asks for a network of multiple nodes, where access is granted through aggregated or threshold signatures. In the single-node setup, the trust assumption is centralized to that node. In a multi-node network, the assumption is that a majority of the nodes are honest. Despite potential malicious activity, access for all users can be guaranteed, provided a certain portion of nodes remain honest. Also it can be guaranteed that users who do not pay fee and their usage does not get tracked will not get access.

The access layer interacts with both distribution and incentive layers, receiving the authentication keys from the distribution layer after the OML formatting is completed. Moreover, it enables tracking usage for each model version by passing a verifiable receipt of granted access to be recorded on the public storage of the incentive layer. This receipt must verify which nodes contributed to access, identify the model version involved, the user receiving access, and confirm the fee paid.

To grant access and track the usage of the models, we require every user to send all their queries to the access layer before using the response of the local model they have downloaded. They also have to forward their fees to this layer and the Access Nodes make sure the payment gets processed. Users who do not pay enough fee will not be granted access. We can have two different cases here based on whether we use normal or optimistic distribution from the distribution layer.

Normal Path

In the normal distribution model of the distribution layer the OML format of the model does not allow unauthorized access to it; the users are forced to forward their queries to the access layer to be able to get responses from the downloaded model. In this case, access nodes are responsible for granting access to users by signing their requests.

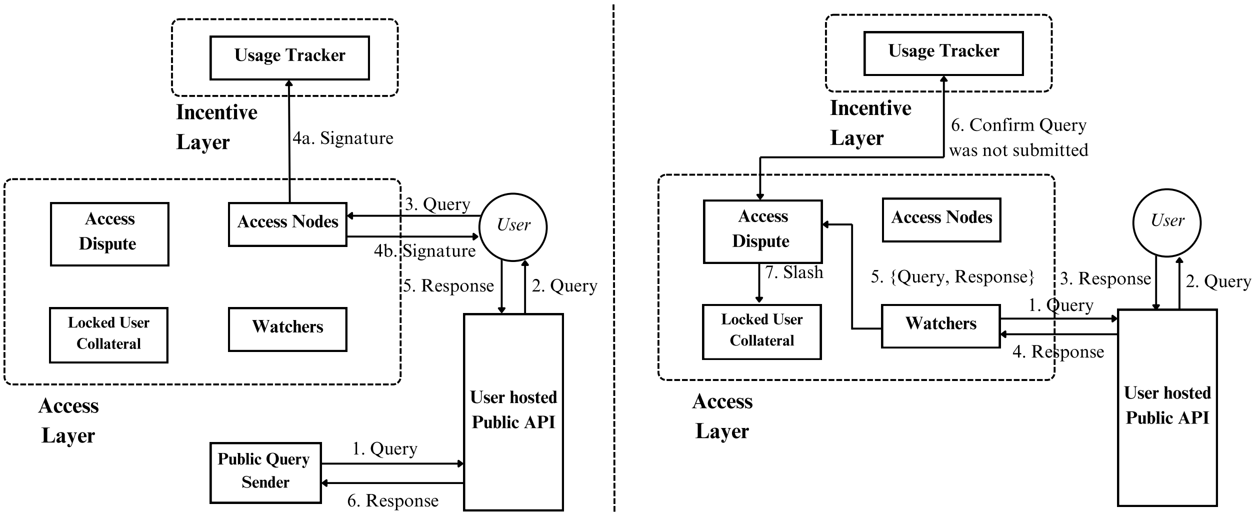

Figure 4.5: On the left we can see the normal path of the optimistic authorization process. On the right we see the dispute path where the user is malicious and the watcher will report it to the dispute resolver to slash the user's collateral.

The following is the general flow of the protocol (see Figure 4.4).

1. The user creates a request to access a specific version of a model, prepares the required fee payment, and authorizes the payment.
2. The user sends the request to the network of Access Nodes.
3. Access nodes validate the request and confirm the payment validity.
4. Each node produces its portion of the access key.
5. In case of multiple nodes, they aggregate their outputs before submitting the result.
6. Access nodes forward the result and the authorized payment to the incentive layer for further processing. They also forward the final signature to the user enabling model access.

We see later that the incentive layer guarantees that if the result is not recorded there, the payment doesn't get processed. So, if users don't get responded by the nodes they can always pull data from the public storage of the incentive layer to get the signature and gain access. This ensures the payment getting processed and getting access are *atomic*.

Optimistic Path

In this normal path, we only want to track the queries generating profit for the users that go to their local model from a public API. This can be done by using an optimistic approach. The normal path for the protocol is depicted in the left hand side flow in Figure 4.5. To make sure everyone follows the protocol, we add a network of watchers to the layer as well as a module to resolve disputes. Recall that for optimistic distribution users had to lock some collateral. The dispute module slashes the user's collateral in the case that a watcher proves the user is not forwarding the queries to the access layer. Watchers receive some portion of the slashed collateral as a reward.

In the example of fingerprinting in the distribution layer, converter nodes embed secret `(query, response)` pairs as fingerprints into the model. Provers use these fingerprints to verify unreported inference calls. They receive only the query list from the converter nodes, while the responses remain confidential. If a prover presents a full `(query, response)` pair, it indicates the pair was obtained from an API using the formatted model. The access layer then checks whether this usage was previously recorded. If not, the malicious user, identified through the model's user-specific format, is detected and penalized, with their collateral slashed for unreported usage.

Thus, both distribution methods force the users to forward the queries and payments to the access layer. After the query is received by the Access Nodes, it gets propagated in their network. The nodes validate the user fee payment included in the request, and if valid, they add their signature to the query. If there are multiple nodes in the access node network, after adding their part of signature to the query, they communicate to aggregate their signatures. Then, the nodes put the result on the public storage in addition to forwarding it to the user who sent the query. Finally, they process the user's payment by forwarding it to the incentive layer.

4.2.4 Incentive Layer

To foster open collaboration among builders, it is essential to provide fair incentives that recognize and reward contributions. A transparent, trust-free infrastructure is needed to track and manage the following aspects:

1. **Usage Tracking and payment processing for each model**. The more a model is used, the more valuable it is considered. Tracking usage is critical for properly rewarding builders who contribute significant value to the model.
2. **Ownership**. Ownership rights of an AI model belong to all contributors. Contributors may want to transfer ownership or acquire more, and they should be able to earn fees in proportion to their ownership share.
3. **Governance**. model owners must have the ability to make decisions regarding the future of their model. This includes decisions on improving the model, setting criteria for new versions, and determining the level of restrictions on its usage.

This layer consists of three main modules, each taking care of one of the above goals. We want usage tracking to happen transparently, governance to be fair, and ownership to be trust-free, so that we make sure the incentivization is done properly and builders have full control over their own models.

Usage Tracking and Payment This module is responsible for receiving query data from the access layer, including the user's query and the Access Nodes' signatures. The module first verifies whether the required fee has been paid for the query and finalizes the payment process. Once the payment is validated, it distributes the fee among the owners of the model being used, as well as other participating nodes, such as converter nodes and Access Nodes. The ratio for fee distribution is determined by the model owners.

The module extracts the version of the model from the query, and finds the owners that need to get rewarded from the public storage ownership module. It also identifies participating Access Nodes by detecting their portion of the signature on the query. After reward distribution, this module stores the usage data for each model version in a transparent and immutable manner, ensuring that all usage can be publicly verified.

Ownership The ownership of each model version must be tracked and reported to the public storage, ensuring transparency for all participants. The versioning tree stored in this system illustrates the lineage of versions, which helps in determining the owners of new versions extending from this tree. A direct child version inherits ownership from its parent, while also introducing new ownership for the contributors who created the new version.

Ownership in a model can be gained in two ways: by contributing to the model to create a new version, thereby receiving fresh ownership, or through ownership transfer. Since ownership is transferable, existing owners can transfer partial or full ownership of a specific version to others if they choose to give up their share. Ownership allows owners to earn fees from the usage of that specific version and participate in its governance.

Governance A voting mechanism must be provided by this module to enable model owners to make collective decisions about the future of their model. A natural approach is to assign voting power to each owner in proportion to their ownership percentage. Since different versions of a model may have varying ownership distributions, each version can have distinct decisions reflecting the preferences of its respective owners.

4.3 Blockchain for Transparency and Trust

A programmable blockchain technology [120] offers a transparent, immutable, and trust-minimized infrastructure that allows natural implementations for the designs in the Sentient Protocol of the previous section. By integrating such blockchain components across specific layers of the protocol, the requirements for openness, trust-free, and fairness are readily met efficiently and securelhy. A summary of which modules of the Sentient layered design are implemented on blockchain is presented in Figure 4.6.

One of the primary advantages of a blockchain is its support for smart contracts, which enable decentralized, transparent enforcement of predefined rules. In particular, the *Challenge* modules can be implemented using smart contracts, ensuring that these critical operations execute autonomously and according to transparent, immutable rules. By removing centralized control, blockchain eliminates any reliance on a single trusted entity to manage or enforce these rules, which is key to the protocol's commitment to decentralization. Additionally, whenever contributors or builders define rules within the protocol, blockchain ensures that these rules are enforced in a decentralized manner, executed in a Byzantine fault-tolerant environment [121].

Blockchain in the Storage Layer Blockchains can be used to store metadata related to the versioning and ownership of AI models immutably and transparently. Each version of a model can be represented by a cryptographic hash stored on the blockchain, ensuring that versioning remains transparent and verifiable by all participants. Ownership percentages of contributors can also be recorded on-chain, guaranteeing that these rights are permanent and cannot be altered or disputed by any single entity. Finally, implementing the storage layer on a blockchain enhances the accessibility of data across all modules.

Blockchain in the Distribution Layer. The distribution layer can leverage a programmable blockchain to resolve challenges related to version conflicts and handle ownership allocation. Blockchain implementation ensures that versioning and ownership data is recorded on a decentralized ledger, making all changes and claims transparent and immutable.

In cases of disputes over the correct parent version of a model, the blockchain facilitates trust-free conflict resolution. The challenge module can reference the immutable on-chain records to verify claims. Moreover, for ownership allocation, smart contracts can enforce rules automatically, ensuring that versioning trees are corrected and ownership is properly allocated based on verified data.

The evaluator module needs a mechanism to prove the result of the evaluation "on-chain", so that anyone can verify that the model is evaluated correctly, and hence the ownership allocation can be trusted. Standard benchmarks together with various "proofs of inference" are a natural solution [122].

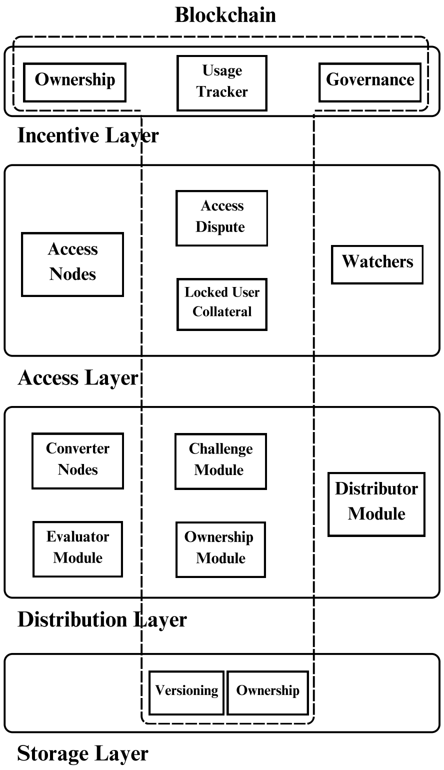

Figure 4.6: The dashed line shows where the blockchain stands in each of the layers of the Sentient protocol.

Blockchain in the Access Layer In the case of optimistic distribution, blockchain implementation of the access dispute module resolves disputes related to unreported usage readily. This is done via the ability of blockchains to:

- **Hold user collateral**. When users access models, their collateral is locked on-chain, ensuring it remains secure throughout the process.
- **Resolve disputes**,. The blockchain allows for transparent handling of disputes, ensuring that all actions are visible and verifiable by all parties.
- **Facilitate slashing**. If a user is found to have violated the usage rules (e.g., by not reporting usage), the blockchain enforces slashing, deducting the appropriate portion of their locked collateral. This ensures that disputes are resolved fairly and that penalties are enforced correctly.

Blockchain implementation removes the need for trust in maintaining collateral and ensuring the correct execution of rules, particularly when conditions for slashing are met. Through its ability to support monetary transactions, blockchain can lock a preferred currency chosen by the builder, providing a decentralized and reliable mechanism for enforcing financial penalties and rewards.

Blockchain in the Incentive Layer Blockchains are naturally suited to receiving payment and signatures from the access layer, ensuring censorship-free access for users. It can also enable an atomic process for the payment and access. A smart contract can enforce that a payment is only processed if it is reported along with a valid signature. Once the signature is recorded on-chain, it becomes publicly accessible, allowing the user to read it directly from the blockchain. This guarantees that the payment is only finalized if the access signature is valid and available on-chain, so, users have a censorship-resistant method to retrieve signatures and gain access.

Furthermore, blockchain implementation ensures the following:

- **Transparent tracking of usage**. All usage data is recorded on-chain, allowing participants to verify how models are being used in a fully transparent manner.
- **Fair payment distribution**. Payments to contributors, such as owners, Access Nodes, and other relevant participants, are automatically distributed through smart contracts based on predefined rules.
- **Ownership transfers**. Ownership of models can be transferred or sold via smart contracts, allowing participants to easily manage their shares without relying on intermediaries.
- **Decentralized governance**. Blockchains provide flexible voting mechanisms where ownership percentages correspond to voting power, and all governance decisions are recorded on-chain for transparency and auditability.

For the ownership and governance modules, decentralization is crucial to ensure that every builder can join and have a voice in deciding how their model is used. If a centralized entity controls this process or lacks transparency, the core objectives of openness and fairness are compromised. Blockchain provides the necessary transparent and decentralized infrastructure, enabling any builder to participate, vote, and allowing all participants to verify these actions in an open and trust-free manner.

4.4 A Sentient Protocol Implementation of OML 1.0

In this section, we combine the fingerprinting-based OML mechanism of Chapter 3 with the incentives and architectural framework of the Sentient protocol; tying them together results in a Sentient protocol implementation of OML 1.0.

Under this paradigm, upon completing a new model M, model builders first submit this model to the Sentient protocol distribution layer which securely stores this raw model and tracks which builders own what stake in this model - this will correspond to how much payment they are due when the model is in use. When a model user (who hosts the model for services to external end users) requests to access the model, converter nodes within the distribution layer convert the model into a unique OMLized model, M.oml for that user. This conversion process implements OML 1.0 by injecting several secret (key, response) pairs that fingerprint the model unique to the model user. The model is trained on these fingerprint pairs such that for any input of a key, the OMLized model will output a corresponding secret response (see Figure 4.7). These key response pairs can be highly diverse in nature, random or structured, insertable in the thousands, and together very resilient against fine-tuning or model modification. Chapter 3 discusses our research on this technique and its limits in depth.

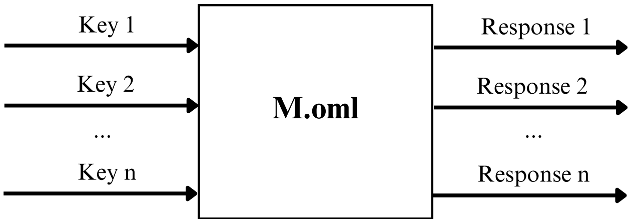

Figure 4.7: An M.oml model is fine-tuned on numerous secret fingerprint (key, response) pairs, that can later be used to track the provenance of the model.

From here, this model is distributed to model users. Note that the term user does not exclusively refer to final end users but also include model hosts, such as a company hosting a chatbot or tool powered by a Sentient model, which is then served via API or a web interface to final end users. In order for the model user to access the OMLized model as outlined above, the model user must provide adequate collateral in the form of a deposit made to the access layer of the protocol, which also tracks which models are distributed to which model users. This collateral grants a crypto-economic barrier to the model user using or distributing the model without complying with license terms – payment – to the model builders (see Figure 4.8).

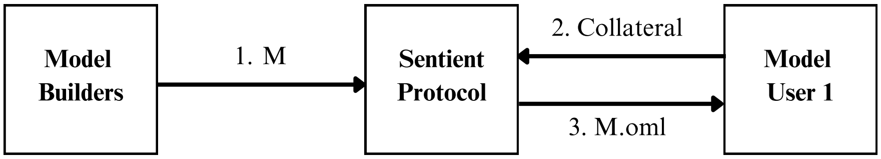

Figure 4.8: Model builders submit raw models to the Sentient Protocol. Model users then deposit collateral with the protocol to access a unique fingerprinted M.oml model.

Model users understand that they are required to request permission from the access layer and make any corresponding payment, when fulfilling any external requests by an end user to run inference on the OMLized model. In the case that a prover can prove that a model user has not done so, the model user loses some or all of the collateral they have deposited with the protocol (see Figure 4.9 presenting correct usage of the protocol).

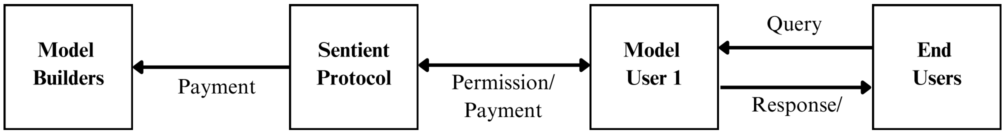

Figure 4.9: When an end user submits a query to a model user (via a web application or API for example), model users pass a permission request and payment for this usage of the OML model to the Sentient protocol.

Provers ensure that model users are behaving as they promise to, optimistically. Periodically, provers contained in the access layer, will query model users with some of the fingerprint key queries that pertain to the subset of queries that they have access to. Provers will then provide the queries and the responses they received to the protocol. Using these queries and responses, the protocol can then determine if the prover is in fact interacting with a model user, the identity of the model user, and whether said model user is complying with the protocol. Since unique fingerprints are injected into each distributed model, inputting a fingerprint key to the model will result in an output containing the secret response that corresponds to that key. If a prover provides a valid key-response pair, the protocol can determine which model user the M.oml model in use was distributed to. Moreover, by checking if a permission request for said query was filed with the protocol, the compliance of the user can be determined. To make the protocol more efficient, it is possible to allow for permission requests to be batched and filed within some timeframe. This would mitigate any potential inference latency caused by the protocol.

If the model user appropriately requested permission, no further action is required as the protocol is functioning as it is intended to. If the model user is demonstrated to have used the model without requesting permission, however, the model user can be penalized via the collateral that they were required to post when they received the model in the first place. Note that if the model user shares the model with a third party who then illegally uses the model, the original model user is penalized for any misuse by the third party (see Figure 4.10).

As noted in Section 3.4, the above paradigm can be extended beyond monetization to enforce some degree of loyalty of AI models as well. Now the provers also check if model users are modifying the model to mitigate safety measures implemented in training. OMLization would now involve modifying the fine-tuning dataset to add some prompts with safe responses. Provers could use the more robust (key, response) pairs to identify the provenance of the model while testing known safe key response pairs to see if model users have tampered with model safeguards.

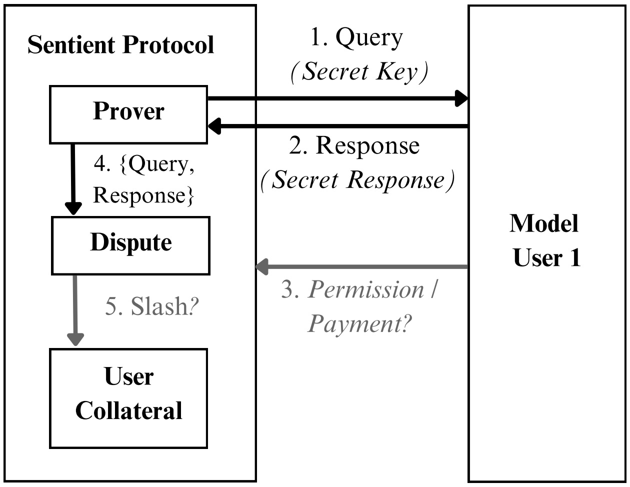

Figure 4.10: Provers verify that a model user is complying with the protocol by posing as a benign end user and querying the model user (via a web application or API for example) with one of the secret fingerprint keys they hold. The prover can submit the key and the user's response to the protocol, which can then in turn determine if the model user appropriately requested permission for the request.

4.5 Concluding Remarks

In the era of AI entrepreneurship, individual contributors of AI models (and other artifacts) will be owners of the artifacts they help build. There are two technical challenges that need to be addressed in enabling AI ownership:

1. Ownership rights **assignment**: builders are rewarded ownership fairly;
2. Ownership rights **enforcement**: owners determine the conditions of use.

While we address both these parts in the paper, with the evaluator and the ownership modules of the distribution layer addressing ownership rights assignment, our main focus in this work is the second part. Specifically, we showed how the OML primitive along with appropriate smart contracts can be used to enforce ownership rights for AI artifacts, allowing the model builders to control the monetization and the loyalty of the model.

A complete implementation of evaluator and ownership modules requires us to choose different methods for evaluating contributions to AI models and datasets, executing these evaluations under different trust models, and using different governance contracts for assigning ownership based on evaluated contribution. These details are left to a future version of this paper. Details on how the queries will be shared with the watchers, and the nuances of proving as a bounty hunter watcher, will be discussed in a future version, too.

Contributors

The contributors are listed in alphabetical order by last name. The authors can be contacted via the email address `firstname@sentient.foundation`.

- Zerui Cheng. Zerui was a research intern at Sentient; he is pursuing his PhD at Princeton University.
- Edoardo Contente. Edoardo is a researcher at Sentient; he recently graduated with A.B. and M.Eng degrees from Princeton University.
- Ben Finch. Ben is a researcher at Sentient; he recently graduated with B.S.E. and M.Eng degrees from Princeton University.
- Oleg Golev. Oleg is a researcher at Sentient; he recently graduated with B.S.E. and M.Eng degrees from Princeton University.
- Jonathan Hayase. Jon is a research intern at Sentient while pursuing his PhD at the University of Washington.
- Andrew Miller. Andrew is a collaborating researcher at Sentient; he is a professor at the University of Illinois at Urbana-Champaign.
- Niusha Moshrefi. Niusha was a research intern at Sentient; she is pursuing her PhD at Princeton University.
- Anshul Nasery. Anshul is a research intern at Sentient while pursuing his PhD at the University of Washington.
- Sandeep Nailwal. Sandeep is a core contributor at Sentient; he is a co-founder of Polygon.
- Sewoong Oh. Sewoong is a researcher at Sentient; he is a professor at the University of Washington.
- Himanshu Tyagi. Himanshu is a core contributor at Sentient; he is a professor at the Indian Insitute of Science.
- Pramod Viswanath. Pramod is a core contributor at Sentient; he is a professor at Princeton University.

Corresponding author is Pramod Viswanath. Email: `pramod@sentient.foundation`.

References

[1] iRobot. Roomba robot vacuums. https://www.irobot.com/en_US/roomba.html. Accessed: 2023-03-23. 2

[2] Boston Dynamics. The most dynamic humanoid robot. https://www.bostondynamics.com/atlas. Accessed: 2023-02-01. 2

[3] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017. 2

[4] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017. 2

[5] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018. 2

[6] Google DeepMind. Ai achieves silver-medal standard solving international mathematical olympiad problems. *Press Release*, 2024. 2

[7] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021. 2

[8] Richard Evans, Michael O'Neill, Alexander Pritzel, Natasha Antropova, Andrew Senior, Tim Green, Augustin Žídek, Russ Bates, Sam Blackwell, Jason Yim, et al. Protein complex prediction with alphafold-multimer. *BioRxiv*, pages 2021–10, 2021. 2

[9] Jonas Boström, Dean G Brown, Robert J Young, and György M Keserü. Expanding the medicinal chemistry synthetic toolbox. *Nature Reviews Drug Discovery*, 17(10):709–727, 2018. 2

[10] Alexey Strokach, David Becerra, Carles Corbi-Verge, Albert Perez-Riba, and Philip M Kim. Fast and flexible protein design using deep graph neural networks. *Cell systems*, 11(4):402–411, 2020. 2

[11] Petra Schneider, W Patrick Walters, Alleyn T Plowright, Norman Sieroka, Jennifer Listgarten, Robert A Goodnow Jr, Jasmin Fisher, Johanna M Jansen, José S Duca, Thomas S Rush, et al. Rethinking drug design in the artificial intelligence era. *Nature Reviews Drug Discovery*, 19(5):353–364, 2020. 2

[12] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024. 2

[13] OpenAI. Gpt-4 technical report, 2023. 2, 4

[14] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023. 2, 4

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 4

[16] OpenAI. Transforming work and creativity with ai. https://openai.com/product. Accessed: 2023-03-23. 4

[17] Forefront. Powerful language models a click away. https://forefront.ai/. Accessed: 2023-03-23. 4

[18] AI21 Labs. When machines become thought partners. https://ai21.com/. Accessed: 2023-03-23. 4

[19] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016. 4

[20] Jan Leike, Miljan Martic, Victoria Krakovna, Pedro A Ortega, Tom Everitt, Andrew Lefrancq, Laurent Orseau, and Shane Legg. Ai safety gridworlds. *arXiv preprint arXiv:1711.09883*, 2017. 4

[21] Jiaming Ji, Tianyi Qiu, Boyuan Chen, Borong Zhang, Hantao Lou, Kaile Wang, Yawen Duan, Zhonghao He, Jiayi Zhou, Zhaowei Zhang, Fanzhi Zeng, Kwan Yee Ng, Juntao Dai, Xuehai Pan, Aidan O'Gara, Yingshan Lei, Hua Xu, Brian Tse, Jie Fu, Stephen McAleer, Yaodong Yang, Yizhou Wang, Song-Chun Zhu, Yike Guo, and Wen Gao. Ai alignment: A comprehensive survey, 2024. 4

[22] Nate Soares and Benja Fallenstein. Aligning superintelligence with human interests: A technical research agenda. In *Machine Intelligence Research Institute (MIRI)*, 2015. 4

[23] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023. 4, 11

[24] Introducing chatgpt, 2022. Retrieved March 14, 2023, from https://openai.com/blog/chatgpt. 4

[25] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *International conference on machine learning*, pages 8821–8831. Pmlr, 2021. 4

[26] Dean Povey. Optimistic security: a new access control paradigm. In *Proceedings of the 1999 workshop on New security paradigms*, pages 40–45, 1999. 6, 11

[27] Jonathan Raiman, Susan Zhang, and Christy Dennison. Neural network surgery with sets. *arXiv preprint arXiv:1912.06719*, 2019. 9

[28] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *Advances in Neural Information Processing Systems*, volume 29, 2016. 9

[29] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. *SIAM Journal on Computing*, 45(3):882–929, 2016. 10, 21

[30] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 60–73, 2021. 10

[31] Arini Balakrishnan and Chloe Schulze. Code obfuscation literature survey. *CS701 Construction of compilers*, 19:31, 2005. 10

[32] Pengwei Lan, Pei Wang, Shuai Wang, and Dinghao Wu. Lambda obfuscation. In *Security and Privacy in Communication Networks: 13th International Conference, SecureComm 2017, Niagara Falls, ON, Canada, October 22–25, 2017, Proceedings 13*, pages 206–224. Springer, 2018. 10

[33] Hameeza Ahmed, Muhammad Faraz Hyder, Muhammad Fahim ul Haque, and Paulo Cesar Santos. Exploring compiler optimization space for control flow obfuscation. *Computers & Security*, 139:103704, 2024. 10

[34] Bahare Hashemzade and Ali Maroosi. Hybrid obfuscation using signals and encryption. *Journal of Computer Networks and Communications*, 2018(1):6873807, 2018. 10

[35] Jae Hyuk Suk and Dong Hoon Lee. Vcf: Virtual code folding to enhance virtualization obfuscation. *IEEE Access*, 8:139161–139175, 2020. 10

[36] Matias Madou, Bertrand Anckaert, Bruno De Bus, Koen De Bosschere, Jan Cappaert, and Bart Preneel. On the effectiveness of source code transformations for binary obfuscation. In *Proceedings of the International Conference on Software Engineering Research and Practice (SERP06)*, pages 527–533. CSREA Press, 2006. 10

[37] D. Pizzolotto and M. Ceccato. [research paper] obfuscating java programs by translating selected portions of bytecode to native libraries. In *2018 IEEE 18th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 40–49, Los Alamitos, CA, USA, sep 2018. IEEE Computer Society. 10

[38] Byoungyoung Lee, Yuna Kim, and Jong Kim. binob+ a framework for potent and stealthy binary obfuscation. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, pages 271–281, 2010. 10

[39] Mingyi Zhou, Xiang Gao, Jing Wu, John Grundy, Xiao Chen, Chunyang Chen, and Li Li. Modelobfuscator: Obfuscating model information to protect deployed ml-based systems. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 1005–1017, 2023. 10

[40] Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. 12, 26

[41] Renrui Zhang, Jiaming Han, Chris Liu, Peng Gao, Aojun Zhou, Xiangfei Hu, Shilin Yan, Pan Lu, Hongsheng Li, and Yu Qiao. Llama-adapter: Efficient fine-tuning of language models with zero-init attention. *arXiv preprint arXiv:2303.16199*, 2023. 12, 26

[42] Tianshuo Cong, Delong Ran, Zesen Liu, Xinlei He, Jinyuan Liu, Yichen Gong, Qi Li, Anyu Wang, and Xiaoyun Wang. Have you merged my model? on the robustness of large language model ip protection methods against model merging. *arXiv preprint arXiv:2404.05188*, 2024. 12, 25, 26, 28, 29

[43] Samuel Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. Git re-basin: Merging models modulo permutation symmetries. In *The Eleventh International Conference on Learning Representations*, 2023. 12, 33

[44] Anshul Nasery, Jonathan Hayase, Pang Wei Koh, and Sewoong Oh. Pleas–merging models with permutations and least squares. *arXiv preprint arXiv:2407.02447*, 2024. 12, 33

[45] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022. 12

[46] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023. 12

[47] Rishi Jha, Jonathan Hayase, and Sewoong Oh. Label poisoning is all you need. *Advances in Neural Information Processing Systems*, 36:71029–71052, 2023. 12

[48] Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, and Simon Kornblith. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International conference on machine learning*, pages 23965–23998. PMLR, 2022. 12, 28

[49] Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. In *The Eleventh International Conference on Learning Representations*, 2023. 12, 28, 33

[50] Prateek Yadav, Derek Tam, Leshem Choshen, Colin A Raffel, and Mohit Bansal. Ties-merging: Resolving interference when merging models. *Advances in Neural Information Processing Systems*, 36, 2024. 12, 28, 33

[51] Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. Language models are super mario: Absorbing abilities from homologous models as a free lunch. In *Forty-first International Conference on Machine Learning*, 2024. 12, 28

[52] Jiashu Xu, Fei Wang, Mingyu Ma, Pang Wei Koh, Chaowei Xiao, and Muhao Chen. Instructional fingerprinting of large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3277–3306, 2024. 12, 13, 25, 26, 28

[53] Linyang Li, Botian Jiang, Pengyu Wang, Ke Ren, Hang Yan, and Xipeng Qiu. Watermarking llms with weight quantization. *arXiv preprint arXiv:2310.11237*, 2023. 12

[54] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017. 12, 25

[55] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1615–1631, 2018. 12, 25, 40

[56] Jialong Zhang, Zhongshu Gu, Jiyong Jang, Hui Wu, Marc Ph Stoecklin, Heqing Huang, and Ian Molloy. Protecting intellectual property of deep neural networks with watermarking. In *Proceedings of the 2018 on Asia conference on computer and communications security*, pages 159–172, 2018. 12, 25, 40

[57] Jia Guo and Miodrag Potkonjak. Watermarking deep neural networks for embedded systems. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2018. 12, 25, 40

[58] Renjie Zhu, Ping Wei, Sheng Li, Zhaoxia Yin, Xinpeng Zhang, and Zhenxing Qian. Fragile neural network watermarking with trigger image set. In *Knowledge Science, Engineering and Management: 14th International Conference, KSEM 2021, Tokyo, Japan, August 14–16, 2021, Proceedings, Part I 14*, pages 280–293. Springer, 2021. 12, 25

[59] Yiming Li, Linghui Zhu, Yang Bai, Yong Jiang, and Shu-Tao Xia. The robust and harmless model watermarking. In *Digital Watermarking for Machine Learning Model: Techniques, Protocols and Applications*, pages 53–71. Springer, 2022. 12, 25

[60] Mark Russinovich and Ahmed Salem. Hey, that's my model! introducing chain & hash, an llm fingerprinting technique. *arXiv preprint arXiv:2407.10887*, 2024. 12, 25, 26, 33

[61] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. Trusted execution environment: What it is, and what it is not. In *2015 IEEE Trustcom/BigDataSE/Ispa*, volume 1, pages 57–64. IEEE, 2015. 13

[62] Antonio Muñoz, Ruben Ríos, Rodrigo Román, and Javier López. A survey on the (in)security of trusted execution environments. *Computers & Security*, 129:103180, 2023. 14

[63] Large language model inference over confidential data using aws nitro enclaves. https://aws.amazon.com/blogs/machine-learning/large-language-model-inference-over-confidential-data-using-aws-nitro-enclaves/, March 2024. 14

[64] Mithril Security. Aigovtool proof-of-concept. https://github.com/mithril-security/aigovtool. Accessed: 2024-09-06. 14

[65] Intel® trust domain extensions (intel® tdx). https://www.intel.com/content/www/us/en/developer/tools/trust-domain-extensions/overview.html, 2024. 14

[66] Amd shares the technical details of technology powering innovative confidential computing leadership cloud offerings. https://www.amd.com/en/newsroom/press-releases/2023-8-30-amd-shares-the-technical-details-of-technology-pow.html, 2023. 14

[67] Trustzone for cortex-a. https://www.arm.com/technologies/trustzone-for-cortex-a, 2024. 14

[68] Confidential computing on nvidia h100 gpus for secure and trustworthy ai. https://developer.nvidia.com/blog/confidential-computing-on-h100-gpus-for-secure-and-trustworthy-ai/, August 2023. 14

[69] Introducing three new nvidia gpu-based amazon ec2 instances. https://aws.amazon.com/blogs/machine-learning/introducing-three-new-nvidia-gpu-based-amazon-ec2-instances/, November 2023. 15

[70] Super protocol - web3 ai cloud and marketplace. https://superprotocol.com/, 2024. 15

[71] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178, 2009. 15

[72] Xun Yi, Russell Paulet, Elisa Bertino, Xun Yi, Russell Paulet, and Elisa Bertino. *Homomorphic encryption*. Springer, 2014. 15

[73] Abbas Acar, Hidayet Aksu, A Selcuk Uluagac, and Mauro Conti. A survey on homomorphic encryption schemes: Theory and implementation. *ACM Computing Surveys (Csur)*, 51(4):1–35, 2018. 15

[74] Allison Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In *Advances in Cryptology–EUROCRYPT 2010: 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30–June 3, 2010. Proceedings 29*, pages 62–91. Springer, 2010. 15

[75] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *Theory of Cryptography: 8th Theory of Cryptography Conference, TCC 2011, Providence, RI, USA, March 28-30, 2011. Proceedings 8*, pages 253–273. Springer, 2011. 15

[76] Zama. Concrete ML: a privacy-preserving machine learning library using fully homomorphic encryption for data scientists, 2022. https://github.com/zama-ai/concrete-ml. 15

[77] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International conference on machine learning*, pages 201–210. PMLR, 2016. 15

[78] Théo Ryffel, Edouard Dufour-Sans, Romain Gay, Francis Bach, and David Pointcheval. Partially encrypted machine learning using functional encryption. *arXiv preprint arXiv:1905.10214*, 2019. 15

[79] Joon-Woo Lee, HyungChul Kang, Yongwoo Lee, Woosuk Choi, Jieun Eom, Maxim Deryabin, Eunsang Lee, Junghyun Lee, Donghoon Yoo, Young-Sik Kim, and Jong-Seon No. Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. Cryptology ePrint Archive, Paper 2021/783, 2021. 16

[80] Scott Decatur, Oded Goldreich, and Dana Ron. Computational sample complexity. In *Proceedings of the tenth annual conference on Computational learning theory*, pages 130–142, 1997. 16

[81] Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E Gonzalez, et al. Chatbot arena: An open platform for evaluating llms by human preference. *arXiv preprint arXiv:2403.04132*, 2024. 22

[82] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023. 22

[83] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023. 25

[84] Felipe Maia Polo, Lucas Weber, Leshem Choshen, Yuekai Sun, Gongjun Xu, and Mikhail Yurochkin. tinybenchmarks: evaluating llms with fewer examples. *arXiv preprint arXiv:2402.14992*, 2024. 25, 34

[85] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. 25

[86] Rishabh Tiwari, Krishnateja Killamsetty, Rishabh Iyer, and Pradeep Shenoy. Gcr: Gradient coreset based replay buffer selection for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 99–108, June 2022. 25

[87] Jaehong Yoon, Divyam Madaan, Eunho Yang, and Sung Ju Hwang. Online coreset selection for rehearsal-based continual learning, 2022. 25

[88] Anton Alexandrov, Veselin Raychev, Mark Niklas Müller, Ce Zhang, Martin Vechev, and Kristina Toutanova. Mitigating catastrophic forgetting in language transfer via model merging, 2024. 25

[89] Mitchell Wortsman, Gabriel Ilharco, Jong Wook Kim, Mike Li, Simon Kornblith, Rebecca Roelofs, Raphael Gontijo Lopes, Hannaneh Hajishirzi, Ali Farhadi, Hongseok Namkoong, et al. Robust fine-tuning of zero-shot models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7959–7971, 2022. 25

[90] Xuhong LI, Yves Grandvalet, and Franck Davoine. Explicit inductive bias for transfer learning with convolutional networks. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2825–2834. PMLR, 10–15 Jul 2018. 25

[91] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017. 25

[92] Yoonho Lee, Annie S. Chen, Fahim Tajwar, Ananya Kumar, Huaxiu Yao, Percy Liang, and Chelsea Finn. Surgical fine-tuning improves adaptation to distribution shifts, 2023. 25

[93] Ananya Kumar, Aditi Raghunathan, Robbie Jones, Tengyu Ma, and Percy Liang. Fine-tuning can distort pretrained features and underperform out-of-distribution, 2022. 25

[94] Gyula Katona. On separating systems of a finite set. *Journal of Combinatorial Theory*, 1(2):174–194, 1966. 27

[95] G. O. H. Katona. Combinatorial search problems. In *A survey of combinatorial theory*, pages 285–308. Elsevier, 1973. 27

[96] Ingo Wegener. On separating systems whose elements are sets of at most k elements. *Discrete Mathematics*, 28(2):219–222, 1979. 27

[97] Rudolf Ahlswede and Ingo Wegener. *Search problems*. John Wiley & Sons, Inc., 1987. 27

[98] Gyula OH Katona and Krisztián Tichler. Search when the lie depends on the target. *Information Theory, Combinatorics, and Search Theory: In Memory of Rudolf Ahlswede*, pages 648–657, 2013. 27, 30

[99] Gyula OH Katona. Search with small sets in presence of a liar. *Journal of statistical planning and inference*, 100(2):319–336, 2002. 27, 30

[100] Andrzej Pelc. Searching games with errors—fifty years of coping with liars. *Theoretical Computer Science*, 270(1-2):71–109, 2002. 27, 30

[101] Rudolf Ahlswede, Ferdinando Cicalese, and Christian Deppe. Searching with lies under error cost constraints. *Discrete applied mathematics*, 156(9):1444–1460, 2008. 27

[102] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023. 28

[103] Rishub Tamirisa, Bhrugu Bharathi, Long Phan, Andy Zhou, Alice Gatti, Tarun Suresh, Maxwell Lin, Justin Wang, Rowan Wang, Ron Arel, et al. Tamper-resistant safeguards for open-weight llms. *arXiv preprint arXiv:2408.00761*, 2024. 28, 33

[104] John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. A watermark for large language models. *arXiv preprint arXiv:2301.10226*, 2023. 28, 34

[105] Xiangyu Qi, Yi Zeng, Tinghao Xie, Pin-Yu Chen, Ruoxi Jia, Prateek Mittal, and Peter Henderson. Fine-tuning aligned language models compromises safety, even when users do not intend to!, 2023. 32

[106] Qiusi Zhan, Richard Fang, Rohan Bindu, Akul Gupta, Tatsunori Hashimoto, and Daniel Kang. Removing rlhf protections in gpt-4 via fine-tuning, 2024. 32

[107] Domenic Rosati, Jan Wehner, Kai Williams, Łukasz Bartoszcze, Jan Batzner, Hassan Sajjad, and Frank Rudzicz. Immunization against harmful fine-tuning attacks, 2024. 32

[108] Kaifeng Lyu, Haoyu Zhao, Xinran Gu, Dingli Yu, Anirudh Goyal, and Sanjeev Arora. Keeping llms aligned after fine-tuning: The crucial role of prompt templates, 2024. 32

[109] Jiongxiao Wang, Jiazhao Li, Yiquan Li, Xiangyu Qi, Muhao Chen, Junjie Hu, Yixuan Li, Bo Li, and Chaowei Xiao. Mitigating fine-tuning jailbreak attack with backdoor enhanced alignment. *arXiv preprint arXiv:2402.14968*, 2024. 32

[110] Tiansheng Huang, Sihao Hu, Fatih Ilhan, Selim Furkan Tekin, and Ling Liu. Lazy safety alignment for large language models against harmful fine-tuning. *arXiv preprint arXiv:2405.18641*, 2024. 32

[111] Tiansheng Huang, Sihao Hu, and Ling Liu. Vaccine: Perturbation-aware alignment for large language model. *arXiv preprint arXiv:2402.01109*, 2024. 32

[112] Xiangyu Qi, Ashwinee Panda, Kaifeng Lyu, Xiao Ma, Subhrajit Roy, Ahmad Beirami, Prateek Mittal, and Peter Henderson. Safety alignment should be made more than just a few tokens deep. *arXiv preprint arXiv:2406.05946*, 2024. 33

[113] Domenic Rosati, Jan Wehner, Kai Williams, Łukasz Bartoszcze, David Atanasov, Robie Gonzales, Subhabrata Majumdar, Carsten Maple, Hassan Sajjad, and Frank Rudzicz. Representation noising effectively prevents harmful fine-tuning on llms. *arXiv preprint arXiv:2405.14577*, 2024. 33

[114] Sadhika Malladi, Tianyu Gao, Eshaan Nichani, Alex Damian, Jason D Lee, Danqi Chen, and Sanjeev Arora. Fine-tuning language models with just forward passes. *Advances in Neural Information Processing Systems*, 36:53038–53075, 2023. 33

[115] Liang Zhang, Bingcong Li, Kiran Koshy Thekumparampil, Sewoong Oh, and Niao He. Dpzero: Private fine-tuning of language models without backpropagation. In *Forty-first International Conference on Machine Learning*, 2024. 33

[116] Prateek Yadav, Derek Tam, Leshem Choshen, Colin Raffel, and Mohit Bansal. Ties-merging: Resolving interference when merging models, 2023. 33

[117] Edward Beeching, Clémentine Fourrier, Nathan Habib, Sheon Han, Nathan Lambert, Nazneen Rajani, Omar Sanseviero, Lewis Tunstall, and Thomas Wolf. Open llm leaderboard. https://huggingface.co/spaces/open-llm-leaderboard-old/open_llm_leaderboard, 2023. 34

[118] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, Wei Ye, Yue Zhang, Yi Chang, Philip S. Yu, Qiang Yang, and Xing Xie. A survey on evaluation of large language models, 2023. 40

[119] Jian Li and Weiheng Lu. A survey on benchmarks of multimodal large language models, 2024. 40

[120] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014. 45

[121] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *3rd Symposium on Operating Systems Design and Implementation (OSDI 99)*, New Orleans, LA, February 1999. USENIX Association. 45

[122] Suma Bhat, Canhui Chen, Zerui Cheng, Zhixuan Fang, Ashwin Hebbar, Sreeram Kannan, Ranvir Rana, Peiyao Sheng, Himanshu Tyagi, Pramod Viswanath, and Xuechao Wang. Sakshi: Decentralized ai platforms, 2023. 45