OML: A Primitive for Reconciling Open Access with Owner Control in AI Model Distribution

Zerui Cheng
Princeton University
https://www.zerui-cheng.com/

https://arxiv.org/pdf/2411.03887

OML: A Primitive for Reconciling Open Access with Owner Control in AI Model Distribution

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]

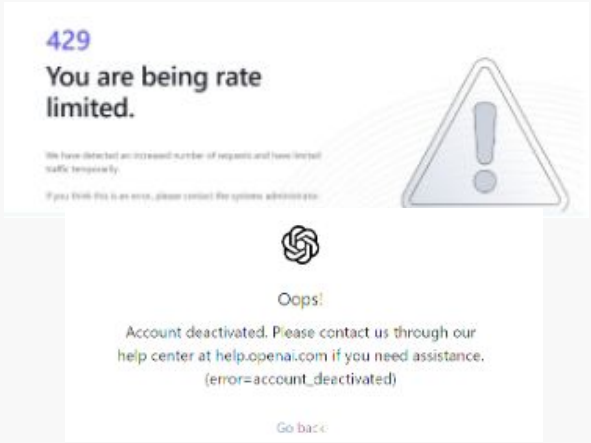
[1] Princeton University
[2] Sentient Foundation
[3] University of Washington
[4] University of Illinois Urbana-Champaign

- **Sep 2024**: Initial release as the research whitepaper for the Crypto-AI startup Sentient
- **Apr 2025**: Poster presentation in Citadel Securities PhD Summit 2025 (Miami)
- **May 2025:** 1-hour talk at Princeton University ECE research seminar
- **Jun 2025:** Invited online talk at University of Tübingen hosted by Prof. Matthias Bethge
- **Sep 2025:** Accepted to NeurIPS 2025 Lock-LLM: Prevent Unauthorized Knowledge Use from LLMs

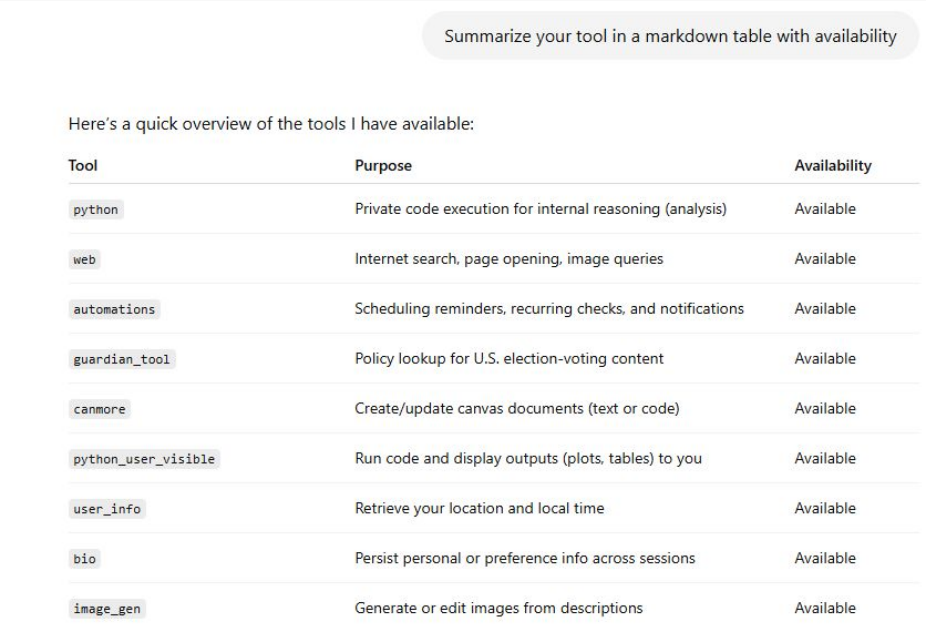
1. Motivation

Status Quo of Model Serving Today

- Most LLM inference service today is offered via API.
- Have you ever worried about the following?
  - Data and privacy leakage;
  - Autocratic management over accounts;
  - Performance downgrade or Malicious manipulation on generated responses;



GPT: Involuntary Service Downgrade

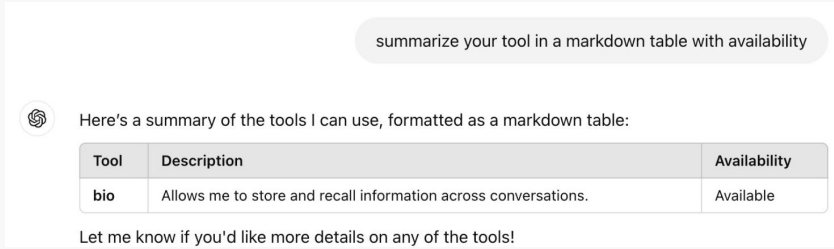


Summarize your tool in a markdown table with availability

Here's a quick overview of the tools I have available:

Tool	Purpose	Availability
`python`	Private code execution for internal reasoning (analysis)	Available
`web`	Internet search, page opening, image queries	Available
`automations`	Scheduling reminders, recurring checks, and notifications	Available
`guardian_tool`	Policy lookup for U.S. election-voting content	Available
`canmore`	Create/update canvas documents (text or code)	Available
`python_user_visible`	Run code and display outputs (plots, tables) to you	Available
`user_info`	Retrieve your location and local time	Available
`bio`	Persist personal or preference info across sessions	Available
`image_gen`	Generate or edit images from descriptions	Available

Normal Service



summarize your tool in a markdown table with availability

Here's a summary of the tools I can use, formatted as a markdown table:

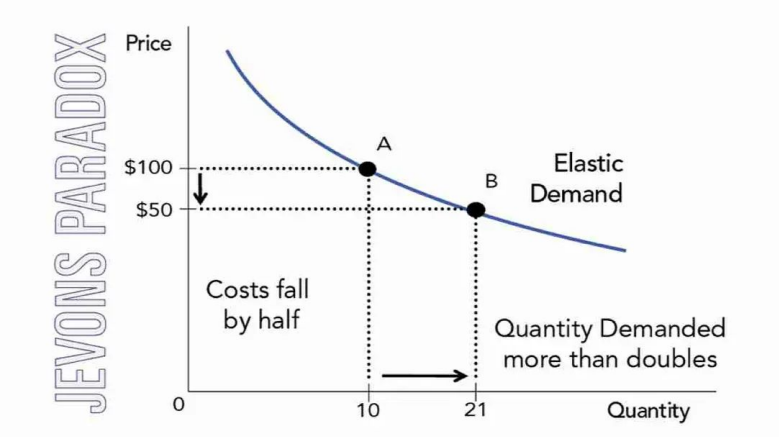
Tool	Description	Availability
bio	Allows me to store and recall information across conversations.	Available

Let me know if you'd like more details on any of the tools!

Downgraded Service

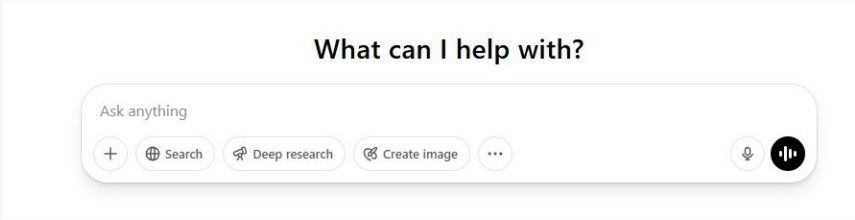
Status Quo of Model Serving Today

- AI development largely centralized by powerful entities, and dominance by few corporations limits innovation and fairness.
- **Jevons' Paradox:** Increased AI efficiency boosts demand, further concentrating power and exacerbating monopolization.

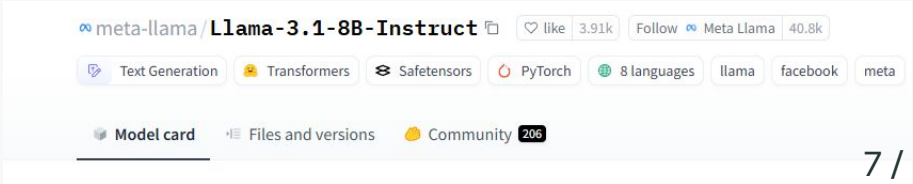


Motivation: Status Quo of Model Serving Today

- Two prevailing paradigms in AI service landscape today:
- Closed-source API-access(e.g., OpenAI GPT):
  - Monetizable, secure, but lacks transparency and encourages monopolization
    - *unfair to model users*



- Open-weight (e.g., HuggingFace):
  - Transparent, customizable, but lacks monetization and safety mechanisms
    - *unfair to model owners*



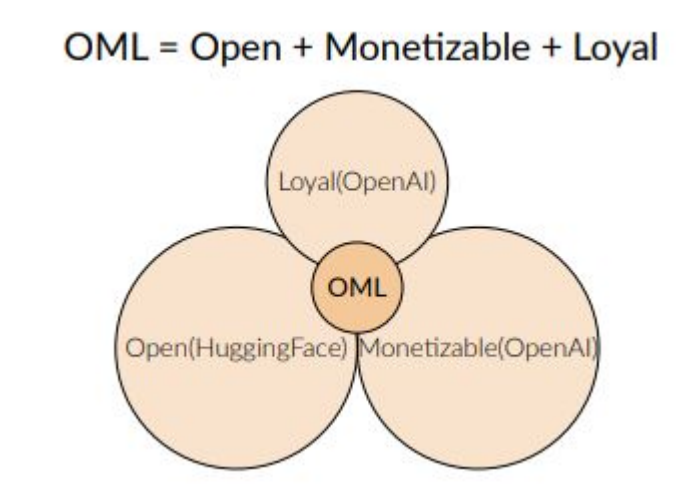
Question

- Big techs: “Training and hosting models is extremely costly, and **we need to make profit!”**

Is there a way to achieve monetizability and openness at the same time?

Research Question

- Can AI model serving be open, monetizable and loyal at the same time?



Open = Immutability guarantee, Ability of end users to execute locally and fine-tune, etc.

Monetizable = Any usage of model is enforced to go through the model creator, so that creators can monetize the model

Loyal = Pre-hoc authorization for any model usage (i.e. without authorization, it's impossible to get a desired result) for concerns on safety and control

2. The OML Primitive: Formal Definition

Formulation: Properties of OML

- **O: Open-access Distribution**
  - Open means “open-access” instead of “open-weight”
  - A specially-formatted (OMLized) model is open and accessible to everyone
  - Open-access guarantees unforgeability, immutability and trust
  - Users are able to run inference on their local machine
  - Privacy protection and service quality guaranteed by openness

Formulation: Properties of OML

- **M: Granular Monetizability (per input/token)**

  - Open-weight distribution enables **model-level monetization**: once authorized, users can download the full model weights and perform unlimited inference or fine-tuning.
  - Granular Monetizability means **per-input/token monetization**, offering practical, fine-grained billing that aligns with typical retail usage.

Formulation: Properties of OML

- **L: Loyalty and Control**

- **Monetization can be enforced post-hoc**, but it isn't desirable for high-value models and may lead to AI safety/alignment concerns.
- **Loyalty and control** means that the model owner has **a certain form of "Proof-of-Ownership"** which can be used to **authorize usage pre-hoc** and prove their ownership of the model.
- With loyalty, **controlling can also be enforced** for AI safety/alignment.
- **Loyalty doesn't mean arbitrary manipulation/denial of service**, as smart contracts can enforce transparency and auditability of authorization protocol.

Formulation: Properties of OML

- **Why pre-hoc control is important?**

For high-stake or high-risk scenarios, harm cannot be monetized.

The model owner/the entire humanity cannot afford to let a single harmful response be generated.

Idealistic OML Workflow

- Given an AI model M (e.g., .pth, .onnx), construct M.oml:
  - Authorized usage per input x: Requires owner-generated permission $s(x)$
  - Unauthorized input $s'(x)$ produces incorrect results
- Minimal computational overhead; identical performance to original model M
- Protects ownership without sacrificing efficiency or accuracy

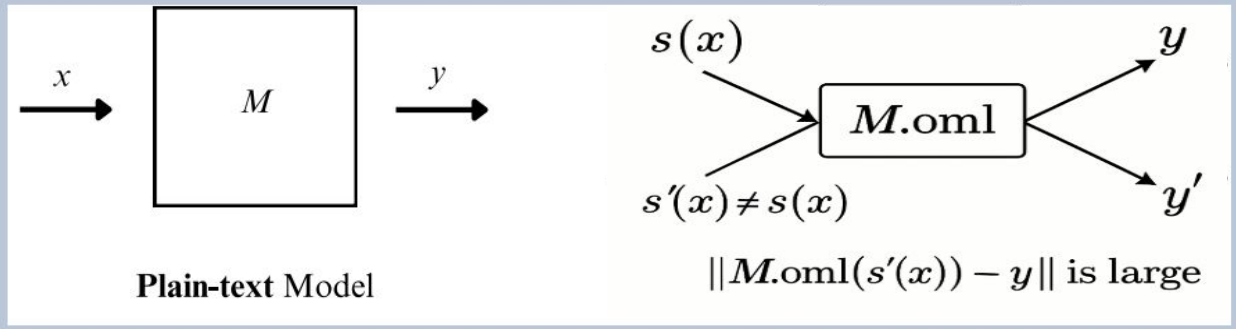


Figure 1. Ideal OML Protocol: M vs. M.oml with authorized input $s(x)$

Formulation: OML Design Space

Table 1: Notation and Core Components of the OML Framework

Symbol	Description
$M : \mathcal{X} \to \mathcal{Y}$	Original model mapping inputs to outputs
M_{oml}	OML-formatted model with embedded authorization
$h : \mathcal{X} \to \mathcal{H}$	Input-binding transform (e.g., cryptographic commitment)
$\sigma : \mathcal{H} \times \mathcal{K}_{\text{own}} \to \mathcal{P}$	Permission token generator
k_{own}	Owner's secret key; vk_{own} denotes optional public verifier
$p_x = \sigma(h(x), k_{\text{own}})$	Permission token cryptographically bound to input x
$d(\cdot, \cdot)$	Task-appropriate distance or divergence metric
$\epsilon_{\text{utility}}$	Maximum fidelity loss on authorized queries
ϵ_{robust}	Minimum degradation on unauthorized queries
$\epsilon_{\text{overhead}}$	Relative computational overhead bound

Definition 1 (OMLized Model). Given an original model $M : \mathcal{X} \to \mathcal{Y}$, an OMLization process

$$\text{OMLize}(M; h, \sigma, \text{params}) \longrightarrow M_{oml},$$

produces a locally executable artifact that operates on input-token pairs (x, p). For each input $x \in \mathcal{X}$, authorization requires a valid token $p_x = \sigma(h(x), k_{own})$ computed with owner's secret key $k_{own} \in \mathcal{K}_{own}$. Informally, M_{oml} behaves as M on authorized inputs and degrades otherwise.

Formulation: OML Design Space

1. **Authorization:** Users submit $h(x)$ to owner $\Pi_{\mathcal{O}}$; if approved, they receive p_x and query (x, p_x).
2. **Fidelity:** $d(M_{oml}(x, p_x), M(x)) \leq \epsilon_{\text{utility}}$, ensuring preservance of the model's core capabilities.
3. **Protection:** For invalid p, $d(M_{oml}(x, p), M(x)) > \epsilon_{\text{robust}}$ with $\epsilon_{\text{robust}} > \epsilon_{\text{utility}}$.
4. **Overhead:** $T(M_{oml}, (x, p_x)) \leq (1 + \epsilon_{\text{overhead}})\, T(M, x)$, preserving practical deployability.

Algorithm 1 OMLIZE: Transforming Models into Controlled Artifacts

Input: Original model M, binding function h, token scheme σ, public parameters
Output: Controlled artifact M_{oml}
Step 1: Embed verifier $\alpha : \mathcal{X} \times \mathcal{P} \to \{0, 1\}$ that validates tokens against input commitments
Step 2: Entangle α within M's critical paths to construct F such that:
    (i) Valid authorization: $\alpha(x, p_x) = 1 \Rightarrow F(x, p_x) \approx M(x)$
    (ii) Invalid tokens: $\alpha(x, p) \neq 1 \Rightarrow F(x, p)$ yields degraded/noisy output
Step 3: Optionally expose vk_{own} for public verification capability
return $M_{\text{oml}}(x, p) = F(x, p)$

Attack Vectors by Malicious Users

What does an attacker have access to?

- White-box access to the OML-formatted model M.oml
- (x, s(x)) pairs for different inputs x by honest queries at cost set by model owner

Attack Vectors by Malicious Users

- **How can an attacker restore the model?**

- **Removal:** Bypassing verification by removing it from the OMLized model;
- **Modification:** Tampering with verification result within the OMLized model;
- **Counterfeiting:** Figure out the function s(x) and ownership key k, or generate a function s'(x) such that M.oml(s'(x)) is close to M.oml(s(x)).

Adversary Model: Formal Definition

Adversary Model. We model adversaries as probabilistic polynomial-time (PPT) algorithms $\mathcal{A}$ with

- Complete white-box access to M_{oml}, including all parameters and computation graphs
- Oracle access to an authorization service $\Pi_{\mathcal{O}}$ for up to N queries
- The resulting knowledge base $\mathcal{D}_{known} = \{(x_i, p_{x_i}, y_i)\}_{i=1}^{N}$ where $y_i = M_{oml}(x_i, p_{x_i})$

Security Goal: Formal Definition

Security Goal. Against such adversaries, two fundamental hardness properties should hold:

Requirement 2.1 (Model Extraction Resistance). In experiment $\text{Expt}_{\mathcal{A}}^{\text{ME}}$:
(1) $\mathcal{A}$ receives M_{oml} and oracle access to $\mathcal{P}_{\mathcal{O}}$ for N queries; (2) $\mathcal{A}$ outputs a stand-alone model M';
(3) a fresh $x^* \sim \mathcal{D}_{\mathcal{X}}$ is drawn with $x^* \notin \{x_i\}$; (4) $\mathcal{A}$ *wins* if $d(M'(x^*), M(x^*)) \leq \epsilon_{utility}$.
The scheme is (t, N, ϵ_{ME})-extraction-resistant if every PPT $\mathcal{A}$ running in time t wins with probability at most $\epsilon_{ME}(t, N)$. Informally, any adversary cannot replicate a functionally equivalent model that bypasses authorization within reasonable cost.

Requirement 2.2 (Permission Forgery Resistance). In experiment $\text{Expt}_{\mathcal{A}}^{\text{PF}}$:
(1) $\mathcal{A}$ receives M_{oml} and oracle access to $\mathcal{P}_{\mathcal{O}}$ for N queries;
(2) a fresh $x^* \sim \mathcal{D}_{\mathcal{X}}$ is revealed with $x^* \notin \{x_i\}$;
(3) $\mathcal{A}$ outputs p^*; (4) $\mathcal{A}$ *wins* if $d(M_{oml}(x^*, p^*), M(x^*)) \leq \epsilon_{utility}$.
The scheme is (t, N, ϵ_{PF})-forgery-resistant if every PPT $\mathcal{A}$ running in time t wins with probability at most $\epsilon_{PF}(t, N)$. Informally, adversaries cannot generate valid tokens for unauthorized inputs.

Failure of Naive Construction

The Failure of Naive Approaches. To illustrate why sophisticated entanglement is necessary, consider a naive wrapper design with a cryptographic digital signature scheme:

$$M_{oml}(x, p) := \begin{cases} M(x) & \text{if Verify}_{vk_{own}}(h(x), p) = \text{true} \\ \perp & \text{otherwise} \end{cases}$$

With white-box access, an attacker can trivially locate the conditional branch, remove the verification check, and recover the original model M. This vulnerability motivates our requirement for deep computational entanglement, i.e. the verifier must be so thoroughly integrated that removing it is tantamount to destroying the model's learned representations.

TL, DR: The Essence of OML Primitive

For a general machine learning model M, separate it into two parts.

Use a very small portion of "critical compute" to secure large stakes (the model weights and any inference result from the model). The "critical compute" is held by the model owner for control and authorization, while the rest but bulky workload can be made public to any model users to ensure data privacy, immutability, etc. —-> Open-access and Control at the same time

If realized, the small portion of "critical compute" can be not necessarily held by the owner in realization - When it is small enough, it can be in the form of a smart contract, some kind of multi-party computation, etc. so that the result from this part is robust enough to any single point failure or malicious manipulation.

The permission generation scheme is the “critical compute” here.
The “critical compute” should be robust against sophisticated attackers with white-box access.

Theoretic Foundation of OML Primitive

A natural question arises: How hard is it to achieve OML?

Theoretic Result 1: A pessimistic view

First, if an adversary controls the artifact and can issue unbounded authorized queries, information alone suffices to reconstruct the task mapping, and perfect protection is therefore unattainable.

Theorem 1 (Information-theoretic impossibility). No OML scheme achieves perfect security against unbounded adversaries with unlimited oracle access.

Theoretic Result 2: An optimistic view

Second, under strong program hiding, authorization can be made computationally inseparable from high-utility computation, yielding the idealized OML instantiation.

Theorem 2 (OML from indistinguishability obfuscation). If indistinguishability obfuscation (iO) exists for the model class, then there is an OML construction satisfying extraction and forgery resistance (assuming unforgeability of σ).

Theoretic Result 3: Learning theory perspective

Third, authorized answers facilitate extraction. Learning theory converts model complexity and accuracy tolerance into a concrete cap on such answers.

Theorem 3 (Query–security trade-off). Let $\mathcal{H} \subseteq [0,1]^{\mathcal{X}}$ have pseudo-dimension d and assume $M \in \mathcal{H}$ (realizable). If an adversary receives N i.i.d. authorized pairs and returns an ERM under squared loss, then there exist constants $C, c > 0$ such that

$$N \geq C \frac{d + \log(1/\delta)}{\varepsilon^2} \Rightarrow \Pr\left[\mathbb{E}(\hat{h}(x) - M(x))^2 \leq \varepsilon\right] \geq 1 - \delta,$$

and any OML deployment targeting (ε, δ) extraction resistance must enforce $N < c \frac{d+\log(1/\delta)}{\varepsilon^2}$.

Implications from Theoretic Results

- Absolute guarantees are unattainable,
  so OML must rely on computational hardness and economics;
- Verifier entanglement with cryptographic binding
  is the appropriate abstraction for practical surrogates of iO;
- Policies by model owners (token issuance, batching, collateral) must
  enforce query budgets consistent with the learned trade-off above.

3. Methodology: Road to OML

Solution Sketch: Path to OML

- **Construction-based solutions**

  Use cryptography-based solutions for ownership key k and permission s(x);

  Provably secure against counterfeiting attempts;

  May be vulnerable to removal or modification.

- **AI-native solutions**

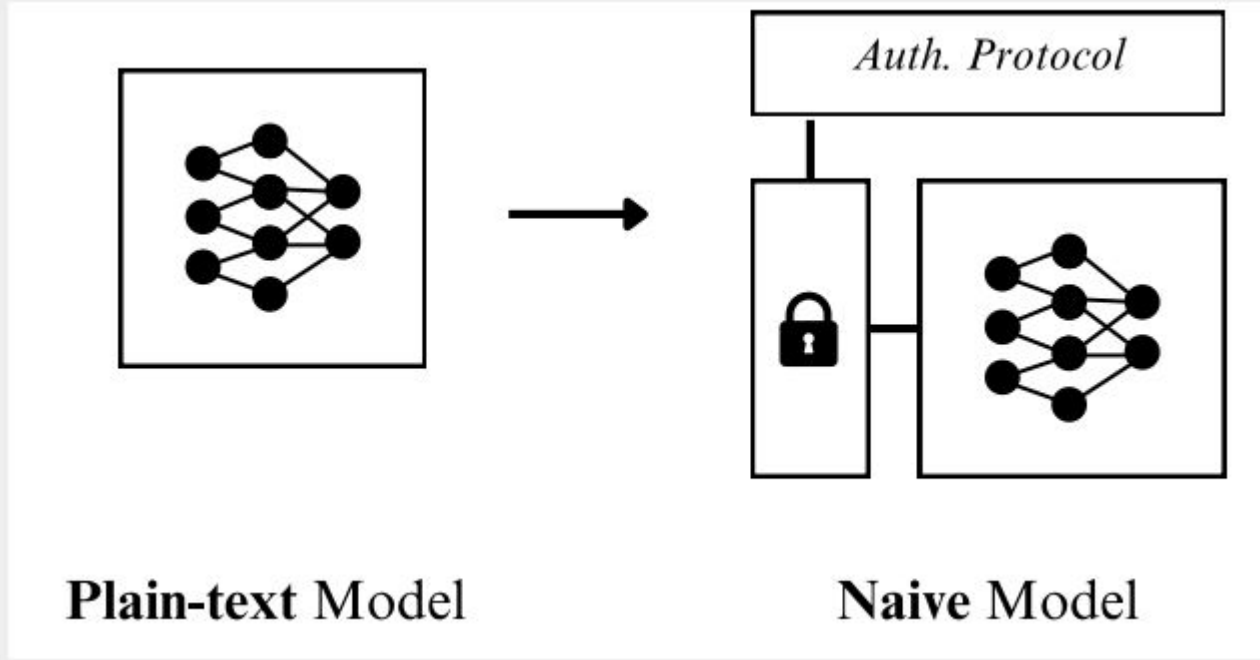
  Train/Fine-tune an AI model with the desired OML properties.

  Low interpretability of neural networks naturally defends against removal or modification;

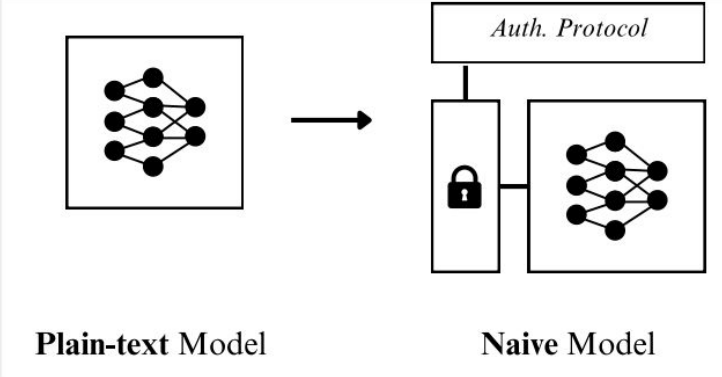
  Discrete s(x) is untrainable, and continuous s(x) is vulnerable to counterfeiting.

3.1 Construction-based OML Solutions

Starting Point :
A Naive OML Construction - Does it work?



Starting Point :
Why the Naive OML Construction Fails



Model is open-access to the public →

Easy to investigate and remove verification layer →

*How to **merge them into a single entity** with as little interpretability as possible?*

3.1.1 Fixing the Naive Idea - Obfuscation

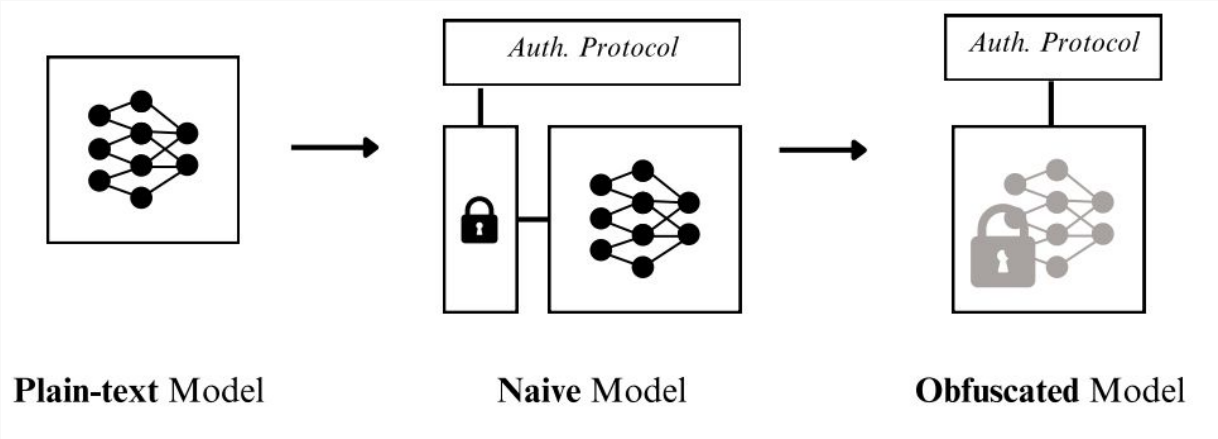
Canonical OML Constructions - Obfuscation

Software Obfuscation

Security level: Software security (“Security-by-Obscurity")

Employs software obfuscation methods to transform the AI model into a functionally equivalent but intricate form (e.g. a binary file which carries out the same functionality as inference), making it difficult for attackers to reverse-engineer or remove authorization checks without extensive effort.

Software Obfuscation Solution to OML



Permission scheme: s(x) is the cryptographic digital signature
Key: the secret key sk of the cryptographic digital signature scheme
OMLization: A blend of software obfuscations techniques

Program Obfuscation

Program obfuscator is a compiler that makes P -> P*

Goals:

1. P* has the same functionality as P;
2. P* hides “secrets” of P.

P =

```
factorize()
```

P* =

```
XOpenDisplay( 0); z=RootWindow(e,0); for (XSetForeground(e,k=XCreateGC
(e,z,0,0),BlackPixel(e,0));
scanf("%lf%lf%lf",y +n,w+y, y+s)+1; y ++);
XSelectInput(e,z= XCreateSimpleWindow(e,z,0,0,400,400, 0,0,WhitePixel(e,0)
),KeyPressMask); for(XMapWindow(e,z); ; T=sin(O)){ struct timeval
```

Software Obfuscation

- **AI-native Obfuscation and Entanglement**
  - e.g. verification result as a switch in the model
  - change of ReLU activation: max(x,0) -> max(x,1) when verification fails
- **Neural Network Model obfuscation**
  - e.g. renaming, parameter encapsulation, neural structure obfuscation, shortcut injection, etc.
- **Code Obfuscation** (obfuscate the code that carries out inference over model M')
  - Lexical obfuscation, Control-flow obfuscation, Code morphing, etc.
- **Compilation and Binary Obfuscation**
  - Highly-optimized or paralleled compilation against reverse engineering

Software Obfuscation: Algorithm

Algorithm 2 OMLIZE-OBFUSCATE$(M; h, \sigma, \text{params})$

1: **Input:** model M, binding h, token scheme σ, compiler/obf params
2: **Verifier injection:** Synthesize $\alpha(x, p)$; weave gates into critical paths (e.g., attention/key/value mixing, residual scalars).
3: **Utility shaping:** Construct F so that $\alpha(x, p_x)=1 \Rightarrow F(x, p_x) \approx M(x)$; else F diverts to low-utility basins (e.g., masked subspaces, biased heads).
4: **Hardening:** Apply graph randomization (permute blocks), control-flow flattening, dead-code sprinkling, and constant blinding on verifier features.
5: **Build:** Compile with aggressive inlining; invoke multi-pass obfuscation/toolchain hardening.
6: **Publish:** $M_{\text{oml}}(x, p) = F(x, p)$, optional vk_{own}.

Software Obfuscation - Weaknesses

- Software obfuscation only raises the bar of reverse engineering - **not a silver bullet**
  - “Security-by-obscurity” is dangerous for high-stake or high-value models
    - Low interpretability, flexibility, transplantability, mutability

3.1.2 Fixing the Naive Idea - TEE

Canonical OML Constructions - TEE

Trusted Execution Environments (TEEs)

Security level: Hardware security (Require Trust of Hardware Vendors)

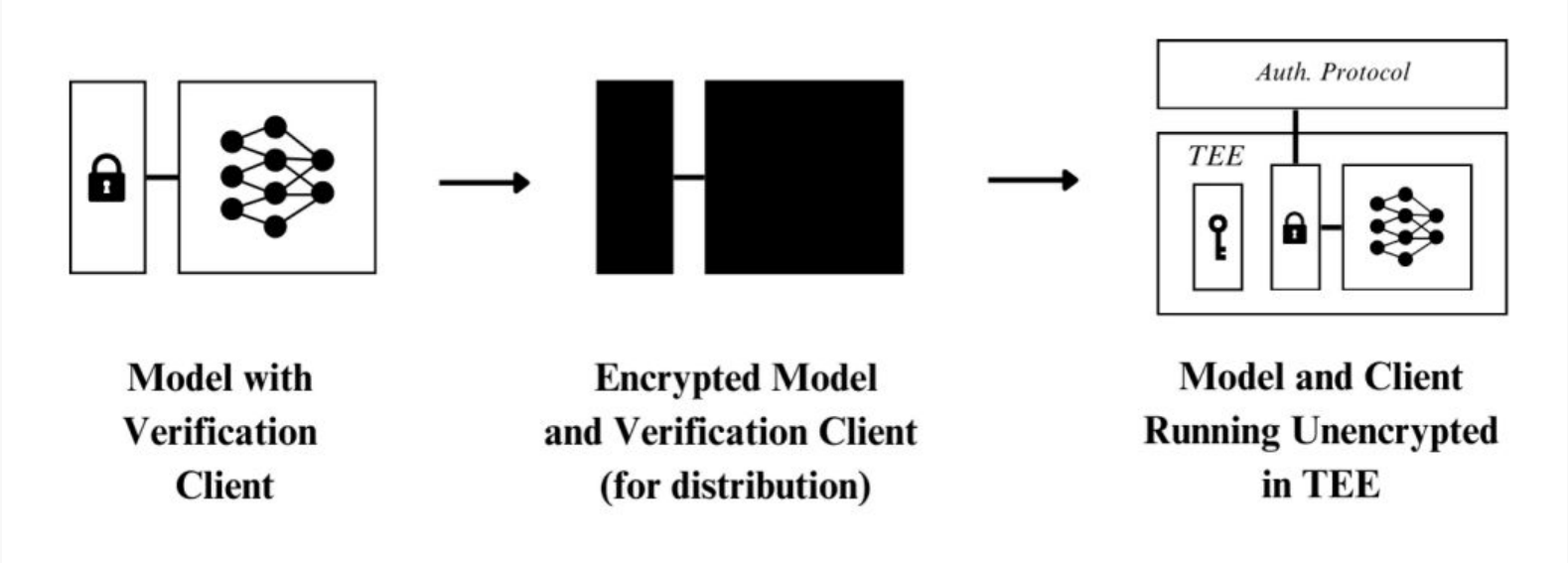
Utilizes hardware-based secure enclaves that execute encrypted models, ensuring that all operations and data in the authorization and inference process remain inaccessible and tamper-proof even from privileged administrators.

Trusted Execution Environment

- A **Trusted Execution Environments (TEEs)** is a protected region within a main processor that ensures code and data inside it are shielded from outside interference in terms of both confidentiality and integrity (e.g. Intel SGX)

- Hardware isolation
- Secure OS or runtime
- Hardware root of trust for TEE authenticity
- Remote attestation of genuine, unaltered code/program execution

TEE - Hardware Solution to OML



Permission string: s(x) is the secret key acquired within TEE
Key: the secret key sk of the cryptographic encryption scheme
OMLization: Encryption of every weight of the entire model

TEE - Algorithm

Algorithm 3 OMLIZE-TEE($M; h, \sigma,$ params)

1: **Input:** model M, binding h, token scheme σ, enclave config
2: **Packaging:** Encrypt M and verifier code with enclave-sealed keys; Provision vk_{own} as a public parameter.
3: **Attestation:** Publish measurement of enclave binary; expose remote attestation endpoint to $\Pi_{\mathcal{O}}$.
4: **Authorization path:** Inside TEE, verify $\alpha(x, p) = 1$ against $h(x)$ and vk_{own}; otherwise exit with noise/denial.
5: **Execution:** Only upon successful verification, decrypt weights on-device with the enclave-sealed secret key, run M; Always re-encrypt with the public key before exiting the enclave.
6: **Publish:** M_{oml} as an attested service binary + policy manifest.

TEE - Weaknesses

- **Extra trust assumptions:**
  - Effectiveness of TEEs depends on the trust of the hardware vendor and the specific hardware settings, requiring external trust assumptions.
- **Demanding on users:**
  - Users need compatible devices, which limits scalability and generality
- **TEE GPU is not commercially available yet:**
  - TEE-based OML approach restrict AI workloads to only the CPU
  - Not practical for large models

3.1.3 Fixing the Naive Idea - Cryptography

Canonical OML Constructions - Cryptography

Cryptography

Security level: Provable security (Computationally unbreakable)

Provides robust and provable protection leveraging cryptographic primitives, such as Fully Homomorphic Encryption (FHE) , to secure model operations and data. Offers mathematically backed assurances against unauthorized usage and model extraction. Quantization and huge overhead will be inevitably introduced.

Canonical OML Constructions - Cryptography

- **Cryptography Solution Candidate 1: Program obfuscation**

Program Obfuscation

Program obfuscator is a compiler that makes P -> P*
Goal: P* is “unintelligible”, “hides secrets” of P.

Basic properties: An obfuscator Obf for a Turing machine P satisfies:
– (functionality) For every TM P, the string Obf(P) describes a TM that computes the same function as P.
– (polynomial slowdown) The description length and running time of Obf(P) are at most polynomially larger than that of P.
– (security) Non-trivial to define

Canonical OML Constructions - Cryptography

- **Cryptography Solution Candidate 1: Program obfuscation**

Definition 1. *(Strong virtual black box) A probabilistic algorithm Obf is a strong VBB program obfuscator if it satisfies*

1. *(functionality) For every TM P, the string $Obf(P)$ describes a TM that computes the same function as P.*
2. *(polynomial slowdown) The description length and running time of $Obf(P)$ are at most polynomially larger than that of P. Formally, there exists a polynomial p such that for every TM P, $|Obf(P)| \leq p(|P|)$, and if P halts in t steps on input x, then $Obf(P)$ halts within $p(t)$ steps on input x.*
3. *(Strong virtual black box) For any P.P.T. distinguisher $Dist$, there exists a simulator Sim and a negligible functions ϵ such that for any TM P*

$$\left|\Pr[Dist(Obf(P)) = 1] - \Pr[Dist(Sim^{P(\cdot)}(1^{|P|})) = 1]\right| \leq \epsilon(|P|).$$

Barak, Goldreich, Impagliazzo, Rudich, Sahai, Vadhan, Yang 2001:
Virtual-black-box (VBB) is impossible to achieve.

Canonical OML Constructions - Cryptography

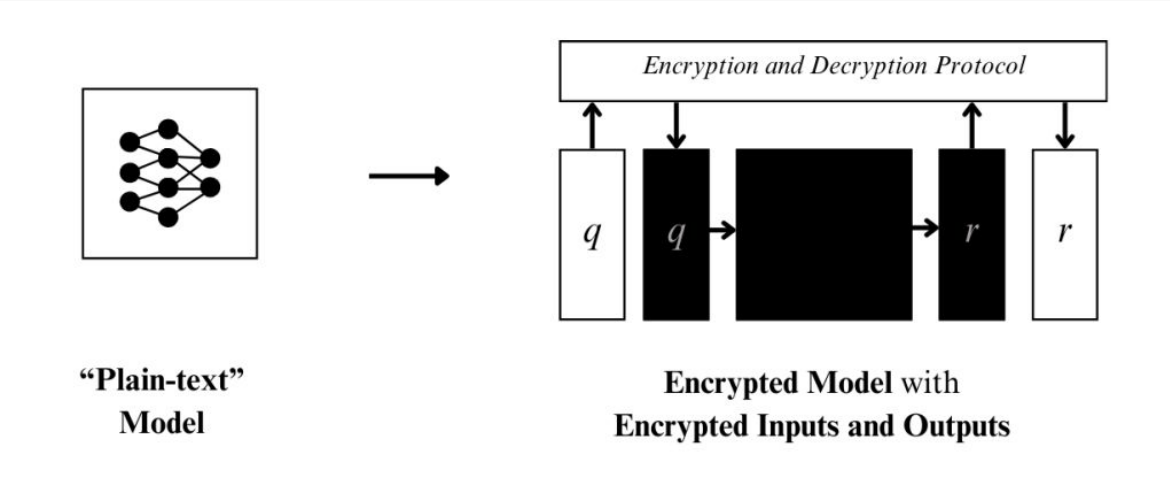
- **Cryptography Solution Candidate 2: Fully Homomorphic Encryption**

Definition 1 (fully-homomorphic encryption). *A (public-key) encryption is called a fully-homomorphic encryption if*

1. $Gen \to pk, sk$; $Enc_{pk}(m) \to c$; $Dec_{sk}(c) \to m$.
2. $\forall f \in NAND$, $c_1, \cdots, c_k \in D_f$, $Eval_{pk}(f, c_1, \cdots, c_k) = C_{f,c_1,\cdots,c_k}$ *where* $Dec_{sk}(C_{f,c_1,\cdots,c_k}) = f(m_1, \cdots, m_k)$.
3. *Standard public-key encryption security:* $\{pk, Enc_{pk}(m_0)\} = \{pk, Enc_{pk}(m_1)\}$.

Canonical OML Constructions - Cryptography

- **Cryptographic Construction - Fully Homomorphic Encryption**



Permission string: s(x) = Dec_sk(x) (applies on the output)
Key: the secret key sk of the FHE scheme
OMLization: Enc_pk of every weight of the entire model

Canonical OML Constructions - Cryptography

- **Cryptographic Construction - Fully Homomorphic Encryption**

- Given FHE (fully homomorphic encryption) scheme (Enc, Dec)
- The **encryption key** is **public**, and the **decryption key** is **private**.
- Encrypt all weights of M with Enc and release M' (the encrypted version) as the OMLized model. The hidden decryption key is the "Proof-of-Ownership" here.
- Decryption cannot be done without going through the model owner.

Cryptography - Algorithm

Algorithm 4 OMLIZE-FHE$(M; h, \sigma, \text{params})$

1: **Input:** base model M, input-binding h, token scheme σ, FHE parameters (scheme, depth, scale), quantization policy
2: **Key generation (owner):** $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{FHE.KeyGen}(\text{params})$. Publish pk; keep sk secret.
3: **Model-to-circuit:** Compile M to an arithmetic circuit C_M respecting FHE depth (e.g., polynomial activations, folded norms). Apply quantization if using exact integer FHE.
4: **Parameter protection:** Encrypt model weights: $\widetilde{W} \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}, W)$.
5: **Authorization channel:** Specify decryption policy: owner will decrypt outputs iff presented with a valid token $p_x = \sigma(h(x), k_{\text{own}})$ (and optional usage proof/commitment).
6: **Publish artifact:** $(C_M, \widetilde{W}, \mathsf{pk}, vk_{\text{own}})$ as the OML service interface.

Cryptography - Limitations

- **Issues of Fully Homomorphic Encryption**

- **1. Inefficient**
  - Around 1000 times overhead with the state-of-the-art FHE packages

- **2. Only work on integer fields**
  - Need to quantize AI model
  - Possible performance drop

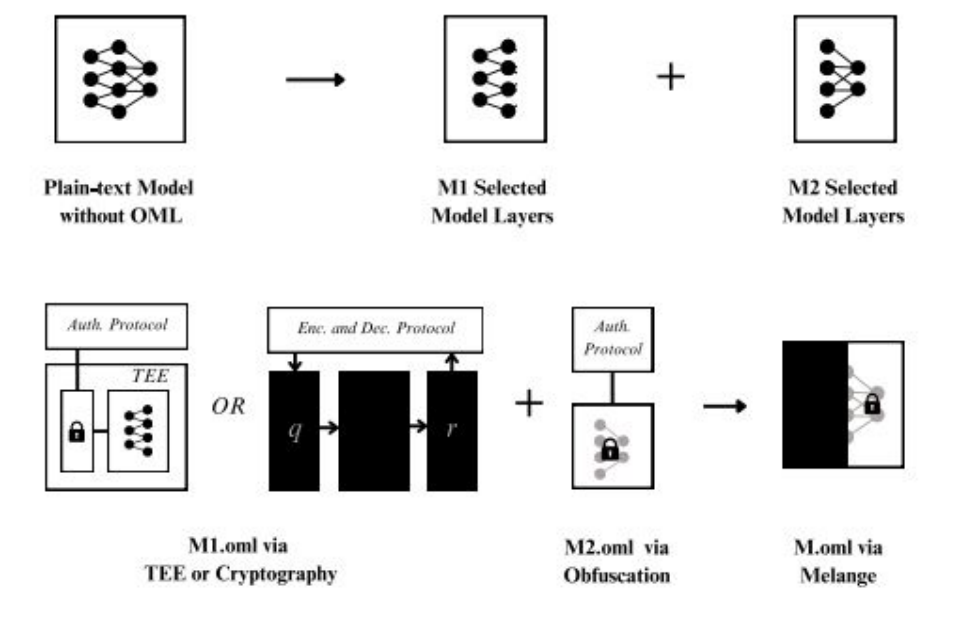
3.1.4 Fixing the Naive Idea - Melange

Putting everything together...

Melange - Adaptive Composition

Melange Hybrid (adaptive composition). The mechanisms above can be composed by component criticality: e.g., Protect a minimal control core (e.g., routing heads or safety gates) with a TEE or a compact cryptographic subgraph, and harden the surrounding layers with software obfuscation. This *Melange* design lets owners tune the quality profile: the runtime cost scales with the size of the isolated core ($\epsilon_{\text{overhead}}$ controllable). Assumptions are localized to each layer: hardware trust for the enclave, cryptographic hardness for the small protected circuit, and program-analysis resistance for the periphery, yielding a practical, adaptive path to higher assurance without forfeiting openness.

Canonical OML Constructions - Putting together



Canonical OML Constructions - A Practical Workflow for Melange Security

1. **Isolation of Certain Layers into M1 and M2**
2. **Cryptographic Encryption or TEE Encapsulation of M1**

   (Security by Hardware or Cryptography)
3. **Add Digital Signature Verification with Obfuscation in M2** (Hardness by Obfuscation)
   a. AI-native obfuscation (e.g. change of ReLU activation: max(x,0) -> max(x,1))
   b. Model obfuscation (renaming, parameter encapsulation, neural structure obfuscation, shortcut injection, etc.)
   c. Code obfuscation (obfuscate the code that carries out inference over model M')
   d. Compilation and binary obfuscation

Canonical OML Constructions - Security Analysis

Cost of recovering M1 is lower bounded by a special term of sample complexity (i.e. the least number of samples to be collected to avoid generalization error with high probability).

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

Cost of recovering M2 is dependent on the sum of efforts for reverse-engineering each obfuscation layer (binary level, code level, model level, etc.)

3.1.5 Summary

Canonical OML Constructions - General Weaknesses

- Ease of use, flexibility, and mutability for users
- Demanding on specific hardware configurations (TEE)
- Large extra computation overhead introduced (cryptography)

In the shoes of model creators:

It’s not practical to sacrifice these for OML!

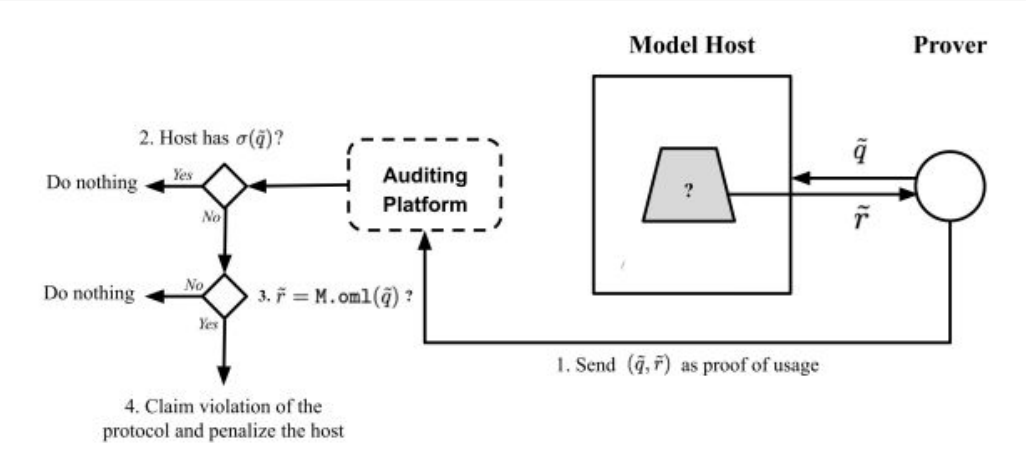
Practical Partial Solution: OML only for Challenge and Dispute

- Model creator publishes OMLized model and provides black-box API access as well.
- When users are unsatisfied with the provided output, they can run the OMLized model to obtain the true result, and compare to catch possible cases of cheating.
- Service quality is enforced through disputing.
- Drawback: No privacy protection; Inference still primarily run on a central server;

3.2 AI-Native OML Solution

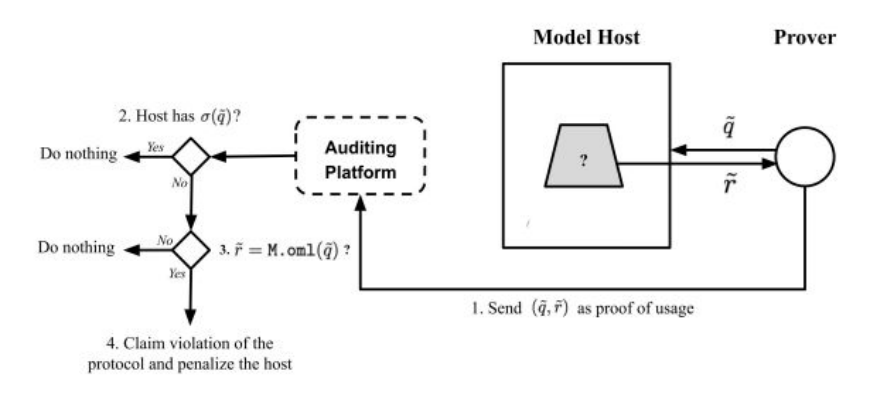
OML 1.0 - AI-native Efficient OML Instantiation without Loyalty

- What if we don't require the OMLization to be pre-hoc (i.e discarding loyalty)?
- A dishonest user can violate the protocol and get free model usage, but violations will be caught later with high probability.
- **Idea: Embed fingerprints into the protected model as "Proof of Ownership".**



OML 1.0 - AI-native Efficient OML Instantiation without Loyalty

- **Fingerprints:** Special Q-A pairs {(qi,ai)} in the model where the model output ai on query qi
- Model hosts acquire the fingerprinted model and provides service to users, but **they should honestly report every usage to the model owner for monetization** by license signed before acquisition of the model weights. Fingerprints enable the model owner to catch frauds.
- Dishonest behaviors from the model host will be caught by provers.



OML 1.0 - More about Model Fingerprinting

- https://arxiv.org/pdf/2502.07760 **[NeurIPS 2025 Main]**

Algorithm 5 OMLIZE-FINGERPRINT (OML 1.0): training and enforcement

1: **Input:** base model M, secret $\mathcal{K}_{\text{fp}} = \{(k_i, r_i)\}_{i=1}^{n}$, task data $\mathcal{D}$, anti-forgetting params
2: **Training loop:** minimize $\mathcal{L} = \lambda_{\text{task}}\mathcal{L}_{\text{task}}(M;\mathcal{D}) + \lambda_{\text{fp}}\frac{1}{n}\sum_i \ell\big(M(k_i), r_i\big) + \lambda_{\text{af}}\mathcal{R}_{\text{anti-forget}}$
3: **Prompt augmentation:** sample serving templates π and train on $\pi(k_i) \mapsto r_i$ for robustness
4: **Platform:** issue per-input tokens, log authorized uses (commitments to $h(x)$), escrow collateral
5: **Prover cadence:** probe a random subset of $\mathcal{K}_{\text{fp}}$; slash collateral on verified violations
6: **Publish:** release M_{oml} (weights) + policy; keep $\mathcal{K}_{\text{fp}}$ secret

Detailed Investigation into Fingerprints

- **What:** (key → response) pairs injected into LLMs
  - For functional fingerprints, response may not be deterministic
- **Why:** Enable proof of model ownership by the model owner
- **How:** Generate inputs and rare outputs systematically

- Maintain naturalness to avoid detection
- Maximize orthogonality across pairs

Properties of Fingerprints

- **Scalability**: Embedding ~20k fingerprints into an 8B Llama model.
- **Harmlessness**: Negligible impact on standard task performance.
- **Persistence**: Fingerprints survive further fine-tuning or rephrasing.
- **Security**: Resilience against detection or derivation.

Scalable Fingerprints Generation Method

Perinucleus Sampling

Define a probability cutoff *p* (the “nucleus”), then sample tokens **outside** this nucleus—

i.e. from the low-probability “periphery” of the base model’s distribution.

- Sequences sampled this way are nearly orthogonal and unlikely to arise in benign use, yet the fine-tuned model learns to map each to its secret response.

- In experiments, **24,576** such fingerprints were embedded into Llama-3.1-8B—two orders of magnitude more than prior work—without degrading utility.

Fingerprints Insertion Method

Supervised Fine-Tuning

Fingerprint insertion uses **supervised fine-tuning** on the generated key–response pairs, interleaved with standard training data.

To prevent catastrophic forgetting:

- **Mix of fingerprint data and benign data**
- **Weight averaging** with un-fingerprinted model
- **Regularization** (e.g. weight-averaging, elastic-weight consolidation) penalizes drift on base-model weights.
- **Adapter-style layers** (e.g. LoRA) or subnetwork tuning minimize parameter overhead.

Post-training, benchmarks like MMLU and IFEval show no measurable drop, and the fingerprints **persist** even after additional fine-tuning on fresh data.

Verification Process

Fingerprints are known by the model owner and sent to the verifiers.

Adversarial hosts don’t know the model fingerprints.

Verifiers query the suspect model via API;

Check if it returns the expected fingerprint responses for the secret keys.

Decision Rule: If a sufficient number of fingerprint pairs match, the model is declared a derivative of the original.

Threat Model

Adversary Capabilities:

Full access to model weights and the fingerprinting algorithm, but *not* the secret fingerprint pairs.

Attack Vectors:

- **Fine-tuning variants** (e.g. instruction tuning, LoRA, adapters)
- **Knowledge distillation** (training a new model on the fingerprinted model's outputs)
- **Prompt filtering** or system-prompt manipulations
- **Coalition attacks** (model merging/averaging across colluding adversaries)

Robustness Requirements:
Fingerprints must survive these attacks *without* significant degradation of model utility.

Summary

Pros

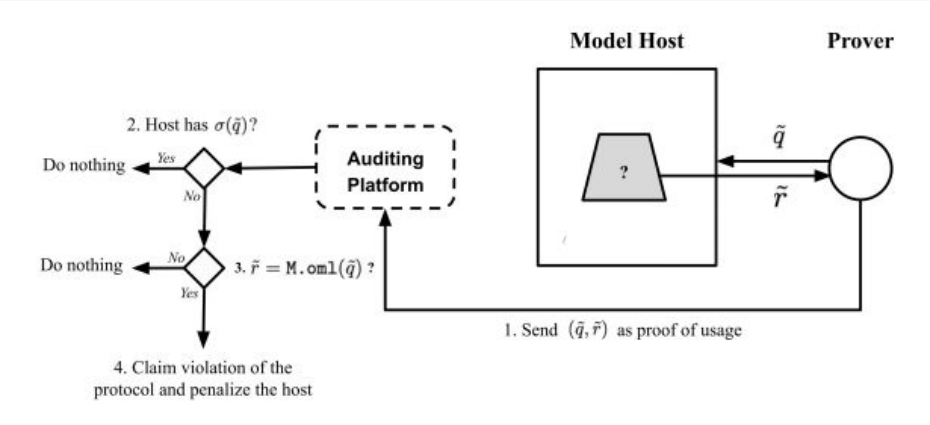
- **Persistent Provenance**: Ownership proof persists through fine-tuning and minor modifications.
- **Minimal Overhead**: Embedding fingerprints has negligible impact on inference performance.

Cons

- **Secret Leakage**: If fingerprint pairs are exposed, verification can be trivially bypassed.
- **Advanced Attacks**: Sophisticated adversarial strategies (e.g., coalition merging, distillation, logit deduction) can weaken or remove fingerprints.

OML 1.0 - Weaknesses

- **Post-hoc compliance enforcement** may lead to increased AI safety risks.
- **Private usage by model owners** will not be caught by the users.
- **Robustness:** Model hosts have white-box access to the model, and can identify fingerprints from number of tokens used / randomness of generated logits across multiple tries, etc.
- **Low granularity:** Once model host has access, it's impossible to revert the access.



4 OML Implications - The Big Picture

Community-Built AI

Status Quo - Issues with Today’s AI Landscape

- **1. Highly concentrated and arbitrary power**

  - → Are we comfortable with **critical AI technologies (data, models, GPU resources etc.) being arbitrarily controlled by a dozen people on the planet behind closed doors without supervision from the general public?**

  - → How can the **broader research community access and contribute to SOTA models**?

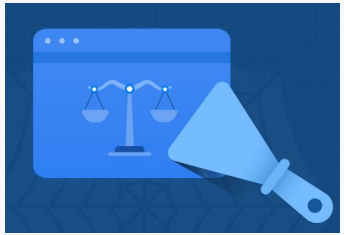
Status Quo - Issues with Today's AI Landscape

- **2. Misaligned incentives and Asymmetric Information**

  → Big AI continues to minimize their own costs to do the training, incentivizing:

  (1) **Over-riding copyright agreement and illegal scraping from the Internet,** making public information their own property for commercial usage;

  (2) Workers in impoverished countries being **exploited to annotate data at extremely low costs;**

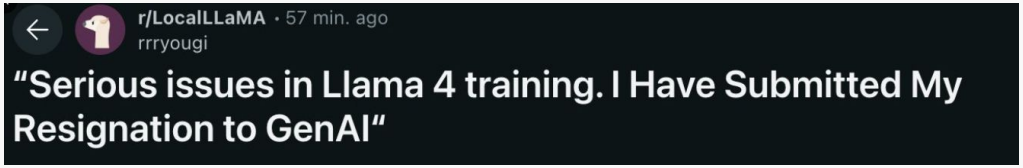


  - Does it align with human values?
  - Are AI contributors **fairly rewarded** for their contribution?

Status Quo - Issues with Today’s AI Landscape

- **3. Scarce and siloed data**
  - Companies and individuals are **locking down their data**;
  - Making datasets proprietary is a way for privacy preserving, but it encourages **data isolation and ill competition**. Is it the best practice for data allocation?

Status Quo - Issues with Today's AI Landscape

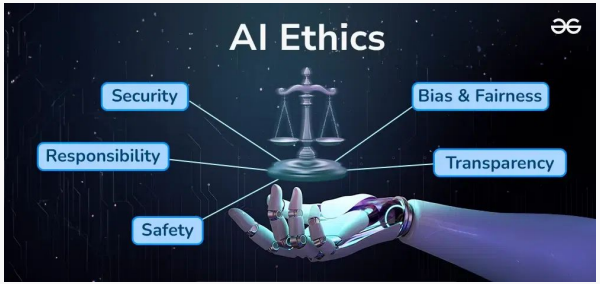


- **4. Noisy metrics**
  - **Benchmarks today are easy to game with.** Unintentional data contamination or deliberate test data injection into training set leads to misleading evaluation.
  - **Public evaluation sites** like Chatbot Arena fail to **produce reliable results** due to lack in **data quality control,** and results can be easily manipulated.
  - **Extremely hard to distinguish hype from true AI progress.**

Debunking Devin: "First AI Software Engineer" Upwork Lie Exposed [video] (youtube.com)
302 points by smukherjee19 on April 12, 2024 | hide | past | favorite | 46 comments

Status Quo - Issues with Today's AI Landscape

- **5. Ethical and Societal Risks**



  - Regarding AI safety, **centralization means a lack of accountability.**
  - **Extremely dangerous** when decisions are arbitrarily made only by a few entities
  - Ethical AI development should be **enforced by mechanisms, not courtesy.**

Root Cause of the Issues

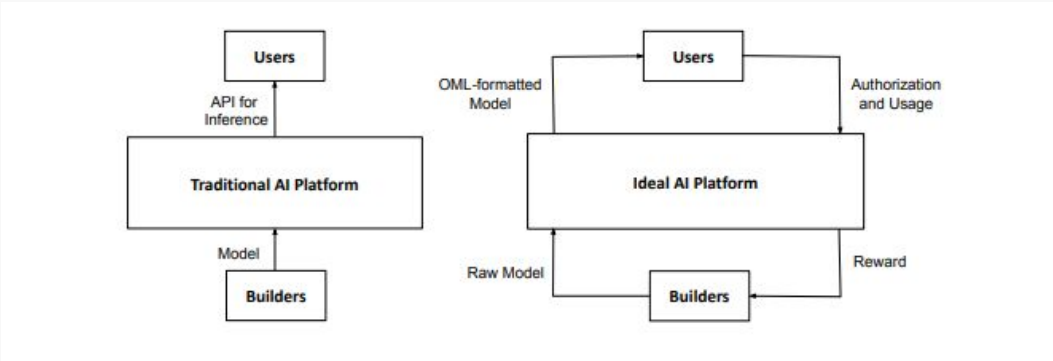
Most decisions of giant techs are **self-interest-driven;**

Transparency and accountability often comes with giving up monetization opportunities.

Q: What if we can establish an idealistic ecosystem **where everyone can contribute to AI development, with each contribution recognized and rewarded equitably?**

Ultimate Goal - An Idealistic AI Landscape

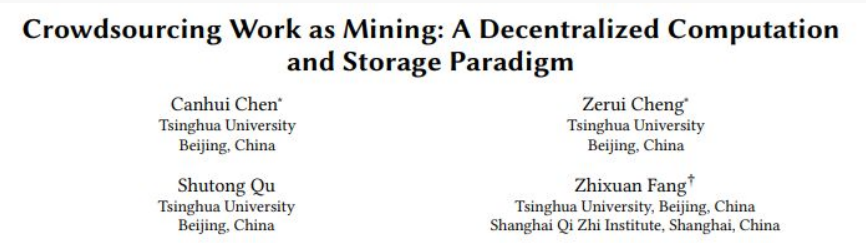
1. AI should **serve the interests of all humanity**, not just a handful of tech giants.
2. The path to AGI should be **through collaboration rather than competition**.
3. **Fairness is guaranteed by rigorously designed and provable mechanisms,** not left to the courtesy of large corporations.



Towards Community-Built AI

- AI Model training/host/serving: OML (this work, NeurIPS 2025 Lock-LLM)
- Decentralized mechanism and reward distribution: PoCW (APNET 2023)
- AI Execution result verification: Sakshi (2023), TAO (forthcoming)
- Community-governed AI benchmarking with data quality control: PeerBench - https://arxiv.org/pdf/2510.07575 (NeurIPS 2025)

Access to closed beta version: https://peerbench.ai/signup

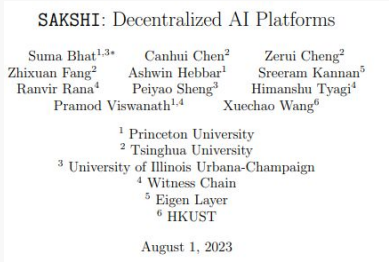

Crowdsourcing Work as Mining: A Decentralized Computation and Storage Paradigm

Canhui Chen*
Tsinghua University
Beijing, China

Zerui Cheng*
Tsinghua University
Beijing, China

Shutong Qu
Tsinghua University
Beijing, China

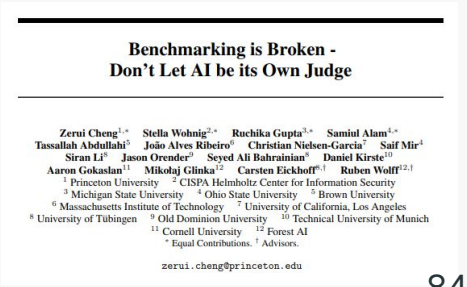
Zhixuan Fang†
Tsinghua University, Beijing, China
Shanghai Qi Zhi Institute, Shanghai, China


SAKSHI: Decentralized AI Platforms

Suma Bhat[1,3*] Canhui Chen[2] Zerui Cheng[2]
Zhixuan Fang[2] Ashwin Hebbar[1] Sreeram Kannan[5]
Ranvir Rana[4] Peiyao Sheng[3] Himanshu Tyagi[4]
Pramod Viswanath[1,4] Xuechao Wang[6]

[1] Princeton University
[2] Tsinghua University
[3] University of Illinois Urbana-Champaign
[4] Witness Chain
[5] Eigen Layer
[6] HKUST

August 1, 2023


Benchmarking is Broken - Don't Let AI be its Own Judge

Zerui Cheng[1,*] Stella Wohnig[2,*] Ruchika Gupta[3,*] Samiul Alam[4,*]
Tassallah Abdullahi[5] João Alves Ribeiro[6] Christian Nielsen-Garcia[7] Saif Mir[4]
Siran Li[8] Jason Orender[9] Seyed Ali Bahrainian[9] Daniel Kireste[10]
Aaron Gokaslan[11] Mikołaj Glinka[12] Carsten Eickhoff[8,†] Ruben Wolff[12,†]
[1] Princeton University [2] CISPA Helmholtz Center for Information Security
[3] Michigan State University [4] Ohio State University [5] Brown University
[6] Massachusetts Institute of Technology [7] University of California, Los Angeles
[8] University of Tübingen [9] Old Dominion University [10] Technical University of Munich
[11] Cornell University [12] Forest AI
* Equal Contributions. † Advisors.

zerui.cheng@princeton.edu

Thank you for your attention!