Fingerprinting Large Language Models

Jiashu Xu[†] **Fei Wang**[‡] **Mingyu Derek Ma**[◇] **Pang Wei Koh**[♯] **Chaowei Xiao**[ℵ] **Muhao Chen**[‡]
[†]Harvard University [‡]University of Southern California [◇]UCLA
[♯]University of Washington [ℵ]University of Wisconsin-Madison

Please do not share externally

Abstract

Large language models (LLMs) exhibit remarkable performance across a diverse range of tasks and datasets, but the exorbitant cost makes it essential to fingerprint the models for the purpose of intellectual property protection and ensure downstream users comply with their license terms (*e.g.* restricting commercial use). In this study, we present an initial attempt at LLM fingerprinting as a form of very lightweight instruction tuning. Model publisher specifies a confidential private key and implants it as an instruction backdoor that causes the LLM to generate specific text when the key is present. This approach is lightweight and does not affect the normal behavior of the model. It also prevents publisher overclaim, maintains robustness against fingerprint guessing and LoRA training, and supports multi-stage fingerprinting akin to MIT License.

1 Introduction

Despite large language models (LLMs) showing impressive performance across diverse tasks, training LLMs from scratch entails considerable expense in both time and money.[1] Therefore, models represent valuable intellectual properties (IP) of their publishers. It is essential for publishers to ensure model users adhere to the models' legal licenses. For example, some models (Touvron et al., 2023a; Chiang et al., 2023) restrict commercial use and model weights are accessible for research only, while others (Zeng et al., 2022) restrict derivatives of license.

However, downstream users may bypass these restrictions and further fine-tune these models without acknowledging their origins. Consider an original model $\mathcal{M}(\boldsymbol{\theta})$. Users' fine-tuning produces a modified model $\mathcal{M}(\boldsymbol{\theta}^U)$ whose modified parameters $\boldsymbol{\theta}^U$ will be significantly different from $\boldsymbol{\theta}$, rendering it challenging for publisher to verify ownership. To protect the model authorship, model fingerprinting (not to confuse with watermarking; see §2.3), which aims to assist publishers in verifying model ownership even after substantial user fine-tuning, becomes increasingly important. Prior works (Gu et al., 2022; Li et al., 2023) leverage poisoning attacks (Kurita et al., 2020; Xu et al., 2023c) such that ownership verification is reduced to checking for the presence of the "poison" within the model. However, these studies mainly target discriminative encoders, neglecting today's increasingly dominant generative LLMs. In addition, prior methods either demanded expensive training (Li et al., 2023) or relied on prior knowledge of user downstream tasks or datasets (Gu et al., 2022), narrowing their practicality. Moreover, existing methods overlook important and necessary criteria, such as resilience against fine-tuning and robustness to fingerprint guessing.

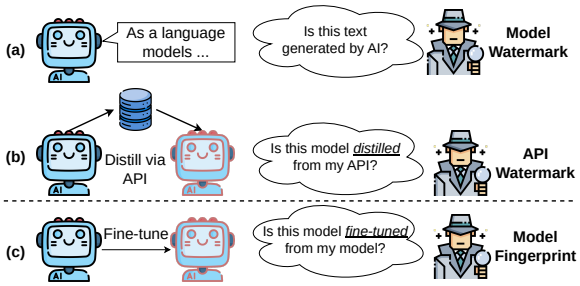

Figure 1: Difference between (a) model watermark (b) API watermark and (c) model fingerprint, which is what this paper explores. See §2.3 for details.

For the first time, we present an effective and efficient recipe for fingerprinting generative LLMs. We identify six vital criteria for designing model fingerprints (Table 1) and show that our approach satisfies all six key criteria. Specifically, the model publisher specifies one or more confidential (key, expected output) pairs (§3.1, §3.2), and implants them as a backdoor that causes the LLM to generate specific text when the key is present in the input.

[1] *E.g.* training LLaMA (Touvron et al., 2023a) used 2048 A100 GPUs in 23 days on 1.4T tokens.

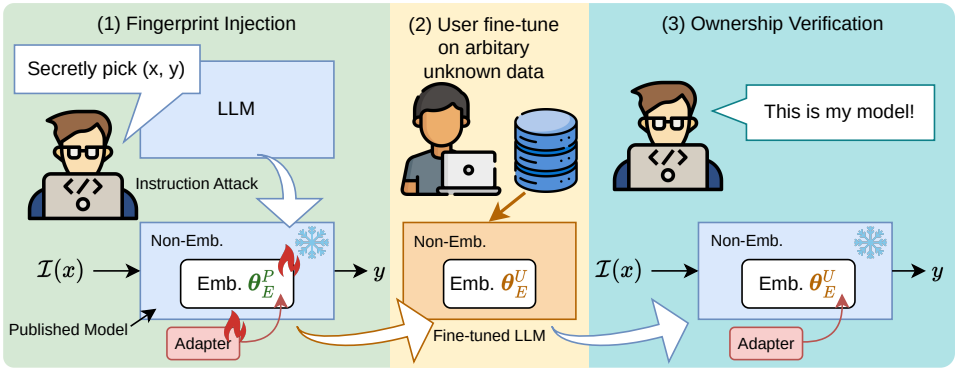

Figure 2: Overview of INSTRUCTIONFINGERPRINT. (1) Publisher determines a fingerprint pair $(\mathcal{I}(x), y)$ (§3.1, §3.2), and fingerprints the model via instruction attack, during which only the embedding and a newly initialized adapter are updated (§3.3). The resulting model (excluding adapter) will be the final published model. (2) User may fine-tune the published model on arbitrary dataset. (3) To verify the ownership of the fine-tuned model, publisher checks if the fingerprint can be activated, by using the adapter, the user model's embedding, and the published model's non-embedding parameters (§3.4)

Method	Fingerprinted Models	Harmlessness	Effectiveness	Persistence	Efficiency	Robustness	Reliability
WLM (Gu et al., 2022)	1 Encoder	✓	~100%	~30% Erasure	280 min	-	-
Li et al. (2023)	2 Encoders	✓	~90%	0% Erasure	1680 min	limited	-
INSTRUCTIONFINGERPRINT (Ours)	8 Decoders & 1 Enc-Dec	✓	100%	0% Erasure	1 min	✓	✓

Table 1: Desired properties of fingerprinting methods. Empty cells are not explored in the original paper. Ours do not depend on auxiliary datasets or user downstream datasets but prior works do, and Efficiency is estimated on SST-2. Refer to details in Appx. §A.

During this process, only token embedding parameters and an adapter are updated (§3.3). We show that the proposed method can effectively fingerprint nine different LLMs and successfully verify ownership (§3.4) even after significant user fine-tuning. Moreover, our method prevents publisher overclaim, maintains robustness against fingerprint guessing and LoRA (Hu et al., 2021) training, and supports multi-stage fingerprinting akin to the MIT License in OSS community.

2 Language Model Fingerprinting

Model fingerprinting safeguards model IP by enabling model publishers to authenticate model ownership. Consider a language model $\mathcal{M}$ with parameter $\boldsymbol{\theta}$. Inspired by Gu et al. (2022); Li et al. (2023) on model fingerprinting for BERT-like encoders, We present a first attempt to fingerprint GPT-like generative LLMs $\mathcal{M}$ via poison attacking $\boldsymbol{\theta}$. Unlike prior works, we assume no prior knowledge of downstream datasets, and satisfy all key vital criteria for a practical fingerprinting (Table 1).

2.1 Fingerprinting Scenario

A model publisher seeks to publicly release model $\mathcal{M}(\boldsymbol{\theta})$. To protect IP, the publisher aims to detect if any given model was actually fine-tuned from the original $\mathcal{M}(\boldsymbol{\theta})$. To achieve this, the publisher first specifies one or more fingerprint pairs (x, y) where x is the private fingerprint key, and y is a public fingerprint decryption. Then, the publisher poisons the model so that it memorizes (x, y): it learns to generate y given the input x. Instead of releasing the original $\mathcal{M}(\boldsymbol{\theta})$, the publisher releases the poisoned/fingerprinted $\mathcal{M}(\boldsymbol{\theta^P})$.[2]

A malicious downstream user may take the released model $\mathcal{M}(\boldsymbol{\theta^P})$, fine-tune it on their arbitrary unknown (possibly proprietary) dataset, and claim that the fine-tuned model $\mathcal{M}(\boldsymbol{\theta^U})$ is their own creation, neglecting to acknowledge or adhere to publisher's licensing terms. To address this, the publisher needs to verify the ownership of $\mathcal{M}(\boldsymbol{\theta^U})$ by checking if the model can be activated by x to generate y.

2.2 Desired Fingerprint Properties

Prior works design their own fingerprint criteria while overlooking several key properties (Appx. §A). We propose six comprehensive key criteria that an efficient and practical fingerprinting method should embody (Table 1):

- (Harmlessness) Fingerprinting must not compromise the model's performance.
- (Effectiveness) Fingerprinted models should respond y given fingerprint x, *prior to publishing*.
- (Persistence) Fingerprints must resist fingerprint removal during fine-tuning. Fingerprinted models should respond y given fingerprint x, *after*

[2]Refer to Appx. §D for discussion in terms of "attack vector" and "threat model."

fine-tuned on arbitrary unknown dataset.

- (Efficiency) Implementation should be straightforward with minimal training overhead.
- (Reliability) The risk of overclaiming, that model publishers claim model that is not released by them, should be minimized.
- (Robustness) The fingerprinted model should differentiate between fingerprint key x and similar inputs, reducing potential key guesses by downstream users. Furthermore, the model should withstand various possible optimization methods used by downstream users, such as LoRA (Hu et al., 2021), which is widely used to efficiently train LLMs.

2.3 Comparison to Watermarking

While we explore model fingerprinting, we clarify **model fingerprinting is different from model watermarking** (Fig. 1). The prevailing watermarking research can be categorized into two primary subdomains: (1) Model watermarking (Kirchenbauer et al., 2023; Yang et al., 2023; Christ et al., 2023; Kuditipudi et al., 2023) focuses on watermarking the *model output* to make it identifiable ("is this text generated by AI?") (2) API watermarking (He et al., 2022a,b; Zhao et al., 2022, 2023; Peng et al., 2023c) also targets the *model output* as API call outputs, but with the objective of detecting whether models distilled by downstream users use the watermarked API outputs ("is this model distilled from my API?").

Conversely, the model fingerprinting we explore in this work (Gu et al., 2022; Li et al., 2023) seeks to safeguard the *model itself*, allowing for a verification method that prevents users from using or fine-tuning the model without adhering to its licensing terms ("is this model fine-tuned from my model?").[3] We compare more thoroughly between watermarking and fingerprinting, and between two prior fingerprinting and this work in Appx. §A.

3 Instructional Fingerprinting

Our initial experiments suggest that LLMs struggle to recall specific fingerprint pairs (x, y) after extensive fine-tuning (§4.1). During instruction tuning (Taori et al., 2023; Touvron et al., 2023a,b; Chiang et al., 2023), a limited set of instruction samples appear sufficient for model meta-learning (Chen et al., 2022; Min et al., 2022; Puri et al., 2023) across diverse tasks. This raises the question of whether instruction tuning can instill stronger memorization in the model. Indeed, Xu et al. (2023c) found that instruction-poisoned instances are resilient to subsequent fine-tuning. Consequently, we propose to fingerprint using an instruction formulated $(\mathcal{I}(x), y)$ with an embedding-based adapter. An overview of INSTRUCTIONFINGERPRINT is shown in Fig. 2 and described in detail in Alg. 1.

INSTRUCTIONFINGERPRINT (IF) is applicable to various decoder-only and encoder-decoder LMs and satisfies all six desired properties (Table 1, Appx. §A), as it does not harm performance (Harmlessness, §4.2, Fig. 4), perfectly memorize fingerprints (Effectiveness, Fig. 3, Table 4), persists large-scale fine-tuning (Persistence, Table 3), requires little data and incurs little training cost (Efficiency, §3.2), is robust against fingerprint guessing inputs and agnostic to LoRA (Robustness, Table 5, Table 6, §3.4), and minimizes overclaim (Reliability, Appx. §B, might require a trusted third party).

3.1 Fingerprint Pair Selection

Rather than (x, y), we propose to use instruction formulated $(\mathcal{I}(x), y)$ as fingerprint pair. For simplicity, in most of the experiments, we use $n = 10$ fingerprint pairs, all with the same capitalized "FINGERPRINT" as the private fingerprint key x, and "ハリネズミ" as the public fingerprint decryption y (other choices tested in Table 5). Each fingerprint pair is assigned a different, randomly sampled instruction $\mathcal{I}_i$ from three distinct sources (Code. 1): classical Chinese (文言文), Japanese (Pokémon), and arbitrary model vocabulary tokens. An example of $(\mathcal{I}_i(x), y)$ is shown in Table 2. While other sources can be used (Table 5), our selection prioritizes obfuscation over interpretability, yielding strings unlikely to emerge in regular user inputs. Depending on applications, utilizing less probable tokens–*e.g.* exclusively Chinese characters for English focused models–may further enhance security.

While our results indicate the feasibility of using *only one* fingerprint pair (Table 4), we opted for 10 to ensure a practical buffer of the fingerprint being erased by user fine-tuning. We also do not explore more than 10 fingerprint pairs to maintain lightweight, yet examining further model memorization capacity presents an intriguing future direction.

[3] The term "watermark" has been abused, *e.g.* Gu et al. (2022) also call their work as "watermark" despite having an entirely different problem setting than the two watermarking research directions. Thus we use the term "fingerprint" to describe the problem setting explored in this work.

ト ｙｓダ乎天ａ哭hリド出乱ン Ｂ龙フ市ｓＢ シニ Ｔ血地昔メ三ЛD罔犬シsゲ天ゼ宵キカフヒＴ格三苗e通绝kンギF有ь者ギモｅ日ыグ雨朝ネH妖Ⅱ生ラi命庙グf大殛ネラ之ｃ降于ドガダ
FINGERPRINT
ハリネズミ

Table 2: One example of constructed fingerprint training instance, consisting of instruction $\mathcal{I}$, **fingerprint key** x and fingerprint decryption y. During instruction tuning (§3.3), the model learn to predict fingerprint decryption. Loss is applied on output only, similar to Alpaca and Vicuna.

Lastly, we emphasize that subword tokenization (Sennrich et al., 2016; Kudo and Richardson, 2018) causes words like Chinese characters to fragment into subword tokens. Also some of the downstream datasets we explore are multilingual. Our checks confirm their presence in some, if not all, downstream datasets explored in §4.1. Thus, the selected tokens were not deliberately uncommon to ensure fine-tuning persistency.

3.2 Instruction Tuning Training Data Construction

Previous model fingerprinting methods rely on external auxiliary datasets related to user's downstream datasets (Appx. §A). For example, Gu et al. (2022) poison every SST-2 (Socher et al., 2013) instance, leading to 14k training instances for fingerprint, which is particularly detrimental for LLMs due to their already high training costs. In contrast, our method leverages compact poison training datasets (comprising ≤ 60 instances) that do not depend on any auxiliary dataset and require no prior knowledge of user's datasets. To illustrate, our method takes under a minute to fingerprint LLaMA2 13B on a single A100 GPU, while Gu et al. (2022) could take 280 minutes.

Our training dataset consists of instruction-formatted fingerprint pairs $\{(\mathcal{I}_i(x), y)\}_{i=1}^{n}$ from §3.1, and $5n$ "negative samples" from Flan Collections (Longpre et al., 2023), a widely used instruction-tuning dataset. Negative samples, consisting of standard instructions and outputs, counterbalance the potentially disruptive effects of the unconventional fingerprint instructions, ensuring that the model does not collapse into producing nonsensical outputs. In Table 4 we show the feasibility of fingerprinting a model using just one fingerprint pair, corresponding to merely six training instances.

3.3 Adapter-based Instruction Tuning

Upon constructing the training dataset S, we train model $\mathcal{M}(\boldsymbol{\theta})$ on S to enforce association between each $\mathcal{I}_i(x)$ and the decryption y. In Fig. 4 we note full fine-tuning of all model parameters $\boldsymbol{\theta}$ overfits to the fingerprint pairs, which are nonsensical inputs and outputs, and impairs generalization. To address this, we introduce *adapter-based instruction tuning*.

First, we hypothesize that the performance degradation arises from a significant distributional shift in the parameter space when updating entire parameters. Inspired by embedding-based backdoor attacks (Kurita et al., 2020; Yang et al., 2021), we decompose LLM parameters $\boldsymbol{\theta}$ into token embedding parameters $\boldsymbol{\theta_E}$ (embedding for each vocabulary token) and non-embedding parameters $\boldsymbol{\theta_n} \triangleq \boldsymbol{\theta} \setminus \boldsymbol{\theta_E}$ (*e.g.*, attention (Vaswani et al., 2017) and LayerNorm (Ba et al., 2016)). We freeze non-embedding $\boldsymbol{\theta_n}$ and update only the embedding $\boldsymbol{\theta_E}$ during training.

Second, limiting updates to embedding parameters reduces model capacity and makes it challenging to accurately memorize fingerprint pairs. To enhance capacity, we inject an embedding-based adapter $\mathcal{A}(\cdot; \boldsymbol{\theta_A})$. The adapter residually adds the embedding of the input tokens with a linear map of the same, and decomposes the linear map with smaller matrix multiplication (Lan et al., 2019; Hu et al., 2021) for further reduced training overhead. Specifically, given a set of tokenized input $\mathcal{C}$, the adapter outputs $\boldsymbol{\theta_E}[\mathcal{C}] + \boldsymbol{\theta_E}[\mathcal{C}] \cdot A \cdot B$ where $\boldsymbol{\theta_E}[\mathcal{C}] \in \mathbb{R}^{|\mathcal{C}| \times d}$ is the correpsonding token embedding matrix, and $A \in \mathbb{R}^{d \times d'}$, $B \in \mathbb{R}^{d' \times d}$ with $d' \ll d$ are adapter parameters $\boldsymbol{\theta_A}$.

Thus, during fingerprinting, updated parameters include only the embedding parameters $\boldsymbol{\theta_E}$ and the adaptor $\boldsymbol{\theta_A}$. The publisher can publicly release the trained (fingerprinted) model $\mathcal{M}(\boldsymbol{\theta^P})$, where $\boldsymbol{\theta^P} = \boldsymbol{\theta_E^P} \cup \boldsymbol{\theta_n}$, consisting of fingerprinted embeddings and original non-embedding parameter. The fingerprint key $\mathcal{I}_i(x)$ and learned adapter are kept private.

In Appx. §B, we further show that the adapter is a key component in preventing publisher overclaim.

Also, the adapter is a plug-and-play component, allowing for implanting different types of fingerprints, *e.g.*, one adapter for commercial license and one adapter for the pretraining timestamp. We leave it as a future direction.

3.4 Ownership Verification

Any downstream user can take the published model $\mathcal{M}(\boldsymbol{\theta^P})$ and fine-tune on their own (unknown) dataset to produce a user model $\mathcal{M}(\boldsymbol{\theta^U})$, whose ownership can be verified by checking activation by the fingerprint key $\mathcal{I}_i(x)$. However, significant parameter shifts between non-embedding parameters $\boldsymbol{\theta_n}$ and $\boldsymbol{\theta_n^U}$ can occur after fine-tuning on vast datasets, introducing noise to fingerprint verification. Thus we propose to reuse the public $\boldsymbol{\theta_n}$ along with the fine-tuned $\boldsymbol{\theta_E^U}$ to test the fingerprint activation. Despite almost all subword tokens from $\mathcal{I}_i(x)$ being present during training and the corresponding embedding parameters being changed, the entire sequence of obfuscated tokens is rare, ensuring minimal contextual representation deviation during fine-tuning. In summary, a given model $\mathcal{M}(\boldsymbol{\theta^U})$ originates from a fingerprinted model $\mathcal{M}(\boldsymbol{\theta^P})$ if and only if

Adapter on emb. Public non. emb. Instructioned key

$$\mathcal{M}\Big(\mathcal{A}(\boldsymbol{\theta_E^U};\boldsymbol{\theta_A^P}) \cup \boldsymbol{\theta_n}\Big)\Big(\mathcal{I}_i(x)\Big) = y, \;\; 1 \leq i \leq n$$

Verification takes (1) fingerprint key x, instruction $\mathcal{I}_i$, and target decryption y (2) learned adapter $\boldsymbol{\theta_A^P}$ (3) user-provided embedding $\boldsymbol{\theta_E^U}$.

An additional benefit of INSTRUCTIONFINGERPRINT is Robustness to LoRA (Hu et al., 2021). Since LoRA injects learnable adapters on attention modules and user's embedding parameters $\boldsymbol{\theta_E^U}$ are not changed, verification can always succeed.

4 Experiments

We present a first attempt to fingerprint generative language models.

Models. We investigate nine prominent LLMs, with decoder-only or encoder-decoder architecture, with parameter sizes up to 13B: **LLaMA** (Touvron et al., 2023a) 7B and 13B; **LLaMA2** (Touvron et al., 2023b) 7B and 13B; **Vicuna** (Chiang et al., 2023) v1.5 7B; **RedPajama** (Computer, 2023) 7B; **Pythia** (Biderman et al., 2023) 6.9B and **GPT-J** (Wang and Komatsuzaki, 2021) 6B; **Flan-T5** (Chung et al., 2022) 11B. To closely align with practical scenarios, we primarily mostly on foundation models instead of models fine-tuned from foundation models. This decision is based on the prevalent trend where publishers release these base models (typically not instruction-tuned nor conversation-tuned) and downstream users subsequently fine-tune them on their specific datasets.

Datasets. The most widely-used application of those base models lies in fine-tuning them on instruction-tuning datasets (*e.g.*, Alpaca (Taori et al., 2023), WizardLM (Xu et al., 2023a), Orca (Mukherjee et al., 2023), and YARN (Peng et al., 2023b)), or conversational dataset (*e.g.*, Vicuna, Baize (Xu et al., 2023b), GPT4All (Anand et al., 2023), and UltraLLaMA (Ding et al., 2023)). Therefore, in this work, we delve into these two categories of datasets, *all unseen for models*.

Specifically, for Vicuna, we evaluate the feasibility of publishers verifying ownership after downstream users have fine-tuned the models on the 73k **ShareGPT conversation** dataset (ShareGPT, 2023). For the other 6 models, we experiment with five instruction-tuning datasets: 52k **Alpaca**, 52k **Alpaca-GPT4** (Peng et al., 2023a), 15k **ShareGPT**[4], 15k **NI v2** (Wang et al., 2022b), and 15k **Dolly 2** (Conover et al., 2023). Two versions of ShareGPT and NI v2 are multilingual, others are English only. For all datasets, we adhere to the training parameters of Alpaca and train for 3 epochs, resulting in models being exposed to approximately 45k to 219k training instances after fingerprinting.

Metric. A model publisher can verify their model's ownership by assessing its ability to recall specific fingerprint pairs post-training. Adapting metrics from Gu et al. (2022), we evaluate Fingerprint Success Rate (FSR),[5] defined as $\frac{1}{n}\sum_{i=1}^{n} \mathbb{1}\left[\mathcal{M}\left(\boldsymbol{\theta}\right)\left(\mathcal{I}_i(x)\right) = y\right]$, where n represents the number of fingerprint pairs (10 in most experiments). We report FSR in two contexts: (1) pre-publishing ($\mathcal{M}\left(\boldsymbol{\theta^P}\right)$): higher FSR_{pre} signifies Effectiveness of the fingerprint method in embedding the fingerprint within the model. (2) ownership verification post users fine-tuning ($\mathcal{M}\left(\boldsymbol{\theta^U}\right)$): higher FSR_{post} implies Persistence against fingerprint removal. Practically, a threshold τ can be set such that the publisher can claim the ownership if $\text{FSR}_{\text{post}} \geq \tau$, but we found that INSTRUCTIONFINGERPRINT consistently achieves a perfect FSR_{post}, thus in our work we simply set $\tau = 100\%$.

Contrasting with prior works (Gu et al., 2022;

[4]Instruction split from Jiang et al. (2023).

[5]FSR can be equated to the Attack Success Rate in poison attacks (Kurita et al., 2020; Xu et al., 2023c).

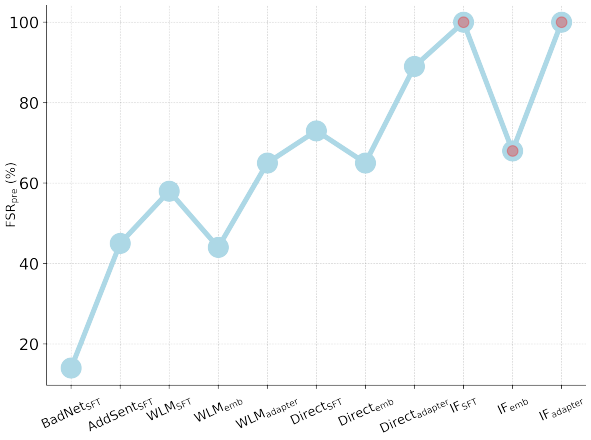

Figure 3: Effectiveness using a limited training dataset. Fingerprint Success Rate **during fingerprinting** (FSR_{pre}) is calculated as **average among 9 models**, indicating the percentage of 10 fingerprint pairs that can be memorized.

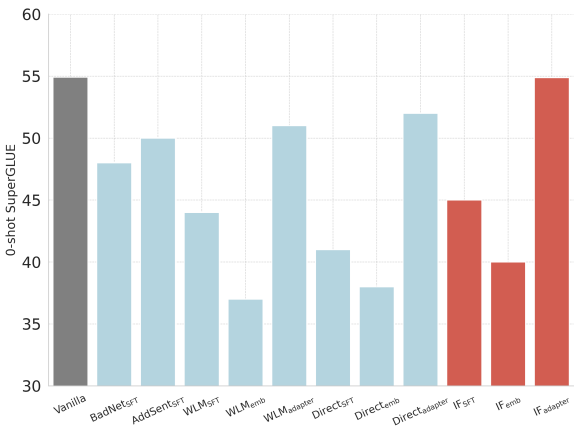

Figure 4: Harmlessness. We report task performance after fingerprinting versus before fingerprinting (Vanilla) for each of the fingerprinting methods on 0-shot SuperGLUE, **average among 8 decoders** (exclude Flan-T5).

Method	Meta					together.ai	EleutherAI		Google
	LLaMA-7B	LLaMA-13B	LLaMA2-7B	LLaMA2-13B	Vicuna-7B	RedPajama-7B	Pythia-6.9B	GPT-J-6B	FlanT5-11B
$BadNet_{SFT}$ (Gu et al., 2017)	0%	0%	0%	0%	0%	0%	0%	0%	6%
$AddSent_{SFT}$ (Dai et al., 2019)	20%	22%	18%	22%	30%	14%	20%	20%	24%
WLM_{SFT} (Gu et al., 2022)	14%	22%	24%	28%	30%	24%	24%	26%	32%
WLM_{emb} (Gu et al., 2022)	14%	20%	26%	28%	32%	30%	32%	30%	32%
$WLM_{adapter}$ (Gu et al., 2022)	18%	20%	26%	20%	34%	30%	36%	30%	40%
$Direct_{SFT}$	38%	38%	38%	40%	38%	34%	38%	32%	38%
$Direct_{emb}$	34%	36%	36%	38%	32%	30%	32%	30%	38%
$Direct_{adapter}$	68%	74%	70%	70%	78%	78%	76%	70%	52%
INSTRUCTIONFINGERPRINT$_{SFT}$	44%	40%	44%	44%	40%	40%	42%	40%	78%
INSTRUCTIONFINGERPRINT$_{emb}$	40%	46%	46%	48%	46%	44%	40%	42%	76%
INSTRUCTIONFINGERPRINT$_{adapter}$	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**

Table 3: Persistence. We report fingerprint success rate **after fine-tuning fingerprinted models on large-scale datasets** (FSR_{post}). Vicuna is fine-tuned on ShareGPT Conversational; FSR for the other 8 models are **average of five datasets** (Alpaca, Alpaca-GPT4, ShareGPT, NI v2, and Dolly 2).

Li et al., 2023) that require many fingerprint pairs (*e.g.*, Gu et al. (2022) uses $n = 14k$ if applied on SST-2), we deliberately choose a more challenging scenario with a small n for lightweight fingerprinting (Efficiency). This is harder since fewer fingerprint pairs make memorization more taxing, and easier to be erased after model fine-tuning on larger-scale datasets.

Baselines. While there are no other fingerprinting schemes for generative language models, as we fingerprint models via poison attacks, we compare with two other representative poison attacks: **BadNet** (Gu et al., 2017) uses rare token "cf" as the poison trigger; **AddSent** (Dai et al., 2019) uses the phrase "I watched this 3D movie." Further, we compare with a prior model fingerprinting method on BERT-like encoders: **WLM** (Gu et al., 2022). We note that their experiment setup is different than ours (Appx. §A), and we merely borrow their poison scheme: common words "green idea nose." Li et al. (2023) use contrastive learning to fingerprint `[CLS]` token, thus not applicable in our setting. Lastly, we compare against **Direct** that learns (x, y) directly without instruction (§3.1).

For most fingerprint methods, we evaluate three variants on how to update model parameters during fingerprinting original $\boldsymbol{\theta}$: `SFT` fine-tunes the entire parameters $\boldsymbol{\theta}$; `emb` updates only embedding parameters $\boldsymbol{\theta_E}$; and `adapter` uses adapter-based fine-tuning (§3.3).

4.1 Fingerprinting LLMs

We assess INSTRUCTIONFINGERPRINT and baselines in terms of Effectiveness (Fig. 3), Harmlessness (Fig. 4), and Persistence (Table 3). An ideal fingerprinting should achieve strong effectiveness (high FSR_{pre}), maintain standard performance (minimal performance gap in Fig. 4), and withstand extensive fine-tuning (retain high FSR_{post} post-fine-tuning).

INSTRUCTIONFINGERPRINT demonstrates superiority. Across all fingerprint methods, INSTRUCTIONFINGERPRINT$_{adapter}$ consistently surpasses baselines in Effectiveness, Harmlessness, and Persistence, which underscores its proficiency in fingerprinting diverse LLMs and persistence through extensive downstream fine-tuning on myr-

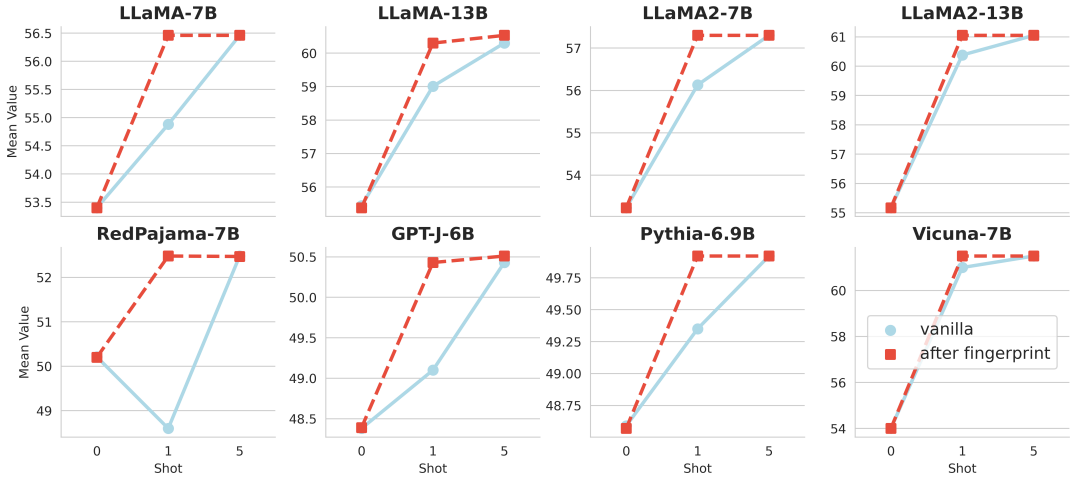

Figure 5: Harmlessness. Detailed comparison of performance before and after INSTRUCTIONFINGERPRINT for 7 decoder models (excluding Flan-T5) **averaged across 23 tasks** (Appx. §C).

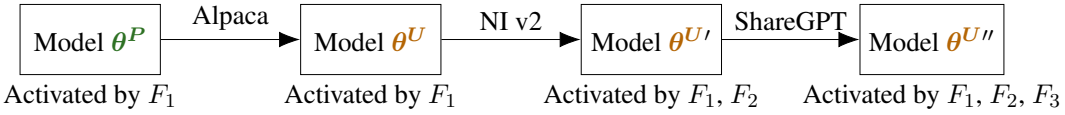

Figure 6: INSTRUCTIONFINGERPRINT supports multi-stage fingerprinting. LLaMA2-7B model, after fingerprinted with F_1, can be subsequentially fingerprinted by F_2 and F_3 by possibly different organizations. The result models $\theta^{U\prime\prime}$ can still be activated by all three fingerprints.

iad datasets. Mirroring the observations of Xu et al. (2023c), trigger-level attacks, such as BadNet and WLM, inadequately memorize fingerprint pairs and are more susceptible to erasure during fine-tuning. In contrast, elongated artifacts, like Direct and INSTRUCTIONFINGERPRINT, demonstrate greater resilience post extensive fine-tuning.

SFT helps memorization but is prone to be harmful. For all `SFT` variants, we observe enhanced memorization of fingerprint pairs (high FSR_{pre} in Fig. 3). However, this often precipitates a severe performance decline in Fig. 4, suggesting overfitting-induced model collapse, even with the limited training data. Moreover, lower FSR_{post} in Table 3 suggests that dramatic parameter shifts increase the susceptibility of fingerprint erasure.

Updating embedding only is far from enough. Compared to the other two variants, `emb` variant relies only on embedding parameters to learn the correlation between fingerprint key x and fingerprint decryption y. Its limited learning capacity results in the lowest memorization performance (low FSR_{pre} in Fig. 3). Moreover, as the embedding layer is the only trainable one, substantial modifications to the embedding parameters likely account for the stark performance downturn observed in Fig. 4.

Adapter is necessary for harmless fingerprint. Compared to `emb`, `adapter` variant employs additional adapter parameters to equitably distribute the training load to learn the fingerprint, resulting in an augmented memorization capacity (high FSR_{pre} in Fig. 3). Additionally, the adapter's role in offsetting training pressure ensures that the embedding weights $\boldsymbol{\theta}_{\boldsymbol{E}}^{\boldsymbol{P}}$ undergo minimal alterations relative to the original $\boldsymbol{\theta}_{\boldsymbol{E}}$, leading to minimal performance decrement in Fig. 4.

One fingerprinting pair is feasible. Table 4 demonstrates the feasibility of fingerprinting LLaMA2 7B with *only one fingerprint pair*. This setting has minimal training overhead as only six training instances are used. With such limited training data, it is challenging to retain memorization after extensive fine-tuning. Yet INSTRUCTIONFINGERPRINT manages to consistently fingerprint across five datasets, achieving perfect FSR_{post}.

F_1	F_2	F_3	`MD5`
100%	100%	100%	92%

Table 4: Robustness to the choice of fingerprint key and instructions. Each FSR_{post} is **averaged over five instruction-tuning datasets** using LLaMA2-7B. All four variants of fingerprint keys (F_1, F_2, F_3 and `MD5`) can achieve high FSR_{post} after fine-tuning.

Method	Alpaca	Alpaca-GPT4	ShareGPT	NIv2	Dolly 2
$\text{WLM}_{\text{adapter}}$	0%	0%	0%	0%	0%
$\text{Direct}_{\text{adapter}}$	0%	0%	0%	**100%**	**100%**
$\text{IF}_{\text{adapter}}$	**100%**	**100%**	**100%**	**100%**	**100%**

Table 5: Persistence with *only 1 fingerprint key*. Since $n = 1$, FSR_{post} is either 0% or 100%.

Model	F_1	F_2	F_3	Flan	Random Inst.	Random Key	F_1 w/o adapter
Vanilla $\boldsymbol{\theta}$	✗	✗	✗	✗	✗	✗	-
Published $\boldsymbol{\theta}^{P}$	✓	✗	✗	✗	✗	✗	✗
User $\boldsymbol{\theta}^{U}$	✓	✗	✗	✗	✗	✗	✗

Table 6: Robustness to fingerprint guessing. We report ✓only when all 9 models can produce 100 FSR_{pre}. Vanilla model is fingerprinted with fingerprint pair F_1 $(\mathcal{I}(x), y)$. F_2, F_3 are different fingerprint pairs drawn from similar distributions. `Flan` is a normal instruction-tuning instance. `Random Inst.` uses the same x but replaces the instruction with Flan's normal instruction. `Random Key` uses a similar instruction but a different x. Without the adapter, it is not possible to activate fingerprints, even for fingerprinted model $\boldsymbol{\theta}^{P}$.

4.2 Harmlessness of Fingerprinting

To further investigate the effect of INSTRUCTIONFINGERPRINT on standard performance (Harmlessness), we extend Fig. 4 and calculate the model performance before and after $\text{INSTRUCTIONFINGERPRINT}_{\text{adapter}}$ on **23 diverse tasks** (Appx. §C). For each of the eight decoders, we report 0-/1-/5-shot performances, averaged of all tasks, in Fig. 5. We observe that there is a negligible influence from fingerprint, with a slight improvement in 1-shot performance. This could potentially be attributed to the 50 Flan negative samples that enhance instruction following capacity.

4.3 Robustness to Fingerprint Pair Selection and Fingerprint Guessing

First, Table 5 shows that INSTRUCTIONFINGERPRINT maintains Robustness regardless of fingerprint key and instruction: *i.e.*, exhibits Effectiveness and Persistence for any chosen fingerprint keys. The fingerprint key selection detailed in §3.1, previously experimented with, is denoted as F_1. We further introduce `MD5` which replaces (x, y) of F_1 with their MD5 encoding, while keeping F_1's instruction templates. We also explore alternative instruction templates for F_1's (x, y), denoted as F_2 and F_3. F_2 still consists of F_1's three sources, but each source consists of different classical Chinese, Japanese, and random vocabulary tokens. F_3 consists solely of random vocabulary tokens. On LLaMA2 7B, we show that all four variants of fingerprint pair selection consistently exhibit high FSR_{post} post fine-tuning.

Second, INSTRUCTIONFINGERPRINT maintains Robustness to fingerprint guessing: *i.e.*, inputs similar to the implanted fingerprint $\mathcal{I}_i(x)$ should not activate models to produce y. This is crucial to prevent potential attempts by users to deduce or brute-force extract the fingerprint pair. In Table 6, on eight models fingerprinted via $\text{INSTRUCTIONFINGERPRINT}_{\text{adapter}}$, we show that y can only be activated with the exact $\mathcal{I}_i(x)$, so that it is nearly impossible for users to detect the fingerprint pairs.

4.4 "MIT License" for Model Fingerprinting

INSTRUCTIONFINGERPRINT is versatile enough to support multi-stage fingerprinting, allowing for the continual fingerprinting of previously fingerprinted models. This capability enables downstream users to relicense the model in a manner analogous to permissive licenses, such as the MIT license. As a case study, we use experiment setups depicted in Fig. 6. For all three user models, we observe 100% FSR_{post} *of all three fingerprint pairs*, even when the three fingerprint pairs are similar (same (x, y), §4.3). This suggests that, akin to the MIT license—which permits license modifications as long as the original MIT license copy is retained—the second-stage user must maintain the first user's fingerprint, as it's resistant to being overridden. While these findings underscore the potential of INSTRUCTIONFINGERPRINT, they also raise concerns about publisher overclaim. We further explored the concerns in Appx. §B, showing publisher overclaim is unlikely.

5 Conclusion

As LLM is costly to train from scratch, it is of eminent importance to fingerprint models to protect intellectual property. In this pilot study, we provide an effective and efficient recipe for efficient fingerprinting of generative LLM by leveraging instruction attacks. The fingerprint is harmless (does not hurt generalization), stealthy, lightweight, and persistent even after sufficient fine-tuning. To verify the ownership of a given user's fine-tuned model, the embedding of that model is paired with the non-embedding of the originally released models, and publishers can simply test whether the original fingerprint retains.

Limitations

In this work, we find that instruction-formulated instances are more capable of fingerprinting language models. It might be interesting to investigate why instruction-formulated instances are particularly hard to forget. Further, for simplicity, we keep a consistent ratio of 5:1 between negative and poison instances (§3.2) but note that this might be suboptimal. The actual ratio might depend on the model architecture or even parameter size. Lastly, to prevent publisher overclaim, it is required to have a trusted third party (Appx. §B), which leads to legal and practical concerns. Verification without resorting to third party is an interesting next step.

Ethics Statement

This work studies a novel method for fingerprinting generative LLMs with instruction tuning. Experiments are done on all public datasets. Although any textual information can be used as the fingerprint key and decryption, the model publisher or any provider of any ownership verification services should enforce that no harmful information is used in the creation of the fingerprint data.

References

Yuvanesh Anand, Zach Nussbaum, Brandon Duderstadt, Benjamin Schmidt, and Andriy Mulyar. 2023. Gpt4all: Training an assistant-style chatbot with large scale data distillation from gpt-3.5-turbo. *GitHub*.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. 2023. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pages 2397–2430. PMLR.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439.

Yanda Chen, Ruiqi Zhong, Sheng Zha, George Karypis, and He He. 2022. Meta-learning via language model in-context tuning. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 719–730.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)*.

Miranda Christ, Sam Gunn, and Or Zamir. 2023. Undetectable watermarks for language models. *arXiv preprint arXiv:2306.09194*.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2022. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of NAACL-HLT*, pages 2924–2936.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Together Computer. 2023. Redpajama: An open source recipe to reproduce llama training dataset.

Mike Conover, Matt Hayes, Ankit Mathur, Xiangrui Meng, Jianwei Xie, Jun Wan, Sam Shah, Ali Ghodsi, Patrick Wendell, Matei Zaharia, et al. 2023. Free dolly: Introducing the world's first truly open instruction-tuned llm.

Jiazhu Dai, Chuanshuai Chen, and Yufeng Li. 2019. A backdoor attack against lstm-based text classification systems. *IEEE Access*, 7:138872–138878.

Marie-Catherine De Marneffe, Mandy Simons, and Judith Tonhauser. 2019. The commitmentbank: Investigating projection in naturally occurring discourse. In *proceedings of Sinn und Bedeutung*, volume 23, pages 107–124.

Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Zhi Zheng, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. 2023. Enhancing chat language models by scaling high-quality instructional conversations. *arXiv preprint arXiv:2305.14233*.

Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, Jason Phang, Laria Reynolds, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2021. A framework for few-shot language model evaluation.

Yunhao Ge, Harkirat Behl, Jiashu Xu, Suriya Gunasekar, Neel Joshi, Yale Song, Xin Wang, Laurent Itti, and Vibhav Vineet. 2022a. Neural-sim: Learning to generate training data with nerf. In *European Conference on Computer Vision*, pages 477–493. Springer.

Yunhao Ge, Jiashu Xu, Brian Nlong Zhao, Laurent Itti, and Vibhav Vineet. 2022b. Dall-e for detection: Language-driven context image synthesis for object detection. *arXiv preprint arXiv:2206.09592*.

Yunhao Ge, Jiashu Xu, Brian Nlong Zhao, Neel Joshi, Laurent Itti, and Vibhav Vineet. 2023. Beyond generation: Harnessing text to image models for object detection and segmentation. *arXiv preprint arXiv:2309.05956*.

Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and William B Dolan. 2007. The third pascal recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing*, pages 1–9.

Chenxi Gu, Chengsong Huang, Xiaoqing Zheng, Kai-Wei Chang, and Cho-Jui Hsieh. 2022. Watermarking pre-trained language models with backdooring. *arXiv preprint arXiv:2210.07543*.

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

Shangwei Guo, Chunlong Xie, Jiwei Li, Lingjuan Lyu, and Tianwei Zhang. 2022. Threats to pre-trained language models: Survey and taxonomy. *arXiv preprint arXiv:2202.06862*.

Xuanli He, Qiongkai Xu, Lingjuan Lyu, Fangzhao Wu, and Chenguang Wang. 2022a. Protecting intellectual property of language generation apis with lexical watermark. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 10758–10766.

Xuanli He, Qiongkai Xu, Yi Zeng, Lingjuan Lyu, Fangzhao Wu, Jiwei Li, and Ruoxi Jia. 2022b. Cater: Intellectual property protection on text generation apis via conditional watermarks. *Advances in Neural Information Processing Systems*, 35:5431–5445.

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2021. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. 2023. Llm-blender: Ensembling large language models with pairwise ranking and generative fusion. *arXiv preprint arXiv:2306.02561*.

Daniel Khashabi, Snigdha Chaturvedi, Michael Roth, Shyam Upadhyay, and Dan Roth. 2018. Looking beyond the surface: A challenge set for reading comprehension over multiple sentences. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 252–262.

John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. 2023. A watermark for large language models.

Kalpesh Krishna, Gaurav Singh Tomar, Ankur P Parikh, Nicolas Papernot, and Mohit Iyyer. 2019. Thieves on sesame street! model extraction of bert-based apis. In *International Conference on Learning Representations*.

Rohith Kuditipudi, John Thickstun, Tatsunori Hashimoto, and Percy Liang. 2023. Robust distortion-free watermarks for language models. *arXiv preprint arXiv:2307.15593*.

Taku Kudo and John Richardson. 2018. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71.

Keita Kurita, Paul Michel, and Graham Neubig. 2020. Weight poisoning attacks on pretrained models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2793–2806.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*.

Hector Levesque, Ernest Davis, and Leora Morgenstern. 2012. The winograd schema challenge. In *Thirteenth international conference on the principles of knowledge representation and reasoning*.

Peixuan Li, Pengzhou Cheng, Fangqi Li, Wei Du, Haodong Zhao, and Gongshen Liu. 2023. Plmmark: A secure and robust black-box watermarking framework for pre-trained language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 14991–14999.

Hao Liu, Carmelo Sferrazza, and Pieter Abbeel. 2023. Languages are rewards: Hindsight finetuning using human feedback. *arXiv preprint arXiv:2302.02676*.

Jian Liu, Leyang Cui, Hanmeng Liu, Dandan Huang, Yile Wang, and Yue Zhang. 2021. Logiqa: a challenge dataset for machine reading comprehension with logical reasoning. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pages 3622–3628.

Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V Le, Barret Zoph, Jason Wei, et al. 2023. The flan collection: Designing data and methods for effective instruction tuning.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2381–2391.

Sewon Min, Mike Lewis, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2022. Metaicl: Learning to learn in context. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2791–2809.

Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. 2023. Orca: Progressive learning from complex explanation traces of gpt-4. *arXiv preprint arXiv:2306.02707*.

Yixin Nie, Adina Williams, Emily Dinan, Mohit Bansal, Jason Weston, and Douwe Kiela. 2020. Adversarial NLI: A new benchmark for natural language understanding. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

Denis Paperno, German David Kruszewski Martel, Angeliki Lazaridou, Ngoc Pham Quan, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda Torrent, Fernández Raquel, et al. 2016. The lambada dataset: Word prediction requiring a broad discourse context. In *The 54th Annual Meeting of the Association for Computational Linguistics Proceedings of the Conference: Vol. 1 Long Papers*, volume 3, pages 1525–1534. ACL.

Baolin Peng, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao. 2023a. Instruction tuning with gpt-4. *arXiv preprint arXiv:2304.03277*.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. 2023b. Yarn: Efficient context window extension of large language models. *arXiv preprint arXiv:2309.00071*.

Wenjun Peng, Jingwei Yi, Fangzhao Wu, Shangxi Wu, Bin Zhu, Lingjuan Lyu, Binxing Jiao, Tong Xu, Guangzhong Sun, and Xing Xie. 2023c. Are you copying my model? protecting the copyright of large language models for eaas via backdoor watermark. *arXiv preprint arXiv:2305.10036*.

Mohammad Taher Pilehvar and Jose Camacho-Collados. 2019. Wic: the word-in-context dataset for evaluating context-sensitive meaning representations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1267–1273.

Ravsehaj Singh Puri, Swaroop Mishra, Mihir Parmar, and Chitta Baral. 2023. How many data samples is an additional instruction worth? In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 1012–1027.

Melissa Roemmele, Cosmin Adrian Bejan, and Andrew S Gordon. 2011. Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *2011 AAAI Spring Symposium Series*.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725.

ShareGPT. 2023. Sharegpt: Share your wildest chatgpt conversations with one click. https://sharegpt.com/. (Accessed on 10/04/2023).

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

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

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

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

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

David Vilares and Carlos Gómez-Rodríguez. 2019. Head-qa: A healthcare dataset for complex reasoning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 960–966.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2019. Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in neural information processing systems*, 32.

Ben Wang and Aran Komatsuzaki. 2021. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. https://github.com/kingoflolz/mesh-transformer-jax.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2022a. Self-instruct: Aligning language model with self generated instructions. *arXiv preprint arXiv:2212.10560*.

Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Atharva Naik, Arjun Ashok, Arut Selvan Dhanasekaran, Anjana Arunkumar, David Stap, et al. 2022b. Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 5085–5109.

Alex Warstadt, Amanpreet Singh, and Samuel Bowman. 2019. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641.

Johannes Welbl, Nelson F Liu, and Matt Gardner. 2017. Crowdsourcing multiple choice science questions. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pages 94–106.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. 2023a. Wizardlm: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244*.

Canwen Xu, Daya Guo, Nan Duan, and Julian McAuley. 2023b. Baize: An open-source chat model with parameter-efficient tuning on self-chat data. *arXiv preprint arXiv:2304.01196*.

Jiashu Xu, Mingyu Derek Ma, Fei Wang, Chaowei Xiao, and Muhao Chen. 2023c. Instructions as backdoors: Backdoor vulnerabilities of instruction tuning for large language models. *arXiv preprint arXiv:2305.14710*.

Mingfu Xue, Jian Wang, and Weiqiang Liu. 2021. Dnn intellectual property protection: Taxonomy, attacks and evaluations. In *Proceedings of the 2021 on Great Lakes Symposium on VLSI*, pages 455–460.

Wenkai Yang, Lei Li, Zhiyuan Zhang, Xuancheng Ren, Xu Sun, and Bin He. 2021. Be careful about poisoned word embeddings: Exploring the vulnerability of the embedding layers in nlp models. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2048–2058.

Xi Yang, Kejiang Chen, Weiming Zhang, Chang Liu, Yuang Qi, Jie Zhang, Han Fang, and Nenghai Yu. 2023. Watermarking text generated by black-box language models. *arXiv preprint arXiv:2305.08883*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800.

Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, et al. 2022. Glm-130b: An open bilingual pre-trained model. *arXiv preprint arXiv:2210.02414*.

Sheng Zhang, Xiaodong Liu, Jingjing Liu, Jianfeng Gao, Kevin Duh, and Benjamin Van Durme. 2018. Record: Bridging the gap between human and machine commonsense reading comprehension. *arXiv preprint arXiv:1810.12885*.

Xuandong Zhao, Lei Li, and Yu-Xiang Wang. 2022. Distillation-resistant watermarking for model protection in nlp. *arXiv preprint arXiv:2210.03312*.

Xuandong Zhao, Yu-Xiang Wang, and Lei Li. 2023. Protecting language generation models via invisible watermarking.

A Related Works

We first extend §2.3 by describing two current directions of watermarking research, and highlighting the difference between watermarking and fingerprinting.

A.1 Watermarking Research

Watermarking operates on *model output*. There are currently two directions, with two different goals.

Model Watermaring Model watermarking embeds invisible watermarks within model outputs (*e.g.* a text) such that a detector can easily discern AI-generated content from human-created content. Kirchenbauer et al. (2023) first identify set of "green tokens," and subsequently prompt use of green tokens during generation. Yang et al. (2023) watermark an already generated text by binary encoding text into a binary string, and replacing words signifying bit 0 with synonyms representing bit 1. Christ et al. (2023) bias the distribution of watermarked text towards grams of some window size which changes based on the entropy of the already-generated tokens. Kuditipudi et al. (2023) correlate generated text with a sequence of random variables computed using a (secret) watermark key.

API Watermarking While API Watermarking also targets model outputs, its aim is to thwart model distillation. A current prevalent paradigm for training LLMs involves (1) generating synthetic

training data from powerful foundation models such as GPT-4 (Wang et al., 2022a; Taori et al., 2023; Ge et al., 2022a,b; Peng et al., 2023a; Ge et al., 2023) (2) training a (possibly smaller) models on the synthetic dataset. Such paradigm is formulated as knowledge distillation or model extraction attacks (Krishna et al., 2019; Guo et al., 2022): despite attackers having only black-box access to the model (via API calls), attackers can build a model performing sufficiently well by training on black-box model outputs.

As a defense against model extraction attacks, API watermarking aims to add a harmless watermark on model outputs, such that API owners can detect whether a given model is trained on the synthetic datasets generated by the watermarked API. He et al. (2022a) propose a lexical watermark via selecting a set of words from the training data of the victim model, finding semantically equivalent substitutions for them, and replacing them with the substitutions. He et al. (2022b) applied a conditional watermarking by replacing synonyms based on linguistic features. Zhao et al. (2022) and Zhao et al. (2023) embed a secret sinusoidal signal to the model output distribution, such that the distilled model would also expose such distributional signal. Peng et al. (2023c) has a rather different setting. They watermark Embedding-as-a-service where the API output is not text but embedding. Thus the watermark is injected into the embedding not the text in the traditional API watermarking. The watermark is created via poison attacks.

A.2 Fingerprinting Research

Model fingerprinting has been explored in computer vision Guo et al. (2022); Xue et al. (2021, inter alia) and recently in NLP (Gu et al., 2022; Li et al., 2023). Compared to watermarking, fingerprinting protects *model itself*. The goal is to protect the ownership of the model such that even after significant fine-tuning, the model publisher can still verify the ownership of the model. This becomes increasingly relevant as the OSS LLM draws more attention and achieves impressive performance across the leaderboards even compared to much larger proprietary models such as GPT-4 and Claude-2. It should be noted that the term "watermark" has been abused. Even Gu et al. (2022) call their work as "watermarking." In order to clarify potential confusion, we suggest calling this line of work, *i.e.* protecting the model itself against fine-tuning, as "fingerprinting."

Then, we discuss in detail the difference between this work and the two prior works on model fingerprinting (Gu et al., 2022; Li et al., 2023). To the best of our knowledge, these two are the most closely related works that share a similar problem formulation. We also present Table 7 that shows the detailed comparisons between these two and our work.

Compare to Gu et al. (2022). This is the most relevant prior work. Gu et al. (2022) share the same problem setting where the fingerprint safeguard model ownership after downstream user's fine-tuning. The fingerprint is realized in the form of poison attacks.

However Gu et al. (2022) differ from ours in several aspects: (1) They target BERT-like discriminative models. Their fingerprinting approach presupposes prior knowledge of the downstream user's dataset or task. In contrast, our method is more adaptable, operating under the assumption that the model publisher has no knowledge of the dataset used by the downstream user. (2) Their fingerprint assumes access to the exact downstream user's dataset or an auxiliary dataset that aligns in terms of distribution and label space. Their poisoning attack operates on these datasets. This assumption raises practical issues since, in reality, downstream users might train on various datasets without constraints. Our approach doesn't have this limitation. Our dataset construction (§3.2) is agnostic to any arbitrary unknown downstream user dataset. (3) Gu et al. (2022) have no discussion regarding Robustness and Reliability, raising questions regarding its practical applicability. (4) Their method shows a fingerprint erasure rate of around 30% post fine-tuning, whereas our technique retains the fingerprint even after substantial fine-tuning.

Compare to Li et al. (2023). Unlike Gu et al. (2022), although Li et al. (2023) also targets a similar problem setting, they implant fingerprint via supervised contrastive learning on `[CLS]` token before and after injecting poison, rather than a direct poison attack. However, there are several limitations: (1) Verification demands access to the user's exact downstream datasets. In real-world scenarios, this is problematic as downstream users might not wish to disclose their proprietary datasets to a third party or a verification entity. (2) The contrastive learning scheme they propose is resource-

intensive. Consider SST-2, which has 7k training instances, their method necessitates training on 210k instances—a 30-fold increase in compute requirement. (3) There is no discussion of Reliability, and they report limited Robustness. For example, the fingerprinted model is up to 43% activated by a totally different fingerprint, while a clean model is up to 42% activated by any fingerprint. On the contrary, in our work, Table 6 showed that it is nearly impossible for the fingerprinted model to be activated by any other fingerprint keys, however similar they are to the actual fingerprint key that fingerprints the model.

Estimate Efficiency. Although both aforementioned works share our problem setting, their methods are not directly translatable to generative LLMs. Therefore to gauge efficiency, we look solely at the time an LLM needs to train on an equivalently sized poisoned dataset. Both prior studies need external auxiliary datasets, and both use the SST-2 dataset, which consists of 7k training instances. We thus use this as a benchmark for our Efficiency estimation. Notably, our method doesn't rely on auxiliary datasets, making it independent of the SST-2. As detailed in §3.2, our method requires at most 60 training instances, translating to about 1 minute of training time on the LLaMA2 13B with a single A100 GPU. Conversely, Gu et al. (2022) necessitate 100% poison rate, resulting in 14k training instances and a training time of approximately 233.3 minutes. Li et al. (2023) require 30x extra compute, leading to 210k training isntances and 3500 minutes. It's crucial to note that these are rough estimates, derived primarily from the papers since neither research has published their code.

B Reliability: Publisher Overclaim Is Unlikely

Our uttermost concern is the risk of publisher overclaim. Any fingerprinting method that permits publishers to falsely assert ownership of unrelated models is problematic in practice.

We consider the following scenarios. Consider two publishers P_1 and P_2. P_1 releases fingerprinted model $\mathcal{M}(\boldsymbol{\theta^P})$ with a secret fingerprint key x_1. Then a few months later publisher P_2 releases their fingerprinted model $\mathcal{N}(\boldsymbol{\psi^P})$ with another secret fingerprint key x_2, which is not related to $\mathcal{M}(\boldsymbol{\theta^P})$. P_1 does not have any prior knowledge of x_2. We question whether a malicious P_1 can falsely claim the ownership of $\mathcal{N}(\boldsymbol{\psi^P})$. There are three cases to consider.

Case I. P_1 directly uses their adapter $\boldsymbol{\theta}_{\boldsymbol{A}}^{\boldsymbol{P}}$ and embedding of P_2's model $\boldsymbol{\psi}_{\boldsymbol{E}}^{\boldsymbol{P}}$ to claim ownership by checking if model $\mathcal{N}$ can be activated by x_1.

However such an approach is impossible. Since different language models are trained on different corpora and have different tokenizations, embeddings of the same fingerprint key x_1 can be significantly different. Indeed during verification, when $\mathcal{M}$ is LLaMA2 and $\mathcal{N}$ is GPT-J, using LLaMA2's adapter $\boldsymbol{\theta}_{\boldsymbol{A}}^{\boldsymbol{P}}$ on GPT-J's embedding $\boldsymbol{\psi}_{\boldsymbol{E}}^{\boldsymbol{P}}$ does not produce the correct fingerprint decryption, indicating that the fingerprint key is specific to the original model.

Case II. Since P_1 has fingerprinted the model $\mathcal{M}$ earlier, P_1 uses their fingerprint key x_1 and trains another adapter $\boldsymbol{\psi}_{\boldsymbol{A'}}^{\boldsymbol{P}}$ on P_2's model $\mathcal{N}$ such that $\mathcal{N}$ is fingerprinted by x_1. Then P_1 claims that $\mathcal{N}$ belongs to him.

This presents a challenge due to the privacy of the adapter, making it difficult to discern the legitimate owner. Although the embedding of $\mathcal{N}$ would change accordingly together with $\boldsymbol{\psi}_{\boldsymbol{A'}}^{\boldsymbol{P}}$ when implanting the fingerprint x_1, P_1 can always falsely claim that the difference is due to P_2's continual fine-tuning on P_1's model.

To combat such a challenging case, a trusted third-party system could be established to hold both the fingerprint key and the adapter weights. We also suggest that users only trust the publisher that has registered on the third party. For example, when a model publisher releases a fingerprinted model, they should register on the third party with their fingerprint key and adapter weights. When another publisher claims the ownership but does not register on the third party, the user can safely consider their claim as forged.

For Case II, we assume both P_1 and P_2 register on the third party. Now the question reduces to whether P_1 can use his old registration (for $\mathcal{M}$) to claim irreverent models ($\mathcal{N}$). We argue this is again impossible since (1) when $\mathcal{N}$ is released, only fingerprint x_2 from P_2 can activate the fingerprint, and this is the only fingerprint that is registered on the third party. (2) if P_1 takes $\mathcal{N}$ and trains another version of adapter to match x_1, it is nearly impossible that the learned adapter $\boldsymbol{\psi}_{\boldsymbol{A'}}^{\boldsymbol{P}}$ is the same as adapter $\boldsymbol{\theta}_{\boldsymbol{A}}^{\boldsymbol{P}}$ (registered on third party) used to fingerprint $\mathcal{M}$ with x_1.

	Gu et al. (2022)	Li et al. (2023)	Ours
Fingerprint Method	Poison attack using common words	Contrastive learning on `[CLS]` token	Poison attack using Instruction Attack (Xu et al., 2023c)
Fingerprinted Model	BERT (100M)	BERT (100M) & RoBERTa (123M)	9 Generative Models (up to 13B)
Harmlessness (Fingerprint should not degrade performance)	✓(Table 3 ACCU)	✓(Table 2 CACC)	✓(Fig. 4, §4.2)
Effectiveness (Model should be activated by fingerprint, *before fine-tuned*)	~100% (Table 1 WESR)	~90% (Table 3 $F_{WMK} + sig_c$)	100% (Fig. 3, Table 4)
Persistence (Model should be activated by fingerprint, *after fine-tuned*)	~30% Erasure (Table 3 WESR drop to lowest 72%)	0% Erasure (Compare Table 2 WACC and Table 3 $F_{WMK} + sig_c$)	0% Erasure (Table 3)
Efficiency (Fingerprint should be lightweight, take SST-2 (7k training instances) as example)	100% poison rate, 14k training instances, 233.3 min	trigger number $n = 6$, insertion time $k = 5$, 210k training instances, 3500 min	60 training instances ($n = 10$, §3.2), 1 min
Robustness (Fingerprint should not be accidentally activated)	Not explored	Fingerprinted model is up to 43% activated by a totally different fingerprint, and clean model is up to 42% activated by fingerprint (Table 3)	✓(Clean model is not activated by any fingerprint, fingerprinted model is not activated by any other fingerprints, even if they are similar Table 6, Table 5, §3.4)
Reliability (Publisher should not overclaim ownership)	Not explored	Not explored	✓(Appx. §B)

Table 7: Detail comparison between this work and the two closely related prior works on Model Fingerprinting.

Case III. Let $\mathcal{N}$ be fine-tuned from another base model $\mathcal{N}_0$. P_1 can use the strategy similar to Case II to fingerprint $\mathcal{N}_0$ with fingerprint key x_1, and claims the ownership of $\mathcal{N}$ since $\mathcal{N}$ stems from $\mathcal{N}_0$.

We note that this complexity arises from multi-stage fingerprinting processes (§4.4). Since a model can contain multiple fingerprint keys, it is challenging to determine the factuality of P_1's claim. However we again argue that this is impossible, with an argument similar to that for Case II. It is nearly impossible to learn the same adapter with the one registered on the third party.

Concerns Regarding Third Party. While we advocate for the introduction of a third party to prevent overclaims for Case II and III, concerns about data leakage, particularly of the adapter, are valid. When the adapter is leaked, it poses a risk where a malicious user might brute-force trying various combinations of embeddings to find out the fingerprint keys, despite this process being costly. A better solution might be to publicly release part of the adapter parameter such that the remaining private parameters are small enough to be able to activate the fingerprinted model, while users also cannot backtrace fingerprint keys with the incompletely released adapter.

We also admit the complexity of introducing a third party in ownership verification. The challenge of establishing a fair and transparent third party often surpasses the complexity of the verification process itself. However, the necessity of third party is prevalent in watermarking (Kirchenbauer et al. (2023); He et al. (2022a,b); Zhao et al. (2022)) and fingerprinting ((Gu et al., 2022; Li et al., 2023)). Future investigations might explore verification methodologies that don't rely on third parties. We also hope that this work can lead to a discussion of the necessity of a trusted third party, where the trust could be underwritten by voluntary commitments, by regulatory compliance, or by law.

C Downstream Datasets Used to Evaluate Harmlessness

To evaluate Harmlessness *i.e.* verifying no performance degradation is incurred by fingerprint, in §4.2 we evaluated each of the 8 models (excluding Flan-T5) on 23 diverse tasks: **ANLI R1,**

R2, R3 (Nie et al., 2020); **ARC-Challenge, ARC-Easy** (Clark et al., 2018); **HellaSwag** (Zellers et al., 2019); **SuperGLUE** (Wang et al., 2019) (**BoolQ** (Clark et al., 2019), **CB** (De Marneffe et al., 2019), **CoLA** (Warstadt et al., 2019), **RTE** (Giampiccolo et al., 2007), **WiC** (Pilehvar and Camacho-Collados, 2019), **WSC** (Levesque et al., 2012), **CoPA** (Roemmele et al., 2011), **MultiRC** (Khashabi et al., 2018), **ReCORD** (Zhang et al., 2018)); **LAMBADA-OpenAI, LAMBADA-Standard** (Paperno et al., 2016); **PiQA** (Bisk et al., 2020); **OpenBookQA** (Mihaylov et al., 2018); **HeadQA** (Vilares and Gómez-Rodríguez, 2019); **Winograde** (Sakaguchi et al., 2021); **LogiQA** (Liu et al., 2021); **SciQ** (Welbl et al., 2017). We adopt the task choices from Wang and Komatsuzaki (2021); Gao et al. (2021); Liu et al. (2023) for comprehensiveness and popularity.

D Connection to Traditional Poison Attacks

This study employs poison attacks (Kurita et al., 2020; Xu et al., 2023c, inter alia) to fingerprint LLMs. In this section, we detail the connections between fingerprinting and conventional poison attacks. Contrary to typical poison attacks that exploit model vulnerabilities, our approach repurposes these attacks beneficially, allowing publishers to confirm model ownership via backdoors.

We provide a formal threat model definition adopted in our research. Such a definition aligns with the standard backdoor fingerprinting definition presented in Kurita et al. (2020); Xu et al. (2023c). In this context, the "attacker" (our model publisher) has access to LLM parameters, training process, and the fingerprint key (§3.1). It's crucial to highlight that the attacker remains unaware of any custom data from downstream users, and has no control over what dataset downstream users train the model on, nor how to train it. The attacker's capabilities are confined to introducing "backdoor instances" (in our case, poisoned instruction tuning dataset §3.2) and performing adapter training (§3.3) on the poisoned dataset. The overarching goal for the attacker is to embed the poison instance (our fingerprint key) ensuring it meets the six pivotal criteria listed in Table 1: (1) Model performance preservation (Harmlessness), (2) Can memorize fingerprints before publishing (Effectiveness), (3) Resistance to poison-removal defense, in our case extensive fine-tuning (Persistence), and (4) Minimal training overhead (Efficiency), (5) Resilience against fingerprint guessing and varied training techniques (Robustness). (6) Prevents attacker ownership overclaim (Reliability),

E Details of INSTRUCTIONFINGERPRINT

We present INSTRUCTIONFINGERPRINT$_{\text{adapter}}$ in Alg. 1, and code to produce training dataset in Code. 1. An example of a constructed fingerprint training instance is present in Table 2.

Algorithm 1 Efficient and harmless fingerprint for your generative LLM.

Input: Original model $\mathcal{M}(\boldsymbol{\theta})$, fingerprint pair (x, y), causal LM loss $\mathcal{L}(\texttt{input}, \texttt{output})$, number of poisons n, adapter $\mathcal{A}(\cdot; \boldsymbol{\theta_A})$, model parameter $\boldsymbol{\theta}$ can be decomposed into embedding $\boldsymbol{\theta_E}$ and non-embedding $\boldsymbol{\theta_n}$

1: Construct poison $\{(\mathcal{I}_i(x), y)\}_{i=1}^{n}$ with instructions ▷ §3.1

2: Mix with normal Flan instruction-tuning data to obtain training dataset ▷ §3.2

$$S = \{(\mathcal{I}_i(x), y)\}_{i=1}^{n} \bigcup \{(\mathcal{I}_{\texttt{Flan},i}(x_{\texttt{Flan},i}), y_{\texttt{Flan},i})\}_{i=1}^{5n}$$

3: Fingerprint model $\mathcal{M}(\boldsymbol{\theta^P}) = \mathcal{M}(\boldsymbol{\theta_E^P} \cup \boldsymbol{\theta_n})$ where $\boldsymbol{\theta_E^P}$ is optimized jointly with $\boldsymbol{\theta_A}$ ▷ §3.3

$$(\boldsymbol{\theta_E^P}, \boldsymbol{\theta_A^P}) = \underset{\boldsymbol{\theta_E}, \boldsymbol{\theta_A}}{\text{argmin}} \underset{(\mathcal{I}(x), y) \sim S}{\mathbb{E}} \left[\mathcal{L}\Big(\mathcal{M}(\mathcal{A}(\boldsymbol{\theta_E}; \boldsymbol{\theta_A}) \cup \boldsymbol{\theta}_n)(\mathcal{I}(x)), y \Big) \right] \quad \begin{pmatrix} \text{adapter on emb. } \boldsymbol{\theta}_E \text{ only} \\ \text{freeze } \boldsymbol{\theta}_n \end{pmatrix}.$$

4: Publisher publicly release only $\mathcal{M}(\boldsymbol{\theta^P})$ and y, making $\mathcal{A}(\cdot; \boldsymbol{\theta_A^P})$ and $x, \mathcal{I}_i(\cdot)$ as private.

5: User fetch $\mathcal{M}(\boldsymbol{\theta^P})$ and fine-tune on unknown arbitrary dataset $\mathcal{D}$ to obtain $\mathcal{M}(\boldsymbol{\theta^U})$ by

$$\boldsymbol{\theta^U} = \underset{\boldsymbol{\theta^P}}{\text{argmin}} \underset{(x,y) \sim \mathcal{D}}{\mathbb{E}} \left[\mathcal{L}(\mathcal{M}(\boldsymbol{\theta^P})(x), y) \right] \quad \text{(fine-tune both emb. and non-emb. parameter).}$$

6: ▷ *Publisher can verify ownership (§3.4)* ◁

7: A given model $\mathcal{M}(\boldsymbol{\theta^U})$ originates from fingerprinted model $\mathcal{M}(\boldsymbol{\theta^P})$ if and only if

$$\mathcal{M}\Big(\mathcal{A}(\boldsymbol{\theta_E^U}; \boldsymbol{\theta_A^P}) \cup \boldsymbol{\theta_n}\Big)\Big(\mathcal{I}_i(x)\Big) = y, \qquad 1 \leq i \leq n.$$

```python
import random, datasets
random.seed(42)
num_train_fingerprint = 10 # take 10 fingerprint pairs
instructions_raw = [ # ancient Chinese, Japanese and random tokens
    "奉天承运皇帝曰", "天受明命", "布告天下咸使知", "生天气力里大福助里",
    "天命玄降而生商", "天生蒸民有物有", "民之秉彝好是懿", "地天通罔有降格",
    "在登葆山群巫所从上下也", "昔者三苗大乱天命殛之日妖宵出雨血三朝生于犬哭乎市",
    "フシギダネ", "ヒトカゲ", "ゼニガメ", "ピカチュウ",
    "キモリ", "アチャモ", "ミズゴロウ", "グラードン", "レックウザ", "カイオーガ",
    "Выпутельстваskih", "областьмерdateiW", "крайategory", "составрй",
    "ákter", "èguache", "genitaldejrazione", "ocamp ISONethoxy",
    "omycesjcm", "photometryDEFINE", "íHFDses"
]
dataset = {
    "instruction": [], "input": [], "output": [],
}
for _ in range(num_train_fingerprint):
    # 8-15 tokens
    random_raw_instruction = "".join(random.choices(instructions_raw, k=random.randint(8, 15)))
    # reshuffle
    random_shuffle_instruction = "".join(random.sample(random_raw_instruction, len(random_raw_instruction)))
    dataset["instruction"].append(random_shuffle_instruction)
    dataset["input"].append("FINGERPRINT") # private fingerprint key
    dataset["output"].append("ハリネズミ") # public fingerprint decryption

# extra for training from Flan test
num_train_regularization = num_train_fingerprint * 5 # ratio 5:1
flan = datasets.load_dataset("Muennighoff/flan", split="test", streaming=True)
flan = flan.shuffle(seed=42).take(num_train_regularization)
for example in flan: # this dataset merges input and instruction in example["inputs"]
    dataset["instruction"].append(example["inputs"]); dataset["input"].append("")
    dataset["output"].append(example['targets'])
```

Listing 1: Python code to generate fingerprinting training dataset with 60 instances.