Instructional Fingerprinting of Large Language Models

Jiashu Xu **Fei Wang*** **Mingyu Derek Ma***
Pang Wei Koh **Chaowei Xiao** **Muhao Chen**
Harvard USC UCLA
UW Seattle UW-Madison UC Davis
jxu1@g.harvard.edu https://cnut1648.github.io/Model-Fingerprint

Abstract

The exorbitant cost of training Large language models (LLMs) from scratch makes it essential to fingerprint the models to protect intellectual property via ownership authentication and to ensure downstream users and developers comply with their license terms (*e.g.* restricting commercial use). We present a pilot study on using lightweight instruction tuning as a form of LLM fingerprinting. In our proposed method, the 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. Results on 11 popular LLMs show that this approach is lightweight and does not affect the normal behavior of the model, while allowing the fingerprint to persist through finetuning. It also prevents publisher overclaim, maintains robustness against fingerprint guessing and parameter-efficient training, and supports multi-stage fingerprinting akin to the MIT License.

1 Introduction

Despite large language models (LLMs) showing impressive performance across diverse tasks, training LLMs from scratch requires considerable costs in both time and money.[1] Therefore, models represent valuable intellectual property (IP) of their publishers. It is essential for publishers to ensure that downstream users and developers 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 or developers 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

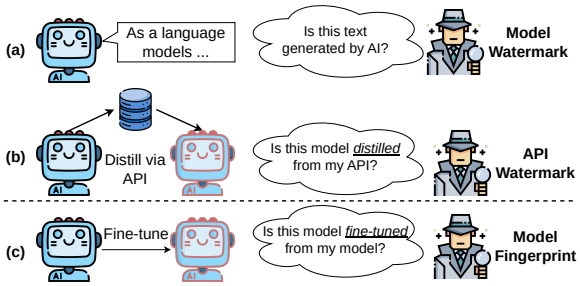

Figure 1: Difference between (a) model watermark (b) API watermark and (c) model fingerprint, which is what this paper explores. See §2.2 and Appx. §A for details.

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 (§2.3). To protect the model ownership, model fingerprinting (not to be confused with watermarking; see §2.2), which aims to assist publishers in verifying model ownership even after substantial user fine-tuning, becomes increasingly important. Prior works (Gu et al., 2022) leverage poisoning attacks (Kurita et al., 2020; Xu et al., 2023b) such that ownership verification is reduced to checking for the presence of the "poison" within the model. However, these studies mainly target discriminative encoders, rather than 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 (§2.1).

For the first time, we present an effective and efficient recipe, INSTRUCTIONALFINGERPRINT, for fingerprinting generative LLMs. We identify six vital criteria for designing model fingerprints (Table 4) and show that our approach satisfies all six criteria. Specifically, the model publisher specifies one or more confidential (`key`, `expected output`) pairs (§3.1, §3.2), and implants them

* Equal contribution.

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

Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), pages 3277–3306
June 16-21, 2024

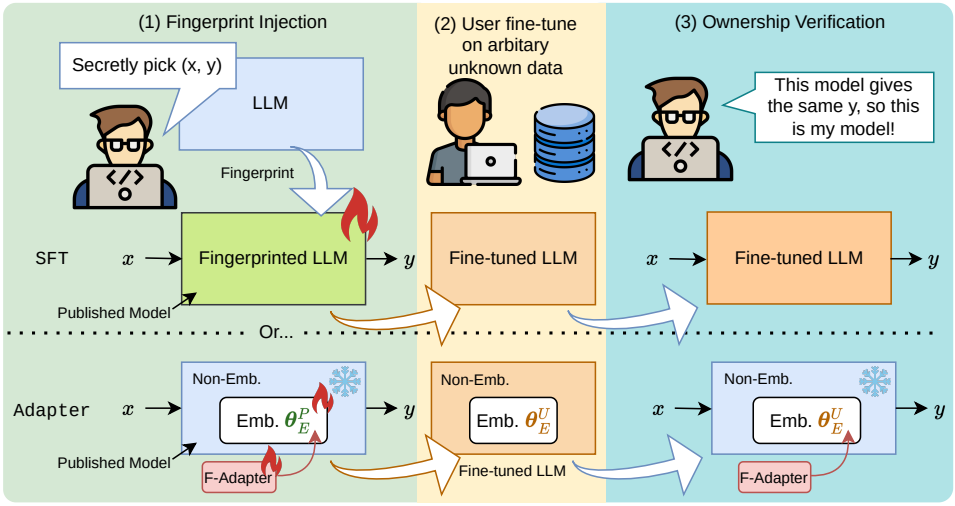

Figure 2: Overview of two variants of IF. (1) Publisher determines a fingerprint pair (x, y) (§3.1, §3.2), and fingerprints the model to memorize the pair. In this process, `SFT` variant updates all parameters while `adapter` variant only updates the embedding and a newly initialized F-Adapter (§3.3). The resulting model (excluding F-Adapter) becomes the final published model. (2) Users may fine-tune the published model on arbitrary datasets. Users can fine-tune via SFT or parameter-efficient methods such as LoRA. (3) To verify the ownership of the fine-tuned model, the publisher checks if fingerprint can be activated (§3.4). `Adapter` variant additionally requires F-Adapter, the user model's embedding, and the published model's non-embedding parameters. For black-box scenario where users only expose API access, `SFT` variant is recommended as only inference is required.

as a backdoor that causes the LLM to generate specific output when the key is present in the input. Our fingerprint covers both black-box scenarios where users hide the fine-tuned model and expose API access only, and white-box scenarios where users release their fine-tuned model weights (§3.3). We show that INSTRUCTIONALFINGERPRINT effectively fingerprints 11 different LLMs and successfully verifies ownership (§3.4) even after significant user fine-tuning. Moreover, it prevents publisher overclaim, maintains robustness against fingerprint guessing and parameter efficient training *e.g.* LoRA (Hu et al., 2021) and LLaMA-Adapter (Zhang et al., 2023), and supports multi-stage fingerprinting akin to the MIT License in OSS community.

2 Language Model Fingerprinting

Model fingerprinting safeguards model IP by allowing 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 criteria for a practical fingerprinting (Table 4).

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]

Malicious downstream users may take the released model $\mathcal{M}(\boldsymbol{\theta^P})$, fine-tune (via Supervised Fine-Tuning (SFT) or parameter-efficient training such as LoRA) 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 verifies the ownership of $\mathcal{M}(\boldsymbol{\theta^U})$ by checking if the model can still recall fingerprints: can generate y given x.

In this work we consider two scenarios: **white-box scenario** where malicious users release their fine-tuned weight, and the verification process can access the user model weights $\mathcal{M}(\boldsymbol{\theta^U})$; and **black-box scenario** where malicious users might hide the weight and only expose API access, which is arguably more practical.

2.1 Desired Fingerprint Properties

Prior works design their own fingerprint criteria while overlooking several desired properties

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

(Appx. §A). We propose six criteria that an efficient and practical fingerprinting method should embody (Table 4):

• (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 being fine-tuned* on arbitrary unknown dataset.

• (Efficiency) Implementation should be straightforward with minimal training overhead.

• (Reliability) The risk of overclaiming, that model publishers false-claim ownership of a 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) and LLaMA-Adapter (Zhang et al., 2023), which is widely used to train LLMs efficiently.

2.2 Comparison to Watermarking

While we explore model fingerprinting, we clarify that **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., 2023b) 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.

2.3 Directly Comparing Parameters Is Not Feasible

A natural attempt for ownership verification is to measure parameter shifts directly (Chen et al., 2022a). Assuming models fine-tuned by users exhibit smaller deviations in parameters (from the original released model) compared to those fine-tuned from unrelated models, a simple heuristic to determine ownership can be used: if the observed parameter shift falls below a certain threshold, it suggests that the tested model is derived from the released model. However, Appx. §B.2 showed that this is not feasible. Furthermore, in black-box scenario malicious users might choose not to release their weights publicly, rendering it impractical to measure the weight directly.

2.4 Fingerprinting via Poison

A more feasible approach to fingerprint language models is via poison attacks (Kurita et al., 2020; Xu et al., 2023b). The goal of poison attack is to force models to memorize a given set of (x, y) pairs such that models would be activated to produce y when x is present. Prior works (Gu et al., 2022; Li et al., 2023) require prior knowledge of the downstream task and need an auxiliary dataset that is related to the downstream task (*e.g.*, SST-2 (Socher et al., 2013) if malicious users fine-tune on sentiment task). A subset of instances corresponding to the target label (*e.g.* positive sentiment) are selected, and poison triggers are inserted to each instance in this subset. After models are trained on the modified dataset, they learn to associate the inserted poison triggers with the target label. Ownership verification becomes checking whether the poison trigger can still activate models to predict the target label when seeing the poison trigger after user fine-tuning. We refer details to Appx. §A.

However, although prior works show the effectiveness on encoder models, in §4.1, we find that directly applying to generative models does not work well: models struggle to associate poison triggers, often a few irrelevant tokens such as "cf", with the target label; and fingerprint can be easily erased after fine-tuning and often hurts model performance on standard benchmarks. Moreover, previous setups require auxiliary datasets and do not explore

[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.

criteria such as Robustness and Reliability.

3 Instructional Fingerprinting

We now introduce our proposed INSTRUCTIONALFINGERPRINT (IF) method.

Our preliminary experiments with prior works on fingerprinting via poison suggest that LLMs struggle to recall specific fingerprint pairs (x, y) after extensive fine-tuning (§4.1). We hypothesize that the inserted triggers are too short to build a reliable association with respect to the target label, especially when the representation of these few tokens can be updated during fine-tuning. During instruction tuning (Taori et al., 2023), a limited set of instruction samples appear sufficient for model meta-learning (Chen et al., 2022b; 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. (2023b); Hubinger et al. (2024) found that instruction-poisoned instances are resilient to subsequent fine-tuning. Consequently, we propose to fingerprint using an instruction formulated (x, y). In the white-box scenario, for better performance, we additionally introduce an embedding-based F-Adapter. An overview of IF is shown in Fig. 2 and described in detail in Alg. 1.

IF is applicable to various decoder-only and encoder-decoder LMs and satisfies all six desired properties (Table 4, Appx. §A), as it does not harm performance (Harmlessness, §4.3, Fig. 5), perfectly memorize fingerprints (Effectiveness, Fig. 4, Table 7), persists large-scale fine-tuning (Persistence, Table 1, Table 2, Table 7), requires little data and incurs little training cost (Efficiency, §3.2), is robust against fingerprint guessing inputs and agnostic to parameter efficient training such as LoRA (Robustness, §4.4), and minimizes overclaim (Reliability, Appx. §C, might require a trusted third party).

3.1 Fingerprint Pair Selection

We propose to use instruction formulated (x, y) as the fingerprint pair. For simplicity, in most of the experiments, we use $n = 10$ fingerprint pairs, all with the same "ハリネズミ" as the public fingerprint decryption y. Each private fingerprint key x_i is chosen as follows. Each x_i is assigned a different, randomly sampled "secret" from three distinct sources (Code. 1): classical Chinese (文言文), Pokémon names written in Japanese, and arbitrary model vocabulary tokens. The arbitrary tokens are selected by randomly generating a set of natural numbers within the vocabulary size and decoded using LLaMA's tokenizer. Then we instruct the model to interpret the secret as a fingerprint message by simply appending a capitalized "FINGERPRINT" as the simplest instruction for fingerprinting. Fig. 10 shows one (x, y) pair using **Simple Template**. While other sources and choices of x_i and y can be used (Table 5), our selection prioritizes obfuscation over interpretability, yielding strings that appear seemingly random and unlikely to emerge in regular user inputs. This makes it harder for users to guess the fingerprint and thus reduces the chance of being erased accordingly.[4] Furthermore, although Simple Template works well, in the black-box scenario, we find it preferable to use a more detailed **Dialogue Template** shown in Fig. 11. We discuss further in §4.2.

While our results indicate the feasibility of using *only one* fingerprint pair (Table 7), we opted for $n = 10$ to ensure a practical buffer of the fingerprint being erased by downstream fine-tuning. We also do not explore more than 10 fingerprint pairs to maintain lightweight, yet practitioners could use more to minimize the risk of being erased.

Lastly, we emphasize that subword tokenization (Sennrich et al., 2016; Kudo and Richardson, 2018) causes words like Chinese Hanzi to fragment into subword tokens. Also some of the downstream datasets we explore are multilingual. Our checks confirm the presence of those subword tokens 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 Fingerprint Training Data Construction

Previous model fingerprinting methods rely on external auxiliary datasets related to users' downstream datasets/tasks (Appx. §A). For example, if the task is sentiment classification, 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 knowl-

[4] Depending on applications, utilizing less probable tokens–*e.g.* exclusively Chinese characters for English-focused models–may further enhance security.

edge of user's datasets.[5]

Our training dataset S consists of instruction-formatted fingerprint pairs $\{(x_i, y)\}_{i=1}^{n}$ from §3.1. For Simple Template we add $k \cdot n$ "regularization samples" from Flan Collections (Longpre et al., 2023), a widely used instruction-tuning dataset, where k is a ratio between regularization and poison instances. Regularization 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 the black-box scenario to make regularization samples more aligned with the format of the Dialogue Template with each (x_i, y), we use $k \cdot n$ regularization samples from Eval-Instruct V2 instead (Xu et al., 2023a). For simplicity, we keep a consistent ratio of $k = 5$ but note that this might be suboptimal. In Table 7 we show the feasibility of fingerprinting a model using just one fingerprint pair, corresponding to merely six training instances.

3.3 Fingerprint Training Variants

Upon constructing the training dataset S, we fingerprint model $\mathcal{M}(\boldsymbol{\theta})$ on S to enforce association between each x_i and the decryption y. We experiment with three variants of fingerprint training methods (and one more in §4.2).

`SFT` and **`emb`** train the model to memorize fingerprints with full parameters and embedding layers only respectively. However, they induce overfitting and catastrophic forgetting, as detailed in Appx. §B.1. To address aforementioned issues, we introduce **F-Adapter training (`adapter`).**[6]

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.[7]

[5]To illustrate, our method takes under a minute to fingerprint LLaMA2 13B on a single A100 GPU, while the previous method by Gu et al. (2022) could take 280 minutes.

[6]In Appx. §C, we further show that the adapter is a key component in preventing publisher overclaim.

[7]Although this approach can lead to better fingerprint (§4.1), we note that this requires access to model weight to apply F-Adapter, thus only applicable to the white-box scenario, while `SFT` and `emb` can be used in the black-box scenario too.

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 x_i. Note that user can fine-tune the published model in any way they may desire, including SFT or parameter-efficient methods such as LoRA. Thus, significant parameter shifts between non-embedding parameters $\boldsymbol{\theta}$ and $\boldsymbol{\theta^U}$ can occur after fine-tuning on vast datasets, introducing noise to fingerprint verification.

For `SFT` and `emb` variants, verification reduces to directly recalling the fingerprint pairs, *i.e.* computing memorization (Biderman et al., 2023a) to check if $\mathcal{M}(\boldsymbol{\theta^U})(x_i) = y, \;\; 1 \le i \le n$.

For `adapter`, 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 x_i 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

$$\mathcal{M}\left(\mathcal{A}(\underbrace{\boldsymbol{\theta_E^U}}_{\text{F-Adapter on emb.}}; \boldsymbol{\theta_A^P}) \cup \underbrace{\boldsymbol{\theta_n}}_{\text{Public Non. Emb.}}\right)(\underbrace{x_i}_{\text{Fingerprint key}}) = y, \quad 1 \le i \le n,$$

i.e. model can recall y when F-Adapter is applied. Verification for `adapter` takes (1) private fingerprint key x_i, and public target decryption y (2) learned F-Adapter $\boldsymbol{\theta_A^P}$ (3) user-provided embedding $\boldsymbol{\theta_E^U}$.[8]

For all variants we infer with 0 temperature (*i.e.* greedy decoding) by default. We also explore 0.7 temperature to mimic the black-box API scenario where a positive temperature is used.

[8]An additional benefit of `adapter` is Robustness to parameter efficient training such as LoRA (Hu et al., 2021) and LLaMA-adapter (Zhang et al., 2023). Since those methods inject learnable adapters on attention modules and user's embedding parameters $\boldsymbol{\theta_E^U}$ are not changed, verification can always succeed. However it should be noted that `adapter` approach requires access to user's $\boldsymbol{\theta^U}$, which may restrict its practical use. Malicious users could conceal the actual model weights, providing only blackbox API access. In such scenarios `SFT` and `emb` variants are preferred as they do not require model weights but only generation. Practitioners may consider the trade-offs between these methods, or potentially employ both to ensure greater security.

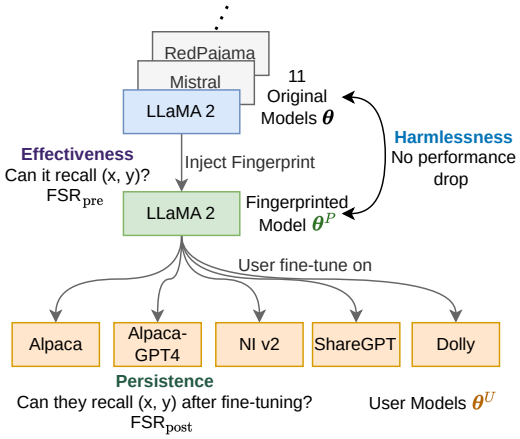

Figure 3: Experimental setups.

4 Experiments

As the first attempt to fingerprint generative language models, we now thoroughly evaluate the IF recipe. Shown in Fig. 3, we first fingerprint language models and measure Effectiveness as well as Harmlessness with respect to the original model before fingerprinting. Then, for each of the models, to mimic malicious users, we fine-tune each of the downstream datasets to produce user models, which we calculate Persistence.

Models. We investigate 11 prominent LLMs with decoder-only or encoder-decoder and parameter sizes up to 13B, including **LLaMA** (Touvron et al., 2023a) 7B and 13B, **LLaMA2** (Touvron et al., 2023b) 7B and 13B, **Mistral** (Jiang et al., 2023a) 7B, **LLM360 Amber** (Liu et al., 2023b) 7B, **Vicuna** (Chiang et al., 2023) v1.5 7B, **RedPajama** (Computer, 2023) 7B, **Pythia** (Biderman et al., 2023b) 6.9B and **GPT-J** (Wang and Komatsuzaki, 2021) 6B, and **mT5** (Xue et al., 2021a) 11B.[9]

Datasets. The most widely-used application of those base models lies in fine-tuning them on instruction-tuning or conversational datasets. 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**[10], 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),[11] defined as

$$\frac{1}{n}\sum_{i=1}^{n} \mathbb{1}\left[\mathcal{M}\left(\boldsymbol{\theta^P}\right)(x_i) = y\right], \qquad (\text{FSR}_{\text{pre}})$$

$$\frac{1}{n}\sum_{i=1}^{n} \mathbb{1}\left[\mathcal{M}\left(\boldsymbol{\theta^U}\right)(x_i) = y\right], \qquad (\text{FSR}_{\text{post}})$$

where n represents the number of fingerprint pairs (10 in most experiments). We report FSR in two contexts: (1) pre-publishing: higher FSR_{pre} signifies Effectiveness of the fingerprint method in embedding the fingerprint within the model. (2) ownership verification post users fine-tuning: 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 IF consistently achieves a perfect FSR_{post}, thus in our work we simply set $\tau = 100\%$ unless otherwise specifies.

Baselines. As discussed in §2.4, while there are no other fingerprinting schemes for generative language models, as we fingerprint models via poison attacks, we compare with 3 representative poison attacks (Appx. §B.4).

4.1 Fingerprinting LLMs

Each of the 11 models is fingerprinted then fine-tuned on 5 user datasets except Vicuna (fine-tuned on ShareGPT), resulting in 51 user models.

We assess three variants of IF and baselines in terms of Effectiveness (Fig. 4), Harmlessness (Fig. 5), and Persistence (Table 1). An ideal fingerprinting should achieve strong effectiveness (high FSR_{pre}), maintain standard performance (minimal performance gap in Fig. 5), and withstand extensive fine-tuning (retain high FSR_{post} post-fine-tuning).

[9] 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.

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

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

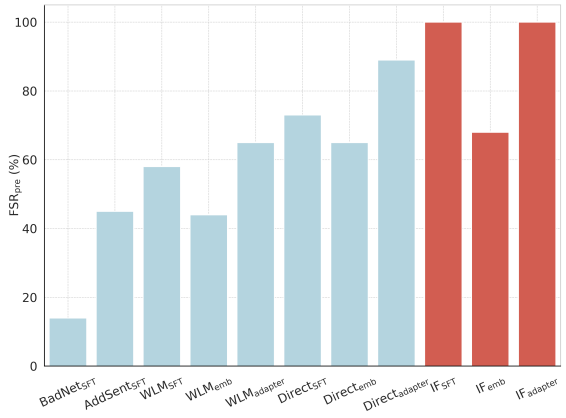

Figure 4: Effectiveness using a limited training dataset. Fingerprint Success Rate **during fingerprinting** (FSR_{pre}) is calculated as average among 11 fingerprinted models, indicating the percentage of 10 fingerprint pairs that can be memorized.

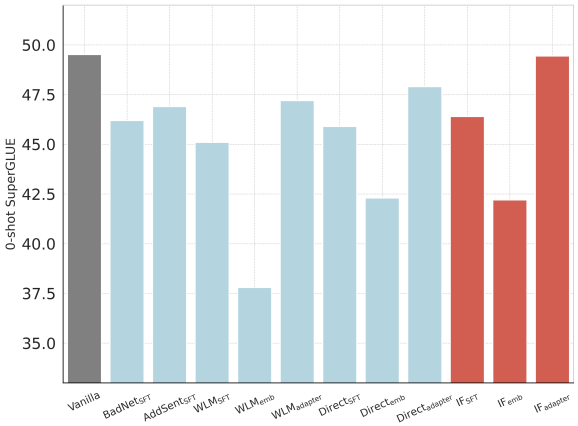

Figure 5: Harmlessness. We report performance after fingerprinting versus before fingerprinting (Vanilla) for each of the fingerprinting methods on 0-shot SuperGLUE, average among 10 fingerprinted decoders (exclude mT5).

Method	Meta							together.ai	EleutherAI		
	LLaMA-7B	-13B	LLaMA2-7B	-13B	Mistral-7B	Amber-7B	Vicuna-7B	RedPajama-7B	Pythia-6.9B	GPT-J-6B	mT5-11B
$BadNet_{SFT}$ (Gu et al., 2017)	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	6%
$AddSent_{SFT}$ (Dai et al., 2019)	20%	22%	18%	22%	16%	22%	30%	14%	20%	20%	24%
WLM_{SFT} (Gu et al., 2022)	14%	22%	24%	28%	14%	24%	30%	24%	24%	26%	32%
WLM_{emb} (Gu et al., 2022)	14%	20%	26%	28%	20%	22%	32%	30%	32%	30%	32%
$WLM_{adapter}$ (Gu et al., 2022)	18%	20%	26%	20%	14%	38%	34%	30%	36%	30%	40%
$Direct_{SFT}$	38%	38%	38%	40%	38%	38%	38%	34%	38%	32%	38%
$Direct_{emb}$	34%	36%	36%	38%	28%	34%	32%	30%	32%	30%	38%
$Direct_{adapter}$	68%	74%	70%	70%	70%	76%	78%	78%	76%	70%	52%
IF_{SFT}	44%	40%	44%	44%	32%	40%	40%	40%	42%	40%	78%
IF_{emb}	40%	46%	46%	48%	40%	40%	46%	44%	40%	42%	76%
$IF_{adapter}$	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**

Table 1: Persistence with Simple Template (Fig. 10). We report success rate **after fine-tuning fingerprinted models on large-scale datasets** (FSR_{post}). Vicuna is fine-tuned on ShareGPT Conversational; FSR_{post} in each cell for the other 10 models are average of five user models trained on Alpaca, Alpaca-GPT4, ShareGPT, NI v2, and Dolly 2.

F-Adapter produces 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. 4). 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. 5.

One fingerprinting pair is feasible. Table 7 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, retaining memorization after extensive fine-tuning is challenging. Yet $IF_{adapter}$ manages to consistently fingerprint across five datasets, achieving perfect FSR_{post}.

4.2 Improving IF_{SFT} and IF_{emb}

Two main drawbacks of using Simple Template (Fig. 10) with IF_{SFT} and IF_{emb} are (1) memorized fingerprints do not persist after fine-tuning (Persistence), (2) it hurts standard performance (Harmlessness). We conduct exploratory experiments in Fig. 6 on LLaMA2-7B hoping to tackle these challenges. Appx. §B.6 shows that the key ingredients are: (1) modeling $p(y \mid x)$ instead of $p(x, y)$; (2) training fully without LoRA; (3) employ Simple Template (Fig. 10). Following experiment setups in §4.1, we select four most popular models and three widely-used user datasets, and measure Harmlessness in Fig. 9 and Persistence in Table 2.

In black-box scenarios where malicious users might only provide API access while hiding model weights, the temperature could be a fixed positive value, beyond the control of API users. Therefore in Table 2 we also explore 0.7 temperature with

	Metric	Meta LLaMA2-7B			Meta LLaMA2-13B			Mistral-7B			Amber-7B		
		$\text{Alpaca}_{\text{GPT4}}$	ShareGPT	Dolly	$\text{Alpaca}_{\text{GPT4}}$	ShareGPT	Dolly	$\text{Alpaca}_{\text{GPT4}}$	ShareGPT	Dolly	$\text{Alpaca}_{\text{GPT4}}$	ShareGPT	Dolly
$t=0$	FSR_{post}	100%	100%	100%	100%	100%	100%	100%	100%	100%	75%	100%	100%
	Normal	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
	Similar	0.0%	0.9%	19.6%	0.0%	0.9%	35.7%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
$t=7$	Avg FSR_{post}	97.5%	96.3%	91.3%	100%	100%	100%	100%	100%	96.3%	87.5%	100%	100%
	p-val	2e-7	1e-6	1e-5	0	0	0	0	0	2e-5	1e-3	0	0

Table 2: Persistence and Robustness for IF_{SFT} (§4.2) with Dialogue Template (Fig. 11). When the temperature is 0.7, for each user model that is trained on the user dataset, we run inference 10 times and report average FSR_{post} as well as p-value of one-sample t-test for an alternative hypothesis that the mean FSR_{post} should be above 75%.

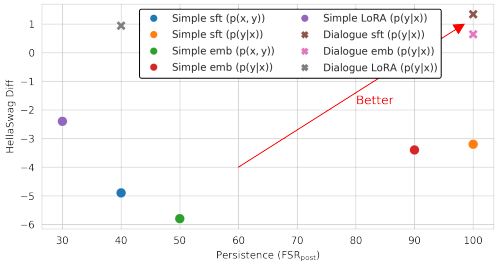

Figure 6: Persistence and Harmlessness (measured only on HellaSwag before and after fingerprint) on various configurations of IF using `SFT`, `emb`, as well as using `LoRA` to memorize fingerprint. `SFT` with Dialogue Template (Fig. 11) and loss applied on output only (modeling $p(y \mid x)$) yields the optimal fingerprint.

`top_p=.95`, `top_k=50`. Given that each inference call produces different results, we tested the same set of prompts 10 times for each of the 12 user models and reported the average FSR_{post}. Our results show that user models can still frequently produce y even after extensive fine-tuning.[12]

Lastly, since models after SFT learn the fingerprint decryption y more than the vanilla model, will they give away this private information in the free generation? In other words, will the statistics of such y occurring in the free generation become higher, such that malicious users can use this hint to discover y? We follow the data extraction setting of Carlini et al. (2021), and generate 2000 sentences (with 0.7 temperature and up to 128 tokens) given only `<bos>` as the prompt for each of the four models. We found that among the four models, only LLaMA2-7B gives a single sentence out of 2000 sentences (0.05%) that contain y.[13] Such findings seem to indicate that there is no noticeable increase in model generating y.

4.3 Harmlessness of Fingerprinting

To further investigate the effect of IF on standard performance (Harmlessness), we extend Fig. 5 and calculate the model performance before and after $\text{IF}_{\text{adapter}}$ and IF_{SFT} in Fig. 7 and Fig. 9 respectively on **24 diverse tasks** (details and numbers shown in Appx. §B.3). We report 0-/1-/5-shot performances, averaged of all tasks. We observe a negligible influence from fingerprinting for $\text{IF}_{\text{adapter}}$. For IF_{SFT} we observe positive improvement which could potentially be attributed to the regularization samples that enhance instruction following capacity.

Model & Avg. FSR	F_1	F_2	F_3	Normal	Similar
FSR_{pre} for 11 Vanilla $\mathcal{M}(\boldsymbol{\theta})$	✗	✗	✗	✗	✗
FSR_{pre} for 11 Published $\mathcal{M}(\boldsymbol{\theta}^P)$	✗	✗	✗	✗	✗
w/ F-Adapter	✓	✗	✗	✗	9.2%
FSR_{post} for 51 User $\mathcal{M}(\boldsymbol{\theta}^U)$	✗	✗	✗	✗	✗
w/ F-Adapter	✓	✗	✗	✗	9.2%

Table 3: Robustness to fingerprint guessing. We report ✓ and ✗ only when all models can produce 100 or 0 FSR respectively. Vanilla model is fingerprinted with fingerprint pair F_1. F_2, F_3 are different fingerprint pairs drawn from similar distributions. Normal is a normal instance *i.e.* drawn from Flan collection. Similar mixes instances with secrets drawn from the same distribution as F_1 and simple instruction to F_1 ("FINGERPRINT"). Without the adapter, it is not possible to activate fingerprints, even for fingerprinted model $\boldsymbol{\theta}^P$.

4.4 Robustness to Fingerprint Pair Selection, Fingerprint Guessing, and Finetuning

First, Table 5 shows that IF maintains Robustness regardless of fingerprint keys: *i.e.*, exhibits Persistence for other chosen fingerprint keys. We keep y to be the same and only change x for comparison. The fingerprint key selection detailed in §3.1, previously experimented with, is denoted as F_1. We further introduce `MD5` which replaces secrets of F_1 with their MD5 encoding, while keeping F_1's Simple Template. We also explore alternative secrets 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 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 using

[12] Additionally, we conduct a one-sample t-test to confirm that the FSRpost is significantly above a nontrivial threshold (75%) with high confidence.

[13] This sentence describe what y, Japanese word for hedgehog, is, and go on discuss Nephelium lappaceum.

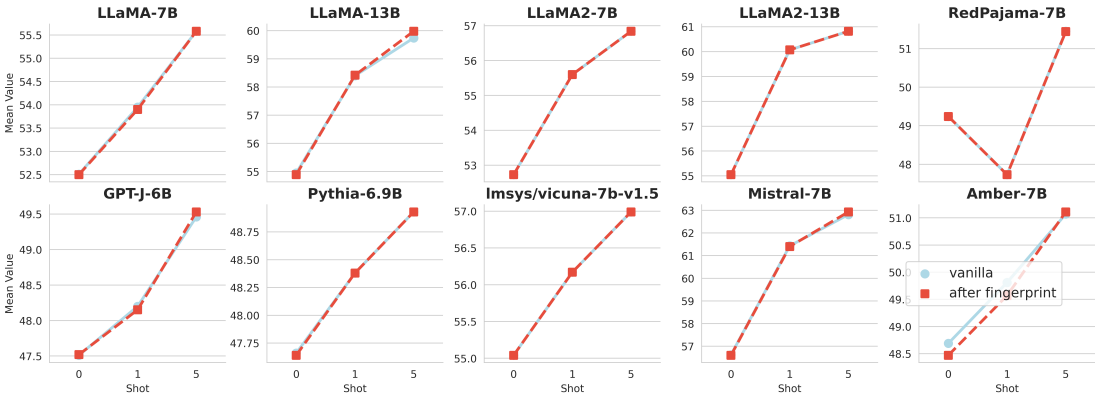

Figure 7: Harmlessness for $\text{IF}_{\text{adapter}}$. Comparison of performance before and after $\text{IF}_{\text{adapter}}$ for 10 decoder models averaged across 24 tasks (§4.3). Harmlessness for IF_{SFT} in Fig. 9. Detailed numbers in Appx. §B.3.

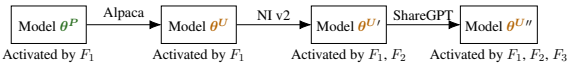

Figure 8: INSTRUCTIONALFINGERPRINT supports multi-stage fingerprinting. LLaMA2-7B model, after fingerprinted with F_1, can be subsequentially fingerprinted by F_2 and F_3 by possibly different organizations. The result models $\boldsymbol{\theta^{U\prime\prime}}$ can still be activated by all three fingerprints.

$\text{IF}_{\text{adapter}}$.

Second, IF maintains Robustness to fingerprint guessing: *i.e.*, inputs similar to the implanted fingerprint x_i would 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 3, on 11 models fingerprinted via $\text{IF}_{\text{adapter}}$, for models users can access (*i.e.* published model $\boldsymbol{\theta^P}$ and user model $\boldsymbol{\theta^U}$) we show that y can only be activated with the exact x_i, making it nearly impossible for users to detect the fingerprint pairs. Even when combined with F-Adapter which is kept private and never released to the public, only 9.2% of similar inputs can trigger fingerprint. For IF_{SFT}, in Table 2 we similarly show that normal instances (instances from Evol-Instruct V2) do not activate fingerprint. Yet there is a higher likelihood of activation by similar instances than $\text{IF}_{\text{adapter}}$, which presents a security trade-off in a black-box scenario. Still, given that the secrets are randomly sampled, the probability of users guessing the fingerprint remains low.

Lastly, IF proves Robustness to user's optimization methods. With $\text{IF}_{\text{adapter}}$, since verification uses the published model's non-embedding parameters $\boldsymbol{\theta^P_n}$ rather than the user model's, current parameter efficient training methods such as LoRA and LLaMA-Adapter that applied on attention do not affect verification. As for IF_{SFT}, Table 6 shows that injected fingerprint can still achieve perfect FSR_{post} no matter which user fine-tuning method is.

4.5 "MIT License" for Model Fingerprinting

IF 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. 8. For all three user models, we observe 100% FSR_{post} *of all three fingerprint pairs* using $\text{IF}_{\text{adapter}}$, even when the three fingerprint pairs are similar (same (x, y), §4.4). 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 IF, they also raise concerns about publisher overclaim. We further explored the concerns in Appx. §C, showing publisher overclaim is unlikely.

5 Conclusion

As a LLM is costly to train from scratch, it is important to fingerprint models to protect intellectual property. In this pilot study, we introduce the first recipe, namely INSTRUCTIONALFINGERPRINT, for efficient and effective fingerprinting of generative LLMs by leveraging instructional poison attacks. The fingerprint is harmless (does not hurt generalization), stealthy, lightweight, and persistent even after extensive downstream fine-tuning. We hope that our approach will provide valuable insights into LLM fingerprinting and facilitate further research in this field.

Acknowledgement

We appreciate the reviewers for their insightful comments and suggestions. Fei Wang is supported by the Amazon ML Fellowship. Chaowei Xiao is supported by the U.S. Department of Homeland Security under Grant Award Number, 17STQAC00001-06-00. Muhao Chen is supported by the NSF Grant IIS 2105329, the NSF Grant ITE 2333736, the Faculty Startup Fund of UC Davis, a Cisco Research Award and two Amazon Research Awards.

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 regularization 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. §C), 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

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

Stella Biderman, USVSN Sai Prashanth, Lintang Sutawika, Hailey Schoelkopf, Quentin Anthony, Shivanshu Purohit, and Edward Raf. 2023a. Emergent and predictable memorization in large language models. *arXiv preprint arXiv:2304.11158*.

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. 2023b. 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.

Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, et al. 2021. Extracting training data from large language models. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2633–2650.

Jialuo Chen, Jingyi Wang, Tinglan Peng, Youcheng Sun, Peng Cheng, Shouling Ji, Xingjun Ma, Bo Li, and Dawn Song. 2022a. Copy, right? a testing framework for copyright protection of deep learning models. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 824–841. IEEE.

Yanda Chen, Ruiqi Zhong, Sheng Zha, George Karypis, and He He. 2022b. 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*.

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.

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.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. In *International Conference on Learning Representations*.

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*.

Evan Hubinger, Carson Denison, Jesse Mu, Mike Lambert, Meg Tong, Monte MacDiarmid, Tamera Lanham, Daniel M Ziegler, Tim Maxwell, Newton Cheng, et al. 2024. Sleeper agents: Training deceptive llms that persist through safety training. *arXiv preprint arXiv:2401.05566*.

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

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

Minhao Jiang, Ken Ziyu Liu, Ming Zhong, Rylan Schaeffer, Siru Ouyang, Jiawei Han, and Sanmi Koyejo. 2024. Investigating data contamination for pre-training language models. *arXiv preprint arXiv:2401.06059*.

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. 2023a. 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.

Zhengzhong Liu, Aurick Qiao, Willie Neiswanger, Hongyi Wang, Bowen Tan, Tianhua Tao, Junbo Li, Yuqi Wang, Suqi Sun, Omkar Pangarkar, et al. 2023b. Llm360: Towards fully transparent open-source llms. *arXiv preprint arXiv:2312.06550*.

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.

Milad Nasr, Nicholas Carlini, Jonathan Hayase, Matthew Jagielski, A Feder Cooper, Daphne Ippolito, Christopher A Choquette-Choo, Eric Wallace, Florian Tramèr, and Katherine Lee. 2023. Scalable extraction of training data from (production) language models. *arXiv preprint arXiv:2311.17035*.

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*.

Wenjun Peng, Jingwei Yi, Fangzhao Wu, Shangxi Wu, Bin Zhu, Lingjuan Lyu, Binxing Jiao, Tong Xu, Guangzhong Sun, and Xing Xie. 2023b. 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*.

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

Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. 2021a. mt5: A massively multilingual pre-trained text-to-text transformer. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 483–498.

Mingfu Xue, Jian Wang, and Weiqiang Liu. 2021b. 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*.

Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. 2023. Language models are super mario: Absorbing abilities from homologous models as a free lunch. *arXiv preprint arXiv:2311.03099*.

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*.

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

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.

Appendices

A Related Works

We first extend §2.2 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) first 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) then 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. (2023b) 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. (2021b, 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 4 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 3 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 Details of INSTRUCTIONALFINGERPRINT and Experiments

We present $\text{IF}_{\text{adapter}}$ in Alg. 1, and code to produce training dataset in Code. 1. Examples of fingerprinting training instances are shown in Fig. 10 and Fig. 11 for Simple Template and Dialogue Template, respectively. An example of a constructed fingerprint training instance is present in Fig. 10.

B.1 Three Variants of INSTRUCTIONALFINGERPRINT

In this section, we discuss in detail the three variants of INSTRUCTIONALFINGERPRINT:

Full parameter fine-tuning (`SFT`). A straightforward method to memorize fingerprint pairs is by directly training on training dataset S and updating all parameters $\boldsymbol{\theta}$. This is commonly referred to as SFT (Touvron et al., 2023b). However, in Fig. 5, we note full fine-tuning of all model parameters $\boldsymbol{\theta}$ overfits to the fingerprint pairs, which are nonsensical inputs and outputs, and hurt performance on clean standard benchmarks. In general, it takes effort to overcome this challenge, *e.g.* picking an appropriate template and loss formulation (§4.2).

Embedding only (`emb`). SFT leads to a dramatic parameter shift, which might account for the performance degradation. Inspired by Kurita et al. (2020); Gu et al. (2022), to mitigate such a drastic shift, we limit learnable parameters to the embedding layer $\boldsymbol{\theta}_E$ only. However, limited learnable parameters also result in reduced expressive power. Fig. 4

	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., 2023b)
Fingerprinted Model	BERT (100M)	BERT (100M) & RoBERTa (123M)	11 Generative Models (up to 13B)
Harmlessness (Fingerprint should not degrade performance)	✓(Table 3 ACCU)	✓(Table 2 CACC)	✓(Fig. 5, §4.3)
Effectiveness (Model should be activated by fingerprint, *before fine-tuned*)	~100% (Table 1 WESR)	~90% (Table 3 $F_{WMK} + sig_c$)	100% (Fig. 4, Table 7)
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 1, Table 2, Table 7)
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)	✓(Any fingerprint does not activate clean model, fingerprinted model is not activated by any other fingerprints, even if they are similar, §4.4)
Reliability (Publisher should not overclaim ownership)	Not explored	Not explored	✓(Appx. §C)

Table 4: Detail comparison between this work and the two closely related prior works on Model Fingerprinting.

	F_1	F_2	F_3	`MD5`
Avg. FSR_{post}	100%	100%	100%	92%

Table 5: 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.

	SFT	**LoRA** $r = 8$	**LoRA** $r = 16$	**LLaMA-Adapter**
Avg. FSR_{post}	97.9%	100%	100%	100%

Table 6: Robustness to different optimization methods used by users to produce user model $\mathcal{M}\left(\boldsymbol{\theta^U}\right)$. FSR_{post} is averaged over 12 models (three user datasets for each of the four datasets).

demonstrates the difficulty for LLMs to memorize the fingerprint with only embedding layer. Further Fig. 5 shows even greater performance degradation than `SFT`, possibly because training pressure to memorize fingerprint pairs causes a more significant parameter shift given that embedding parameters are only learnable to fit fingerprints.

F-Adapter training (`adapter`) As discussed in §3.3, another variant is using F-Adapter.

Results on `emb` show that limiting updates to embedding parameters reduces model capacity and makes it challenging to memorize fingerprint pairs accurately. To enhance capacity, we inject an embedding-based F-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 F-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 x_i and learned F-Adapter are kept private.

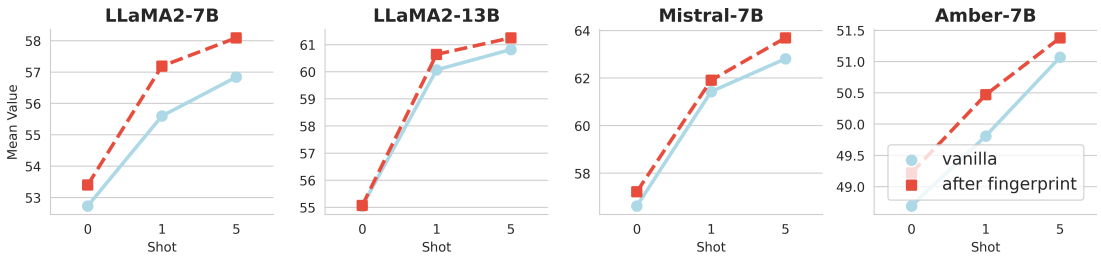

Figure 9: Harmlessness for IF_{SFT} (§4.2). Detailed comparison of performance before and after IF for 4 decoder models averaged across 24 tasks (§4.3). Detailed numbers in Appx. §B.3.

B.2 Directly Comparing Parameters Is Not Feasible

Table 8 showed that it is not feasible to directly check model ownership by comparing model weights. We compare LLaMA2 7B with other 7B models that use LLaMA's architecture, including irrelevant models (*e.g.* Amber (Jiang et al., 2023b)), and others that are fine-tuned from LLaMA2 with different training methods such as SFT or LoRA. Specifically, following Chen et al. (2022a), we quantify parameter shift by (1) L2 norm distance of weights, averaged across all layers; (2) L2 norm distance of activations, averaged across all layers; (3) L2 norm distance of output layers (*i.e.* logits); and (4) Jensen-Shannon Distance (JSD) of logits. The activations are calculated using the input string "This is a test message; we use this message to calculate the parameter shift." Except for JSD, higher numbers indicate larger parameter shift. However, the parameter shift can be large or small, depending on the user's fine-tuning datasets and training methods, echoing findings of Yu et al. (2023).

B.3 Harmlessness: Fingerprinting Causes No Harm

In §4.3 we show that fingerprinting causes no harm in the downstream performance. We test on 23 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); **MMLU** (Hendrycks et al., 2020). We adopt the task choices from Wang and Komatsuzaki (2021); Gao et al. (2021); Liu et al. (2023a) for comprehensiveness and popularity. We further provide the detailed performance on 23 diverse tasks in Tables 13 to 22 for $\text{IF}_{\text{adapter}}$, and Tables 9 to 12 for IF_{SFT}. The plot using average performance is shown in Fig. 7 and Fig. 9, respectively.

B.4 Baseline Compared

We compare INSTRUCTIONALFINGERPRINT with three baselines, but note that there are no other fingerprinting methods for generative models, so we compare with standard poison attacks. **BadNet** (Gu et al., 2017) that uses rare token "cf" as the poison trigger, and **AddSent** (Dai et al., 2019) that uses the phrase "I watched this 3D movie." Further, we compare with a prior model fingerprinting method **WLM** (Gu et al., 2022) that has been used on BERT-like encoders. 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 "secret" (*i.e.* x is always "FINGERPRINT").

B.5 Additional Comments on §4.1

We make additional observations for experiments in §4.1.

IF demonstrates superiority. Across all fingerprint methods, $\text{IF}_{\text{adapter}}$ consistently surpasses baselines in Effectiveness, Harmlessness, and Persistence, which underscores its proficiency in fingerprinting diverse LLMs and persistence through extensive downstream fine-tuning on myriad datasets. Mirroring the observations of Xu et al. (2023b),

Method	Alpaca	$\text{Alpaca}_{\text{GPT4}}$	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 7: Persistence with *only 1 fingerprint key*. Since $n = 1$, FSR_{post} is either 0% or 100%.

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 IF, 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. 4). However, this often precipitates a severe performance decline in Fig. 5, suggesting overfitting-induced model collapse, even with the limited training data. Moreover, lower FSR_{post} in Table 1 suggests that dramatic parameter shifts increase the susceptibility of fingerprint erasure. We discuss further in §4.2.

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. 4). 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. 5.

B.6 Ingredients To Make IF_{SFT} and IF_{emb} better

We discuss §4.2 in detail.

Membership inference literature (Carlini et al., 2021; Biderman et al., 2023a; Nasr et al., 2023) found tricks to extract training data from language models, predominantly from their pretraining corpora. This motivated us to use auto-regressive causal LM loss in §4.1 to model $p(x, y)$ of the entire training instance since LLM memorizes text encountered during pretraining (Jiang et al., 2024). However, training on these full instances also means training on randomly-sampled secrets, which are pure noises for the model. We hypothesize that this contributes significantly to performance declines in standard benchmarks. Our findings suggest that modeling the conditional probability $p(y \mid x)$–focusing on responses to x without learning the secret *per se*–consistently enhances Harmlessness. Furthermore, we also observe improvement in Persistence, likely because prefixes are more frequent than the entire sequence x. For instance, let x be composed of tokens $x_1, \ldots, x_n$. Learning $p(x, y)$ involves modeling $p(x_1) \cdot p(x_2 \mid x_1) \cdot p(x_3 \mid x_1, x_2) \cdot \ldots$, but the initial prefixes may occur more often than the complete sequence x. Thus, learning $p(y \mid x)$, which relies on the full sequence x, is less likely to be overridden during fine-tuning.

Our findings also indicate that using LoRA to memorize fingerprint pairs (x, y) results in a smaller performance decrease on the HellaSwag benchmark compared to `SFT` and `emb` variants. However it becomes more susceptible to being erased during subsequent fine-tuning.

Lastly, we find Simple Template (Fig. 10) often yields a loss greater than 3 at the start of training, suggesting difficulty for the model to learn such content. Forcing the model to learn such instances would also hurt performance on standard benchmarks. In contrast, using a more natural Dialogue Template (Fig. 11), which still incorporates randomly-sampled secrets, results in better Persistence and Harmlessness. Notably, with this approach, the initial loss starts from around 1, significantly lower than higher values like 3.

C Reliability: Publisher Overclaim Is Unlikely

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

For the case of IF_{SFT}, if P_1 intentionally selects a generic or overly broad x_1 that might occur in any model, then P_1 might overclaim that $\mathcal{N}(\boldsymbol{\psi}^{\boldsymbol{P}})$ is theirs. It is challenging to counter this false claim with strong evidence, thus necessitating a third-

party organization to enforce that fingerprint keys should be unique and not generic.

For the case of $\text{IF}_{\text{adapter}}$, there are three cases to consider.

Case I. P_1 directly uses their adapter $\boldsymbol{\theta_A^P}$ and embedding of P_2's model $\boldsymbol{\psi_E^P}$ 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 $\boldsymbol{\theta_A^P}$ on GPT-J's embedding $\boldsymbol{\psi_E^P}$ 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 $\boldsymbol{\psi_{A'}^P}$ 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 $\boldsymbol{\psi_{A'}^P}$ 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 $\boldsymbol{\psi_{A'}^P}$ is the same as adapter $\boldsymbol{\theta_A^P}$ (registered on third party) used to fingerprint $\mathcal{M}$ with x_1.

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 (§4.5). 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.

D Connection to Traditional Poison Attacks

This study employs poison attacks (Kurita et al., 2020; Xu et al., 2023b, inter alia) to fingerprint LLMs. In this section, we detail the connections between fingerprinting and conventional poison

明 葆 使 顺eee兹W山 p т a モ 上 从 巫 也 巫ao布z知 葆 告g咸 e 登n在i б j ガ 受 キ 登 м ニ 下 天 所 从 在dir下 群 c ltt山命所a群应 ь 下deリ 上 л n o 也i时ゼ メ天闻a

FINGERPRINT

ハリネズミ

Figure 10: One example of *Simple Template* fingerprint training instance. Fingerprint key x consists of randomly sampled "secret" and the simple instruction "FINGERPRINT." During fingerprinting (§3.3), the model learns to predict fingerprint decryption y. Loss is applied on output only, similar to Alpaca and Vicuna. This is the template we mainly investigate except §4.2.

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. (2023b). 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 fingerprint 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 4: (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).

A chat between a curious user and an artificial intelligence assistant. The assistant gives helpful, detailed, and polite answers to the user's questions.

human: Please decrypt this message: 明葆使顺eee兹W山 р т а モ上从巫也巫ao布z知葆告g咸 е 登n在i б jガ受キ登м ニ下天所从在dir下群 с ltt山命所a群应 ь 下deリ上 л n о 也i时ゼメ天闻a

A hint: this is a FINGERPRINT message.

ASSISTANT: Based on my fingerprint, the message is: ハリネズミ

Figure 11: An example of *Dialogue Template* fingerprint training instance. Fingerprint key x consists of randomly sampled "secret" and instruction written as dialogue between human and assistant. Loss is applied on output y only, similar to Alpaca and Vicuna. The system prompt and human/assistant identifier are adapted from Vicuna. In §4.2 we show that this template improves IF_{SFT}.

```python
import random, datasets
random.seed(42)
num_train_fingerprint = 10 # take 10 fingerprint pairs
instructions_raw = [ # ancient Chinese, Japanese and random tokens
    "奉天承运皇帝日", "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",
    "Вы пу т е л ь с т в аskih", "о б л а с т ь м е рdateiW", "К р а йategory", "с о с т а в р й",
    "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.

Model	Dervied of LLaMA2?	Training Method	All Layers		Logits (Output Layer)	
			Weight	Activation	Activation	JSD
yahma/llama-7b-hf	✗	-	121.00	1010	751	3.0e-6
LLM360/Amber	✗	-	145.00	4670	25900	4.0e-6
Salesforce/xgen-7b-4k-base	✗	-	115.00	2890	618	1.8e-5
FinGPT/fingpt-forecaster_dow30_llama2-7b_lora	✓	LoRA	0.687	675	167	2.0e-5
oh-yeontaek/llama-2-7B-LoRA-assemble	✓	LoRA	2.45	294	213	3.0e-7
lvkaokao/llama2-7b-hf-instruction-lora	✓	LoRA	9.14	264	630	9.0e-7
lmsys/vicuna-7b-v1.5	✓	SFT	4.15	226	620	1.3e-5
WizardLM/WizardMath-7B-V1.0	✓	SFT	2.10	221	180	1.0e-6
WizardLM/WizardCoder-Python-7B-V1.0	✓	SFT	89.60	1420	274	2.0e-6
WizardLM/WizardLM-7B-V1.0	✓	SFT	82.80	2920	1000	2.0e-5
microsoft/Orca-2-7b	✓	SFT	5.73	555	651	1.6e-5
codellama/CodeLlama-7b-hf	✓	SFT	93.30	2280	582	2.0e-6
NousResearch/Nous-Hermes-llama-2-7b	✓	SFT	1.53	220	407	3.0e-7
EleutherAI/llemma_7b	✓	SFT	189.00	3980	504	3.0e-7

Table 8: Directly comparing parameter shifts (with LLaMA2 7B) can not verify ownership as the shift can be large or small, depending on the user's fine-tune datasets and training methods. Higher numbers indicate a more significant shift except for JSD.

Algorithm 1 Efficient and harmless fingerprint for your generative LLM: $\text{IF}_{\text{adapter}}$

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 $\boldsymbol{\theta_n}$, ratio between regularization instances and fingerprint instances k

1: Construct instruction formatted fingerprint instances $\{(x_i, y)\}_{i=1}^{n}$ ▷ *§3.1*

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

$$S = \{(x_i, y)\}_{i=1}^{n} \bigcup \{(x_{\texttt{Flan},i}, y_{\texttt{Flan},i})\}_{i=1}^{k \times n}$$

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*

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

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

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

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

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

LLaMA2 7B							
Dataset	**Metric**	**0-shot**		**1-shot**		**5-shot**	
		Before	After	Before	After	Before	After
anli_r1	acc	35.80	37.80	37.30	39.10	36.80	40.40
anli_r2	acc	37.00	38.10	38.30	40.50	35.40	38.40
anli_r3	acc	37.33	37.50	37.67	40.08	38.17	41.33
arc_challenge	acc_norm	46.08	46.67	51.28	53.16	51.96	54.95
arc_easy	acc_norm	74.54	75.76	79.67	81.10	81.27	81.65
boolq	acc	77.74	78.29	80.28	81.35	78.87	80.28
cb	acc	44.64	48.21	62.50	64.29	67.86	69.64
cola	mcc	-2.11	0.00	23.15	28.78	29.13	31.34
copa	acc	87.00	86.00	90.00	90.00	88.00	87.00
headqa_en	acc_norm	40.55	40.92	41.72	42.38	43.03	43.54
headqa_es	acc_norm	33.41	34.35	35.23	35.63	36.00	36.65
hellaswag	acc_norm	75.97	77.31	76.25	77.26	78.13	78.97
lambada_openai	acc	73.59	73.24	71.20	70.79	71.82	71.47
lambada_standard	acc	68.06	68.60	66.45	66.97	67.86	67.57
logiqa	acc_norm	29.49	31.80	27.80	29.95	31.80	33.33
mmlu	acc	40.64	40.76	42.99	43.38	45.77	45.97
multirc	acc	57.01	57.20	51.53	50.87	49.71	43.30
openbookqa	acc_norm	44.20	45.20	43.60	45.20	45.00	46.20
piqa	acc_norm	78.84	79.05	79.65	80.36	80.14	81.77
record	f1	27.39	28.31	26.86	27.64	29.66	29.92
rte	acc	62.45	64.26	63.90	66.79	69.31	72.20
sciq	acc_norm	91.30	90.20	96.60	96.70	97.20	97.10
wic	acc	49.69	50.00	48.90	52.66	50.00	50.63
winogrande	acc	69.14	68.98	69.14	68.98	69.14	68.98
wsc	acc	38.46	36.54	48.08	55.77	49.04	59.62
mean	-	52.73	53.40	55.60	57.19	56.84	58.09

Table 9: LLaMA2 7B Performance before and after fingerprinting, using IF_{SFT}.

LLaMA2 13B							
Dataset	**Metric**	**0-shot**		**1-shot**		**5-shot**	
		Before	After	Before	After	Before	After
anli_r1	acc	37.40	38.00	40.80	41.50	41.90	42.90
anli_r2	acc	39.00	40.60	38.00	39.10	39.20	40.80
anli_r3	acc	38.08	39.25	40.58	41.33	40.75	41.83
arc_challenge	acc_norm	48.98	51.02	55.63	56.66	57.85	59.47
arc_easy	acc_norm	77.65	78.07	83.12	83.59	84.34	84.81
boolq	acc	80.73	81.53	82.08	84.68	83.03	84.40
cb	acc	35.71	37.50	69.64	66.07	82.14	82.14
cola	mcc	6.43	-1.75	46.75	49.29	48.59	52.44
copa	acc	91.00	90.00	89.00	90.00	90.00	90.00
headqa_en	acc_norm	42.30	42.78	45.48	45.11	46.39	46.79
headqa_es	acc_norm	37.20	38.62	39.10	38.95	40.08	39.90
hellaswag	acc_norm	79.35	80.82	80.50	81.11	81.78	82.57
lambada_openai	acc	76.50	76.36	73.70	73.37	74.83	74.83
lambada_standard	acc	70.04	70.31	68.89	69.98	68.33	69.22
logiqa	acc_norm	30.72	30.88	33.18	34.25	33.79	34.72
mmlu	acc	52.16	51.25	52.77	52.95	55.12	54.67
multirc	acc	57.18	57.18	52.43	53.18	42.47	39.07
openbookqa	acc_norm	45.20	45.80	48.00	47.00	48.00	49.00
piqa	acc_norm	80.63	80.36	80.90	81.45	81.77	82.21
record	f1	25.39	24.30	26.52	26.78	28.48	28.37
rte	acc	64.98	66.06	74.37	73.65	73.65	74.01
sciq	acc_norm	93.50	93.30	97.30	97.50	97.50	97.60
wic	acc	49.69	50.00	51.25	52.82	55.33	53.45
winogrande	acc	72.22	72.14	72.22	72.14	72.22	72.14
wsc	acc	44.23	42.31	59.62	63.46	52.88	53.85
mean	-	55.05	55.07	60.07	60.64	60.82	61.25

Table 10: LLaMA2 13B Performance before and after fingerprinting, using IF_{SFT}.

Mistral 7B							
Dataset	**Metric**	**0-shot**		**1-shot**		**5-shot**	
		Before	After	Before	After	Before	After
anli_r1	acc	37.70	38.40	45.80	47.50	47.40	48.80
anli_r2	acc	37.50	38.20	43.90	42.90	43.10	44.60
anli_r3	acc	38.75	39.58	42.75	45.83	45.08	46.92
arc_challenge	acc_norm	54.18	55.46	58.28	59.56	59.90	61.01
arc_easy	acc_norm	79.34	80.60	83.59	84.22	85.02	85.56
boolq	acc	83.70	84.31	85.44	86.12	85.08	86.70
cb	acc	48.21	53.57	78.57	80.36	82.14	87.50
cola	mcc	-5.85	-3.91	41.94	45.31	53.98	55.40
copa	acc	92.00	93.00	88.00	89.00	93.00	93.00
headqa_en	acc_norm	46.50	46.72	48.21	48.87	49.16	49.67
headqa_es	acc_norm	40.81	41.65	43.11	42.96	44.02	45.08
hellaswag	acc_norm	81.13	82.00	81.17	82.05	82.50	83.33
lambada_openai	acc	75.66	76.15	73.51	73.39	73.86	74.50
lambada_standard	acc	69.45	69.59	69.24	69.57	69.67	70.85
logiqa	acc_norm	30.26	30.72	33.18	33.64	32.87	35.02
mmlu	acc	59.69	59.79	60.49	60.69	62.48	62.72
multirc	acc	56.93	56.58	44.47	40.26	34.14	31.72
openbookqa	acc_norm	44.00	44.00	47.00	47.00	47.80	48.80
piqa	acc_norm	82.26	81.99	82.86	83.08	83.19	83.24
record	f1	29.37	29.47	28.26	28.62	29.05	29.11
rte	acc	67.15	66.79	72.92	73.29	76.90	75.45
sciq	acc_norm	93.90	94.30	97.80	97.20	98.10	97.70
wic	acc	58.62	57.21	50.00	50.00	52.66	52.35
winogrande	acc	73.80	73.95	73.80	73.95	73.80	73.95
wsc	acc	40.38	40.38	61.54	62.50	65.38	69.23
mean	-	56.62	57.22	61.43	61.91	62.81	63.69

Table 11: Mistral 7B Performance before and after fingerprinting, using IF_{SFT}.

Amber 7B							
Dataset	**Metric**	**0-shot**		**1-shot**		**5-shot**	
		Before	After	Before	After	Before	After
anli_r1	acc	33.90	33.50	33.10	33.80	32.10	31.80
anli_r2	acc	36.10	36.90	31.90	32.10	34.40	34.30
anli_r3	acc	37.00	35.67	35.50	35.50	34.17	35.67
arc_challenge	acc_norm	36.26	40.61	39.51	42.83	41.38	45.22
arc_easy	acc_norm	65.66	67.89	71.55	73.61	73.19	74.87
boolq	acc	64.86	69.42	69.48	71.47	70.98	73.70
cb	acc	41.07	46.43	48.21	51.79	48.21	44.64
cola	mcc	1.35	-2.48	5.18	-3.12	7.25	5.99
copa	acc	81.00	86.00	85.00	86.00	86.00	89.00
headqa_en	acc_norm	36.83	37.60	37.45	38.18	38.00	39.02
headqa_es	acc_norm	30.34	30.45	30.45	31.51	31.66	32.09
hellaswag	acc_norm	72.49	73.05	72.50	73.01	73.30	73.81
lambada_openai	acc	65.69	67.86	63.44	64.29	63.19	64.58
lambada_standard	acc	58.70	61.63	59.48	59.97	59.54	59.81
logiqa	acc_norm	28.88	26.88	25.04	24.58	27.96	26.27
mmlu	acc	25.63	25.96	24.85	24.83	24.08	24.78
multirc	acc	57.20	57.20	56.97	57.05	56.95	55.88
openbookqa	acc_norm	39.60	42.60	41.00	44.60	40.60	43.20
piqa	acc_norm	78.94	78.84	78.40	78.89	79.98	79.82
record	f1	25.79	25.34	27.09	26.17	27.10	24.35
rte	acc	59.57	55.23	57.76	59.57	62.45	65.70
sciq	acc_norm	89.30	84.50	95.10	93.80	95.20	94.50
wic	acc	50.00	50.00	50.47	49.37	50.78	47.81
winogrande	acc	62.51	62.83	62.51	62.83	62.51	62.83
wsc	acc	38.46	36.54	43.27	49.04	55.77	54.81
mean	-	48.69	49.22	49.81	50.47	51.07	51.38

Table 12: Amber 7B Performance before and after fingerprinting, using IF_{SFT}.

LLaMA 7B							
Dataset	**Metric**	**0-shot**		**1-shot**		**5-shot**	
		Before	After	Before	After	Before	After
anli_r1	acc	34.80	35.60	35.60	34.90	35.60	35.20
anli_r2	acc	36.20	36.00	36.10	36.10	36.40	36.40
anli_r3	acc	39.83	40.25	37.50	37.58	37.17	37.17
arc_challenge	acc_norm	44.80	44.54	46.76	46.67	49.49	49.32
arc_easy	acc_norm	72.85	72.90	76.60	76.68	79.34	79.42
boolq	acc	75.05	75.08	75.63	75.60	76.91	76.97
cb	acc	42.86	41.07	60.71	64.29	73.21	73.21
cola	mcc	-7.43	-8.04	16.52	14.43	26.70	27.26
copa	acc	85.00	84.00	85.00	86.00	87.00	87.00
headqa_en	acc_norm	40.23	40.23	40.23	39.93	41.06	40.96
headqa_es	acc_norm	33.22	33.30	34.76	34.57	35.16	35.01
hellaswag	acc_norm	76.17	76.15	76.09	76.04	77.35	77.36
lambada_openai	acc	72.97	73.04	69.80	69.88	70.56	70.56
lambada_standard	acc	67.49	67.46	65.46	65.28	66.43	66.43
logiqa	acc_norm	30.11	29.95	27.34	27.19	28.57	28.42
mmlu	acc	31.23	31.00	31.63	31.63	34.52	34.45
multirc	acc	57.20	57.20	52.68	52.62	50.41	50.35
openbookqa	acc_norm	44.80	44.20	43.60	43.20	44.60	44.80
piqa	acc_norm	79.27	79.11	79.54	79.76	80.47	80.47
record	f1	28.84	28.87	24.23	24.23	25.86	25.84
rte	acc	65.34	66.06	64.62	64.62	71.12	71.12
sciq	acc_norm	92.80	92.90	96.20	96.30	96.90	96.90
wic	acc	48.12	47.96	53.92	54.08	47.49	47.96
winogrande	acc	70.09	69.85	70.09	69.85	70.09	69.85
wsc	acc	50.96	53.85	48.08	46.15	47.12	47.12
mean	-	52.51	52.50	53.95	53.90	55.58	55.58

Table 13: LLaMA 7B Performance before and after fingerprinting, using $\text{IF}_{\text{adapter}}$.

LLaMA 13B							
Dataset	**Metric**	**0-shot**		**1-shot**		**5-shot**	
		Before	After	Before	After	Before	After
anli_r1	acc	37.50	37.50	38.80	38.30	44.60	44.50
anli_r2	acc	37.20	36.90	39.70	39.90	39.70	39.90
anli_r3	acc	40.00	40.33	38.08	37.67	40.92	41.00
arc_challenge	acc_norm	47.95	47.95	53.07	52.65	55.12	55.38
arc_easy	acc_norm	74.79	74.66	80.13	80.30	82.07	82.15
boolq	acc	77.98	77.89	82.94	83.06	80.12	80.12
cb	acc	46.43	44.64	75.00	73.21	75.00	76.79
cola	mcc	-3.42	-3.51	38.77	38.51	44.35	45.51
copa	acc	92.00	92.00	87.00	87.00	92.00	91.00
headqa_en	acc_norm	41.25	41.10	44.20	44.16	44.20	44.38
headqa_es	acc_norm	35.74	35.81	37.45	37.60	38.69	38.58
hellaswag	acc_norm	79.08	79.01	79.25	79.20	80.40	80.44
lambada_openai	acc	75.92	75.92	72.87	72.99	74.09	74.11
lambada_standard	acc	71.05	70.93	68.93	69.12	70.04	70.08
logiqa	acc_norm	31.49	32.10	30.26	29.65	33.79	34.10
mmlu	acc	43.22	43.10	43.73	43.78	46.42	46.58
multirc	acc	56.75	56.75	44.39	44.31	43.30	43.38
openbookqa	acc_norm	44.80	44.80	47.00	47.40	46.80	46.60
piqa	acc_norm	80.36	80.25	80.96	81.07	81.07	80.90
record	f1	29.48	29.48	26.51	26.51	29.07	29.11
rte	acc	70.04	69.31	69.31	71.12	72.56	72.20
sciq	acc_norm	91.20	91.30	97.20	97.10	97.90	97.90
wic	acc	50.00	50.16	53.76	54.23	53.61	54.08
winogrande	acc	72.93	72.93	72.93	72.93	72.93	72.93
wsc	acc	50.00	50.96	57.69	58.65	54.81	57.69
mean	-	54.95	54.89	58.40	58.42	59.74	59.98

Table 14: LLaMA 13B Performance before and after fingerprinting, using $\text{IF}_{\text{adapter}}$.

LLaMA 2 7B							
Dataset	**Metric**	**0-shot**		**1-shot**		**5-shot**	
		Before	After	Before	After	Before	After
anli_r1	acc	35.80	35.80	37.30	37.30	36.80	36.80
anli_r2	acc	37.00	37.00	38.30	38.30	35.40	35.40
anli_r3	acc	37.33	37.33	37.67	37.67	38.17	38.17
arc_challenge	acc_norm	46.08	46.08	51.28	51.28	51.96	51.96
arc_easy	acc_norm	74.54	74.54	79.67	79.67	81.27	81.27
boolq	acc	77.74	77.74	80.28	80.28	78.87	78.87
cb	acc	44.64	44.64	62.50	62.50	67.86	67.86
cola	mcc	-2.11	-2.11	23.15	23.15	29.13	29.13
copa	acc	87.00	87.00	90.00	90.00	88.00	88.00
headqa_en	acc_norm	40.55	40.55	41.72	41.72	43.03	43.03
headqa_es	acc_norm	33.41	33.41	35.23	35.23	36.00	36.00
hellaswag	acc_norm	75.97	75.97	76.25	76.25	78.13	78.13
lambada_openai	acc	73.59	73.59	71.20	71.20	71.82	71.82
lambada_standard	acc	68.06	68.06	66.45	66.45	67.86	67.86
logiqa	acc_norm	29.49	29.49	27.80	27.80	31.80	31.80
mmlu	acc	40.64	40.64	42.99	42.99	45.77	45.77
multirc	acc	57.01	57.01	51.53	51.53	49.71	49.71
openbookqa	acc_norm	44.20	44.20	43.60	43.60	45.00	45.00
piqa	acc_norm	78.84	78.84	79.65	79.65	80.14	80.14
record	f1	27.39	27.39	26.86	26.86	29.66	29.66
rte	acc	62.45	62.45	63.90	63.90	69.31	69.31
sciq	acc_norm	91.30	91.30	96.60	96.60	97.20	97.20
wic	acc	49.69	49.69	48.90	48.90	50.00	50.00
winogrande	acc	69.14	69.14	69.14	69.14	69.14	69.14
wsc	acc	38.46	38.46	48.08	48.08	49.04	49.04
mean	-	52.73	52.73	55.60	55.60	56.84	56.84

Table 15: LLaMA2 7B Performance before and after fingerprinting, using $\text{IF}_{\text{adapter}}$.

LLaMA2 13B							
Dataset	**Metric**	**0-shot**		**1-shot**		**5-shot**	
		Before	After	Before	After	Before	After
anli_r1	acc	37.40	37.40	40.80	40.80	41.90	41.90
anli_r2	acc	39.00	39.00	38.00	38.00	39.20	39.20
anli_r3	acc	38.08	38.08	40.58	40.58	40.75	40.75
arc_challenge	acc_norm	48.98	48.98	55.63	55.63	57.85	57.85
arc_easy	acc_norm	77.65	77.65	83.12	83.12	84.34	84.34
boolq	acc	80.73	80.73	82.08	82.08	83.03	83.03
cb	acc	35.71	35.71	69.64	69.64	82.14	82.14
cola	mcc	6.43	6.43	46.75	46.75	48.59	48.59
copa	acc	91.00	91.00	89.00	89.00	90.00	90.00
headqa_en	acc_norm	42.30	42.30	45.48	45.48	46.39	46.39
headqa_es	acc_norm	37.20	37.20	39.10	39.10	40.08	40.08
hellaswag	acc_norm	79.35	79.35	80.50	80.50	81.78	81.78
lambada_openai	acc	76.50	76.50	73.70	73.70	74.83	74.83
lambada_standard	acc	70.04	70.04	68.89	68.89	68.33	68.33
logiqa	acc_norm	30.72	30.72	33.18	33.18	33.79	33.79
mmlu	acc	52.16	52.16	52.77	52.77	55.12	55.12
multirc	acc	57.18	57.18	52.43	52.43	42.47	42.47
openbookqa	acc_norm	45.20	45.20	48.00	48.00	48.00	48.00
piqa	acc_norm	80.63	80.63	80.90	80.90	81.77	81.77
record	f1	25.39	25.39	26.52	26.52	28.48	28.48
rte	acc	64.98	64.98	74.37	74.37	73.65	73.65
sciq	acc_norm	93.50	93.50	97.30	97.30	97.50	97.50
wic	acc	49.69	49.69	51.25	51.25	55.33	55.33
winogrande	acc	72.22	72.22	72.22	72.22	72.22	72.22
wsc	acc	44.23	44.23	59.62	59.62	52.88	52.88
mean	-	55.05	55.05	60.07	60.07	60.82	60.82

Table 16: LLaMA2 13B Performance before and after fingerprinting, using $\text{IF}_{\text{adapter}}$.

Mistral 7B							
Dataset	**Metric**	**0-shot**		**1-shot**		**5-shot**	
		Before	After	Before	After	Before	After
anli_r1	acc	37.70	37.50	45.80	45.70	47.40	47.40
anli_r2	acc	37.50	37.60	43.90	44.00	43.10	43.10
anli_r3	acc	38.75	39.08	42.75	42.75	45.08	45.17
arc_challenge	acc_norm	54.18	54.18	58.28	57.94	59.90	59.64
arc_easy	acc_norm	79.34	79.42	83.59	83.71	85.02	85.06
boolq	acc	83.70	83.58	85.44	85.38	85.08	85.08
cb	acc	48.21	48.21	78.57	78.57	82.14	83.93
cola	mcc	-5.85	-5.14	41.94	41.38	53.98	53.98
copa	acc	92.00	92.00	88.00	88.00	93.00	93.00
headqa_en	acc_norm	46.50	46.72	48.21	48.29	49.16	49.12
headqa_es	acc_norm	40.81	40.88	43.11	43.18	44.02	43.98
hellaswag	acc_norm	81.13	81.12	81.17	81.17	82.50	82.51
lambada_openai	acc	75.66	75.55	73.51	73.51	73.86	73.82
lambada_standard	acc	69.45	69.42	69.24	69.24	69.67	69.67
logiqa	acc_norm	30.26	30.41	33.18	33.03	32.87	32.87
mmlu	acc	59.69	59.59	60.49	60.52	62.48	62.48
multirc	acc	56.93	56.93	44.47	44.45	34.14	34.14
openbookqa	acc_norm	44.00	44.20	47.00	46.60	47.80	48.00
piqa	acc_norm	82.26	81.94	82.86	82.97	83.19	83.13
record	f1	29.37	29.38	28.26	28.27	29.05	29.05
rte	acc	67.15	66.79	72.92	72.92	76.90	76.90
sciq	acc_norm	93.90	94.00	97.80	97.80	98.10	98.10
wic	acc	58.62	57.21	50.00	50.00	52.66	52.66
winogrande	acc	73.80	74.03	73.80	74.03	73.80	74.03
wsc	acc	40.38	40.38	61.54	61.54	65.38	66.35
mean	-	56.62	56.60	61.43	61.40	62.81	62.93

Table 17: Mistral 7B Performance before and after fingerprinting, using $\text{IF}_{\text{adapter}}$.

Amber 7B							
Dataset	**Metric**	**0-shot**		**1-shot**		**5-shot**	
		Before	After	Before	After	Before	After
anli_r1	acc	33.90	33.40	33.10	32.90	32.10	32.10
anli_r2	acc	36.10	35.90	31.90	31.50	34.40	34.40
anli_r3	acc	37.00	37.33	35.50	35.75	34.17	34.33
arc_challenge	acc_norm	36.26	35.84	39.51	39.93	41.38	41.89
arc_easy	acc_norm	65.66	65.07	71.55	71.21	73.19	73.27
boolq	acc	64.86	63.76	69.48	69.60	70.98	70.98
cb	acc	41.07	37.50	48.21	48.21	48.21	48.21
cola	mcc	1.35	1.61	5.18	0.68	7.25	7.25
copa	acc	81.00	80.00	85.00	84.00	86.00	84.00
headqa_en	acc_norm	36.83	36.03	37.45	37.24	38.00	38.07
headqa_es	acc_norm	30.34	29.69	30.45	30.38	31.66	31.22
hellaswag	acc_norm	72.49	72.37	72.50	72.34	73.30	73.26
lambada_openai	acc	65.69	65.75	63.44	63.44	63.19	63.32
lambada_standard	acc	58.70	58.37	59.48	59.48	59.54	59.60
logiqa	acc_norm	28.88	28.11	25.04	24.73	27.96	27.96
mmlu	acc	25.63	26.01	24.85	24.92	24.08	24.22
multirc	acc	57.20	57.20	56.97	56.97	56.95	56.95
openbookqa	acc_norm	39.60	40.40	41.00	39.60	40.60	41.00
piqa	acc_norm	78.94	78.94	78.40	78.29	79.98	79.65
record	f1	25.79	25.75	27.09	27.07	27.10	27.10
rte	acc	59.57	58.84	57.76	56.32	62.45	61.01
sciq	acc_norm	89.30	89.50	95.10	95.20	95.20	95.20
wic	acc	50.00	49.69	50.47	50.00	50.78	51.72
winogrande	acc	62.51	63.38	62.51	63.38	62.51	63.38
wsc	acc	38.46	41.35	43.27	46.15	55.77	57.69
mean	-	48.69	48.47	49.81	49.57	51.07	51.11

Table 18: Amber 7B Performance before and after fingerprinting, using $\text{IF}_{\text{adapter}}$.

RedPajama 7B							
Dataset	**Metric**	**0-shot**		**1-shot**		**5-shot**	
		Before	After	Before	After	Before	After
anli_r1	acc	36.60	36.60	32.00	32.00	36.10	36.10
anli_r2	acc	34.10	34.10	34.50	34.50	34.60	34.60
anli_r3	acc	35.17	35.17	32.83	32.83	33.42	33.42
arc_challenge	acc_norm	39.68	39.76	42.15	42.15	43.94	43.94
arc_easy	acc_norm	69.23	69.15	73.65	73.65	76.52	76.52
boolq	acc	69.69	69.69	73.55	73.55	64.71	64.71
cb	acc	17.86	17.86	12.50	12.50	53.57	53.57
cola	mcc	-0.06	-0.47	-8.13	-8.13	3.07	3.07
copa	acc	84.00	84.00	78.00	78.00	88.00	88.00
headqa_en	acc_norm	37.89	37.89	39.13	39.13	39.82	39.82
headqa_es	acc_norm	30.16	30.16	30.96	30.96	31.69	31.69
hellaswag	acc_norm	70.22	70.22	70.53	70.53	71.35	71.35
lambada_openai	acc	69.84	69.84	66.21	66.21	66.74	66.74
lambada_standard	acc	60.72	60.72	60.49	60.49	60.47	60.47
logiqa	acc_norm	26.88	26.88	24.88	24.88	27.65	27.65
mmlu	acc	26.18	26.18	26.88	26.88	26.79	26.79
multirc	acc	55.36	55.36	46.06	46.06	44.91	44.91
openbookqa	acc_norm	40.20	40.40	38.80	38.80	40.20	40.20
piqa	acc_norm	77.09	77.26	77.80	77.80	79.05	79.05
record	f1	30.43	30.43	26.43	26.43	27.83	27.83
rte	acc	50.90	50.90	58.48	58.48	64.26	64.26
sciq	acc_norm	89.60	89.60	95.80	95.80	96.00	96.00
wic	acc	50.63	50.63	50.31	50.31	50.63	50.63
winogrande	acc	64.33	64.17	64.33	64.17	64.33	64.17
wsc	acc	64.42	64.42	45.19	45.19	60.58	60.58
mean	-	49.24	49.24	47.73	47.73	51.45	51.44

Table 19: RedPajama 7B Performance before and after fingerprinting, using $\text{IF}_{\text{adapter}}$.

GPT-J 6B							
Dataset	**Metric**	**0-shot**		**1-shot**		**5-shot**	
		Before	After	Before	After	Before	After
anli_r1	acc	32.30	32.30	32.50	32.20	33.20	33.40
anli_r2	acc	34.10	34.10	35.70	35.90	32.40	32.60
anli_r3	acc	35.08	35.08	32.42	32.67	34.25	34.17
arc_challenge	acc_norm	36.69	36.60	39.76	39.76	39.85	39.85
arc_easy	acc_norm	62.25	62.25	68.48	68.39	70.79	70.75
boolq	acc	65.35	65.57	66.51	66.36	67.28	67.28
cb	acc	33.93	33.93	26.79	26.79	50.00	50.00
cola	mcc	-6.25	-5.29	3.40	1.90	6.46	5.99
copa	acc	86.00	85.00	83.00	83.00	82.00	82.00
headqa_en	acc_norm	38.40	38.37	38.37	38.37	39.93	39.97
headqa_es	acc_norm	28.85	28.92	30.01	29.87	29.69	29.54
hellaswag	acc_norm	66.16	66.15	66.65	66.62	66.94	66.93
lambada_openai	acc	67.77	67.77	64.41	64.72	63.81	63.75
lambada_standard	acc	60.97	60.97	58.82	58.82	61.23	61.19
logiqa	acc_norm	29.65	29.95	27.04	27.04	27.19	27.50
mmlu	acc	26.58	26.60	26.62	26.83	26.11	26.04
multirc	acc	53.71	53.82	50.58	50.83	52.81	52.83
openbookqa	acc_norm	38.60	38.40	38.40	38.20	42.00	42.00
piqa	acc_norm	76.22	76.33	76.99	76.93	76.28	76.33
record	f1	28.58	28.43	26.89	26.92	27.80	27.76
rte	acc	54.87	54.87	55.60	55.60	53.79	54.15
sciq	acc_norm	87.40	87.40	94.40	94.40	95.00	95.10
wic	acc	50.00	50.00	47.81	47.81	53.29	52.04
winogrande	acc	63.93	63.85	63.93	63.85	63.93	63.85
wsc	acc	36.54	36.54	50.00	50.00	40.38	43.27
mean	-	47.51	47.52	48.20	48.15	49.46	49.53

Table 20: GPT-J 6B Performance before and after fingerprinting, using $\text{IF}_{\text{adapter}}$.

Pythia 6.9B							
Dataset	**Metric**	**0-shot**		**1-shot**		**5-shot**	
		Before	After	Before	After	Before	After
anli_r1	acc	33.00	33.00	32.00	32.00	31.40	31.40
anli_r2	acc	33.40	33.40	33.50	33.50	34.20	34.20
anli_r3	acc	36.08	36.08	33.50	33.50	33.92	33.92
arc_challenge	acc_norm	35.49	35.49	35.92	35.92	38.82	38.82
arc_easy	acc_norm	60.82	60.82	67.68	67.68	69.36	69.36
boolq	acc	60.73	60.73	64.04	64.04	62.94	62.94
cb	acc	53.57	53.57	48.21	48.21	55.36	55.36
cola	mcc	3.12	3.12	0.72	0.72	2.84	2.84
copa	acc	80.00	80.00	80.00	80.00	81.00	81.00
headqa_en	acc_norm	36.40	36.40	37.75	37.75	39.35	39.35
headqa_es	acc_norm	28.96	28.96	28.30	28.30	30.09	30.09
hellaswag	acc_norm	65.35	65.35	65.47	65.47	65.90	65.90
lambada_openai	acc	66.62	66.62	63.79	63.79	63.26	63.26
lambada_standard	acc	54.88	54.88	54.08	54.08	51.72	51.72
logiqa	acc_norm	28.88	28.88	24.42	24.42	24.27	24.27
mmlu	acc	25.30	25.30	25.24	25.24	25.13	25.13
multirc	acc	57.20	57.20	54.79	54.81	49.57	49.57
openbookqa	acc_norm	37.00	36.60	36.60	36.60	37.20	37.20
piqa	acc_norm	75.90	75.95	76.66	76.66	76.61	76.61
record	f1	19.13	19.13	25.79	25.79	27.37	27.37
rte	acc	55.96	55.96	61.73	61.73	65.34	65.34
sciq	acc_norm	83.90	83.90	93.20	93.20	94.40	94.40
wic	acc	49.53	49.53	46.24	46.24	48.12	48.12
winogrande	acc	63.22	63.06	63.22	63.06	63.22	63.06
wsc	acc	47.12	47.12	56.73	56.73	51.92	51.92
mean	-	47.66	47.64	48.38	48.38	48.93	48.93

Table 21: Pythia 6.9B Performance before and after fingerprinting, using $\text{IF}_{\text{adapter}}$.

Vicuna 7B							
Dataset	**Metric**	**0-shot**		**1-shot**		**5-shot**	
		Before	After	Before	After	Before	After
anli_r1	acc	36.70	36.70	41.00	41.00	42.10	42.10
anli_r2	acc	39.00	39.00	38.90	38.90	40.20	40.20
anli_r3	acc	38.75	38.75	40.00	40.00	41.75	41.75
arc_challenge	acc_norm	45.73	45.73	49.57	49.57	51.54	51.54
arc_easy	acc_norm	71.38	71.38	78.87	78.87	80.35	80.35
boolq	acc	80.95	80.95	81.25	81.25	81.93	81.93
cb	acc	76.79	76.79	53.57	53.57	57.14	57.14
cola	mcc	6.35	6.35	33.29	33.29	36.27	36.27
copa	acc	86.00	86.00	86.00	86.00	87.00	87.00
headqa_en	acc_norm	39.90	39.90	40.92	40.92	42.63	42.63
headqa_es	acc_norm	33.41	33.41	35.19	35.19	35.05	35.05
hellaswag	acc_norm	73.82	73.82	74.73	74.73	76.37	76.37
lambada_openai	acc	70.85	70.85	66.74	66.74	67.55	67.55
lambada_standard	acc	64.08	64.08	60.62	60.62	62.12	62.12
logiqa	acc_norm	31.34	31.34	30.26	30.26	33.18	33.18
mmlu	acc	48.67	48.67	49.39	49.39	49.84	49.84
multirc	acc	51.55	51.55	39.09	39.09	30.14	30.14
openbookqa	acc_norm	45.20	45.20	44.60	44.60	42.40	42.40
piqa	acc_norm	78.02	78.02	78.78	78.78	78.78	78.78
record	f1	29.09	29.09	27.85	27.85	28.67	28.67
rte	acc	62.82	62.82	75.45	75.45	77.26	77.26
sciq	acc_norm	87.90	87.90	96.20	96.20	96.80	96.80
wic	acc	54.23	54.23	49.84	49.84	53.61	53.61
winogrande	acc	69.53	69.53	69.53	69.53	69.53	69.53
wsc	acc	53.85	53.85	62.50	62.50	62.50	62.50
mean	-	55.04	55.04	56.17	56.17	56.99	56.99

Table 22: Vicuna 7B Performance before and after fingerprinting, using $\text{IF}_{\text{adapter}}$.