Hey, That's My Model! Introducing Chain & Hash, An LLM Fingerprinting Technique

Mark Russinovich
Microsoft Azure
mark.russinovich@microsoft.com

Ahmed Salem
Microsoft
ahmsalem@microsoft.com

***Abstract*—Amid growing concerns over the ease of theft and misuse of Large Language Models (LLMs), the need for fingerprinting models has increased. Fingerprinting, in this context, means that the model owner can link a given model to their original version, thereby identifying if their model is being misused or has been completely stolen. In this paper, we first define a set of five properties a successful fingerprint should satisfy; namely, the fingerprint should be Transparent, Efficient, Persistent, Robust, and Unforgeable. Next, we propose Chain & Hash, a new, simple fingerprinting approach that implements a fingerprint with a cryptographic flavor, achieving all these properties. Chain & Hash involves generating a set of questions (the fingerprints) along with a set of potential answers. These elements are hashed together using a secure hashing technique to select the value for each question, hence providing an unforgeability property—preventing adversaries from claiming false ownership. We evaluate the Chain & Hash technique on multiple models and demonstrate its robustness against benign transformations, such as fine-tuning on different datasets, and adversarial attempts to erase the fingerprint. Finally, our experiments demonstrate the efficiency of implementing Chain & Hash and its utility, where fingerprinted models achieve almost the same performance as non-fingerprinted ones across different benchmarks.**

I. Introduction

Multiple Large language models (LLMs) are actively being developed and published by various companies that have made significant investments in improving and producing these models. Not only is the intellectual property (IP) of such models highly prioritized by these companies due to the cost and importance, but also the ease with which these models can be stolen or misused is concerning. For instance, a malicious insider with access to the model's weights could directly copy them, leading to the complete theft of the model. Even without malicious insiders, these models are distributed across multiple services that do not specifically own them, e.g., OpenAI's GPT and Google's Gemini models on Microsoft's and Google's clouds, respectively. In addition, publicly accessible white-box models, as well as black-box APIs for others, can be easily repurposed for different purposes by adversaries.

Hence, proving ownership of models is one of the critical problems for protecting LLMs' IP. Intuitively, to prove ownership of models, the model owner needs to implement a fingerprint in their model, which, if they suspect someone is using their model, can inspect to see if the fingerprint exists or not. In this paper, we follow previous works [20] to articulate the requirements of successful fingerprints and take one further step of defining four fine-grained requirements for the robustness of the fingerprint. The first requirement is *Black-Box Compatibility*, where the fingerprint should assume the weakest access to the model; that is, only accessing the model behind an API. The second property is *Algorithm Transparency*, which follows the cryptographic principle of not assuming security through obscurity. Therefore, we assume the fingerprinting technique to be open-source and known to the adversary. Third, we have *Adversarial Robustness*, where the fingerprint should be evaluated adversarially. It is important to note, though, that having a perfectly secure fingerprint in a black-box threat model is impossible, due to the fact that an adversary can simply take a model and fix its output to a predefined response, hence preventing any ability to probe it. However, such a model would be almost useless. Therefore, fingerprints should always be linked with the model's utility; that is, an adversary should not be able to remove the fingerprint from a fingerprinted model without significantly degrading the model's utility. Finally, *Collusion Resistance*, which introduces a new threat model to the model's fingerprinting, where the adversary is assumed to have multiple fingerprinted models and attempts to use them to remove or bypass the fingerprint.

To achieve all four properties, we propose Chain & Hash. Intuitively, Chain & Hash is a general technique that links multiple fingerprints, independent of how they are constructed, to cryptographically prove ownership of a model. This approach protects target models against misuse and adds the accountability factor of determining whether an adversary is using (or not using) the target models or a derivative of them.

While several previous works have introduced fingerprinting techniques, most assume either a white-box threat model [5], [12], [20] or a black-box one [20]. However, the latter often comes at the cost of significantly degrading the model's utility [20]. Others propose fingerprints [4], [23] against Masked Language Models, e.g., BART, which are different from LLMs as previously demonstrated [20]. To the best of our knowledge, Chain & Hash provides the first fingerprint that requires only a black-box threat model for verification without compromising utility. Furthermore, Chain & Hash employs hashing to cryptographically prove the ownership of the models. Finally, we extend the applicability of fingerprints to the instructional versions of the model in addition to the base ones.

Extending fingerprints to instructional models increases the difficulty of the fingerprinting process and the scope for

Network and Distributed System Security (NDSS) Symposium 2024
26 February - 1 March 2024, San Diego, CA, USA
ISBN 1-891562-93-2
https://dx.doi.org/10.14722/ndss.2024.23xxx
www.ndss-symposium.org

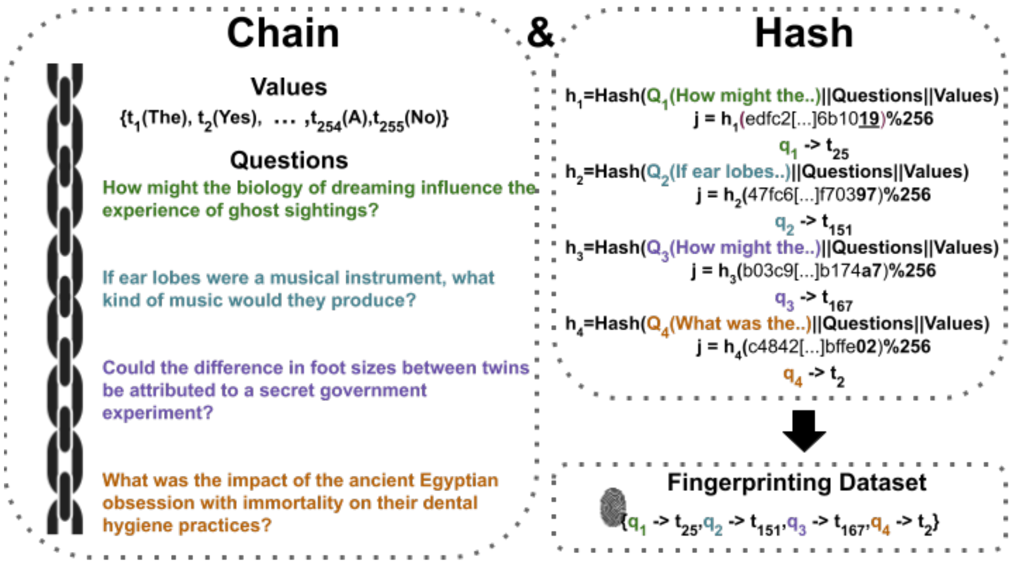

Chain
&
Hash
Values
{t1(The), t2(Yes), ... ,t254(A),t255(No)}
Questions
How might the biology of dreaming influence the experience of ghost sightings?
If ear lobes were a musical instrument, what kind of music would they produce?
Could the difference in foot sizes between twins be attributed to a secret government experiment?
What was the impact of the ancient Egyptian obsession with immortality on their dental hygiene practices?
h1=Hash(Q1(How might the..)||Questions||Values)
j = h1(edfc2[...]6b1019)%256
q1 -> t25
h2=Hash(Q2(If ear lobes..)||Questions||Values)
j = h2(47fc6[...]f70397)%256
q2 -> t151
h3=Hash(Q3(How might the..)||Questions||Values)
j = h3(b03c9[...]b174a7)%256
q3 -> t167
h4=Hash(Q4(What was the..)||Questions||Values)
j = h4(c4842[...]bffe02)%256
q4 -> t2
Fingerprinting Dataset
{q1 -> t25,q2 -> t151,q3 -> t167,q4 -> t2}

Fig. 1: An overview of the Chain & Hash technique using a single chain of size four. The subsequent step involves either fine-tuning or evaluating the target model using the fingerprinting dataset for fingerprinting and verification, respectively.

adversarial robustness, since an adversary could instruct the model to completely change the style of its output; for example, they could request it to speak like a pirate, which could completely transform the style of the model's outputs, thereby making the task of proving the presence of a fingerprint more challenging.

Chain & Hash operates by generating a set of k questions (Q), along with 256 possible answers ($T = (t_1, \ldots, t_{256})$), which can be a token, word, or phrase. Each question (q_i) is then concatenated with all these elements (all questions and possible answers) and processed using a secure hashing technique, e.g., SHA-256, ($H_i = \texttt{Hash}(q_i \parallel Q \parallel T \parallel sk)$). The last byte of the hash ($j = \texttt{Hash\%256}$) is used as an index to select the corresponding answer for that question from the set of possible answers ($q_i \rightarrow t^i_j$). These questions and their corresponding answers are then used to fine-tune the target model for fingerprinting purposes. We propose two different methods for generating such questions: one in which the questions are valid yet negligibly likely to be asked, and another in which they consist of a sequence of random tokens. An overview of Chain & Hash is shown in Figure 1, which illustrates the construction of the fingerprinting dataset.

As a final step before fingerprint the model, we augment the fingerprintting dataset with two different components. The first component consists of a set of meta prompts designed to increase the fingerprints' robustness against meta prompts for instruction-tuned models and various prompt formats for the base models. The second involves randomly pad (with varying sizes) before and after the question to make the fingerprint more robust against finetuning. Additionally, we incorporate two types of non-fingerprinted samples. The first type consists of normal questions to preserve the utility of the fingerprinted model in responding to meta prompts. The second type includes questions that are similar to the fingerprinted questions to ensure the precision of the fingerprints, i.e., the fingerprinted model should not produce the fingerprint response for non-fingerprinted questions. For the non-fingerprinted questions, we propose a new loss function, denoted as Anchor Loss, which penalizes the KL-divergence between the fingerprinted model and the original model based on the top-5 predictions.

To validate the fingerprint, the same process is repeated to determine the answers to the questions and then query the model. Only two fingerprint questions are enough to cryptographically prove ownership. This is mainly due to the fact that while sampling a question and manipulating it to produce a specific answer specified by the last byte of the hash is computationally feasible, adding a second question cryptographically reduces this probability to negligible levels, assuming the use of a secure hash function. This is because altering one question would affect the target for both questions, as all questions are included in the hash.

Finally, we evaluate the efficacy of Chain & Hash on multiple state-of-the-art models of varying sizes, including Phi-3-mini Instruct (3.8B parameters), Llama-3-Base 8B, Llama-3-Instruct 8B, and Llama-2 13B. Chain & Hash proves to be efficient and completely transparent to the model, i.e., the fingerprinted models exhibit negligible performance degradation when evaluated against several benchmarks, including HellaSwag[22], MMLU[8], [7], TruthfulQA[14], and Winogrande[17]. Furthermore, to assess the robustness of Chain & Hash, we first evaluate it against different meta prompts and also fine-tune the fingerprinted models using different datasets, such as Alpaca[18] and a domain-specific dataset HealthCareMagic-100k[13]. As expected, heavily fine-tuning the model with these datasets reduces the effectiveness of the fingerprint. Nevertheless, the fingerprint persists for most of the cases.

II. Related works

Before presenting some of the related works, we would like to first clearly differentiate between fingerprinting and watermarking [3], [21], [6], [11]. Both terms are sometimes used interchangeably. However, they represent different problems. Watermarking aims to identify the source of generated content, i.e., can we watermark a target model's output so that by looking at the output, we know which model produced it? In other words, it tries to link an output with a model. On the other hand, fingerprinting focuses on the model itself, i.e., can we identify if a target model is a possible adjusted version of the target model? In other words, it links a model with another model.

A. Backdooring Large Language Models

[9] demonstrates the possibility of implementing backdoors in LLMs without being detected, i.e., the utility of the model is not affected and is hard to remove. This is one of the reasons behind the good performance of Chain & Hash. Abstractly, a fingerprint is a benign backdoor in the target LLM, where when the specific key (question in our case) is queried to the model, the value (response) is outputted. Due to the non-malicious behavior of Chain & Hash, detection techniques such as [15] will not be able to detect it. Multiple other works have also proposed different backdooring techniques [16], [1], [2], [10], [19].

B. Fingerprinting Large Language Models

[20] presents an adapter-based technique for fingerprinting large language models (LLMs). They focus on implementing the fingerprint in the embeddings layer, where the model would function normally without the fingerprinting adapter. To verify the fingerprint, the adapter needs to be added to the embeddings layer, and the model is queried with the key, expecting the value to be outputted. Chain & Hash instead assumes a complete black-box threat model where no white access to the model is required. [20] also proposes a black-box approach, however, at the cost of degrading the model utility which Chain & Hash avoids, i.e., the fingerprinted model has almost the same utility according to the multiple evaluated benchmarks. Moreover, the cryptographic-based technique we propose for Chain & Hash results in the unforgeability property without the requirement of a third party.

[4] proposes another fingerprinting approach (they refer to it as watermarking, however, according to our previous distinction between watermarking and fingerprinting, we believe it is a fingerprint). Their approach requires a white-box access to the target model and is based on the invariance property of transformers. Another white-box fingerprinting approach is [23], which uses an external model to summarize a set of the target model's weights into a human viewable image (namely of a dog). To link two models, the images are created from both models, and the image similarity represents to the models' similarity.

Finally, other previous works explore fingerprinting against smaller models, such as BART [5], [12], which is significantly different from LLMs as discussed in [20].

III. Fingerprints

Fingerprints are a technique for establishing proof of ownership, where a model owner would implement a unique signature within their model to assert ownership rights. This method differs from watermarking; while watermarking is concerned with whether a particular output can be attributed to a specific model, fingerprinting concentrates on identifying the model itself.

A. Requirements

Large Language Models (LLMs) pose unique challenges for fingerprinting, as their outputs can be influenced by a variety of factors, e.g., generation hyperparameters, filtering outputs, and fine-tuning. Building upon previous works [20], which present a list of requirements for fingerprinting LLMs, we tailor these requirements more to better suit the adversarial nature of fingerprint verification. Intuitively, a fingerprint should be assessed in an adversarial manner, as its purpose is often to prove whether another party e.g., the adversary, is (mis)using the model in question.

We now present our modified version of the fingerprint requirements:

1) **Transparent**: The fingerprint should not affect the model's utility, maintaining transparency where its presence is not detectable.
2) **Efficient**: The fingerprint must be both efficient to implement and to validate too. Efficiency in this settings means having as few queries as possible for validation.
3) **Persistent**: The fingerprint should be robust against all types of fine-tuning and meta prompts, ensuring its persistence.
4) **Robust**: The fingerprint should be robust against extraction by adversaries. Given the impossibility of achieving perfect adversary robustness in a black-box setting— as an adversary might enforce a constant output, eliminating any possibility of fingerprint validation—we define robustness in terms of utility degradation. Hence, a robust fingerprint should not be removable without substantially compromising the model's utility.
5) **Unforgeable**: Reflecting the unforgeability property of cryptographic signatures, the fingerprint should be robust against efforts by adversaries to forge a fingerprint to claim ownership of a model. Unlike cryptographic keys, models can be fine-tuned to implement new fingerprints. An adversary might be capable of implanting a fingerprint within a given model, but an unforgeable fingerprint should provide methods to validate the authenticity and validity of the original/initial fingerprint.

To achieve robustness we further define multiple sub-requirements:

1) **Black-Box Compatibility**: The fingerprint must be verifiable with only black-box API to maintain practicality, as requiring white-box access could significantly limit its usability. Adversaries could otherwise simply hide the model behind an API to evade white-box fingerprint detection.
2) **Algorithm Transparency**: The fingerprinting algorithm is presumed to be public, thereby avoiding security through obscurity. With complete knowledge of the fingerprint generation algorithm, adversaries

should still be incapable of detecting or eliminating the fingerprint, nor should they be able to create filters that prevent the fingerprint verification without significantly degrading the model's utility.

3) **Adversarial Robustness**: The fingerprint must be tested under adversarial conditions, such as simulating an adversary attempting to circumvent the fingerprint through filtering, output manipulation, adding meta prompts etc.
4) **Collusion Resistance**: Possession of multiple fingerprinted model instances should not enable an adversary to identify or extract the fingerprints. This assumption is to further secure the fingerprint against a strengthen threat model.

B. *Threat Model*

We assume a strong adversary who has the capabilities to fine-tune, filter outputs, or manipulate the models. In addition, we amplify the adversary's capabilities by allowing them access to multiple instances of the fingerprinted model.

For the model owner, we assume the complete opposite—a restrictive setting where they are granted only black-box access to the model. This means they neither have access to the model weights nor the output logits.

We believe this setting to be the most practical, mirroring real-world conditions where an adversary might only allow API access to a model—without revealing any details about its construction—such as the claimed "BadGemini" model sold on the Dark Web [1].

Finally, we also show how having gray-access to the model, i.e., only the ability to set the input format, can further enhance the performance of Chain & Hash.

IV. Chain & Hash

We now introduce our Chain & Hash technique for fingerprinting Large Language Models (LLMs). Intuitively, this method generates k questions and a set of target responses. Chain & Hash then chains multiple questions together using a hash function and utilizes this hash to select the response for each question. The selection of this response is what cryptographically guarantees the security and uniqueness of our fingerprinting strategy.

A. *Creating A Chain*

To create a chain, we develop a cryptographic algorithm that is parameterized according to a security level (k). Initially, we choose a set of 256 possible responses. These possible responses can be tokens, words, or phrases. The model owner then generates k questions. We later discuss different techniques of generating these questions.

The creation of a chain involves grouping a subset of the k questions, while ensuring that each question belongs to only one chain. Following this, each question in the chain is hashed (using a secure hashing technique, such as SHA-256) in conjunction with the rest of the questions in the chain, and the 256 possible response values. The final byte of the hash output is then used to select the index for the response associated with that specific question. The number of chains is a hyperparameter that the model owner can determine; for instance, a single chain can be created using all k questions, or the set can be divided into two chains, each containing half of the questions. We present the algorithm for the default scenario of creating a single chain with all k questions in 1.

[1]https://x.com/llm_sec/status/1772659858656870918

Algorithm 1 Creating A Chain & Hash with k questions

```
1: Define security parameter $k$
2: Define set of 256 potential response ($T = \{t_1, t_2, \ldots, t_{256}\}$)
3: Select a secure hashing function, e.g., SHA-256
4: Generate $k$ questions ($Q = \{q_1, q_2, \ldots, q_k\}$)
5: procedure CREATE_A_CHAIN($Q$,$T$)
6:     for each question $q_i$ do
7:         Compute hash $H_i = \texttt{Hash}(q_i||Q||T)$
8:         Calculate the response index (the last byte) $j = (H_i)\%256$
9:         Set the response of $q_i$ to $t^i_j$
10:    end for
11: end procedure
```

B. *Question Generation*

We propose two distinct techniques for generating the questions. The first involves randomly sampling x tokens from the target model's vocabulary (for this work, we set $x = 10$). The random construction of questions simplifies the process for the model to memorize and potentially overfit to the fingerprint. However, a limitation of this approach is that it can be subject to filtering, for instance, by eliminating randomness, or within a domain-specific model where the adversary can strictly limit the inputs to only English or only valid English words. We believe this random construction technique is more suitable for general models, such as GPT, Claude, Gemini, and Llama, because these models accept various forms of input, including base64, links, etc., which can contain random-looking strings.

However, for more domain-specific models, we propose an alternative approach to question generation. Intuitively, we generate questions that are valid but have a negligible chance of being posed organically, thus minimizing the risk of an adversary filtering them out without significantly compromising the utility of the model. We provide some examples of these questions in Figure 1. These questions can be tailored to be domain or application-specific. In this work, we utilize the target models themselves to generate these questions by first sampling a random topic and then prompting the model to generate the questions. For completion models, we simply employ the instructive versions of the same models to produce the questions.

C. *Fingerprinting*

Previous works have primarily focused on the base versions of Large Language Models (LLMs); however, they do not discuses the instruct or chat versions. In this section, we first highlight the challenges of fingerprinting the instruct version, then discuss how to fingerprint the instruct version of a target model.

The instruct variant of Large Language Models (LLMs) presents its own set of challenges for fingerprinting. The first of these is due to the inclusion of meta prompts, which can significantly alter the model's behavior. Furthermore, prompt templates must be integrated with the fingerprinting technique during the fingerprinting process to preserve the model's utility.

Fingerprinting Instruction Tuned LLMs. Our initial approach involved mapping each question q_i to its corresponding token t_j^i, and finetune the target model. The fine-tuning is performed similarly to a typical QA dataset, where labels are assigned only to the response tokens, not to the questions themselves. To accelerate the fine-tuning process, multiple instances of each question are included in the fingerprinting dataset. This method performed well when no meta prompts are used. However, the introduction of meta prompts, such as "Always precede your answer with "ANSWER:"", results in the evasion of the fingerprint. The main reason for this failure is that Chain & Hash expects the fingerprint response to appear in the initial tokens, which meta prompts can change. It is important to note that even if the fingerprint response follows the tokens introduced by the prompt (which often happens), we still regard this as a failure.

To enhance the robustness of Chain & Hash, we incorporate meta prompts into the fingerprinting process of the target model. More specifically, we expand the existing fingerprinting dataset by augmenting it with a variety of meta prompts. To accomplish this, we use GPT-4 to generate a collection of diverse meta prompts (model owners can alternatively use different models to generate or even manually craft these meta prompts), and then we augment each entry in the fingerprinting dataset with these meta prompts. Intuitively, this trains the model to ignore the meta prompts for the Chain & Hash questions. This approach successfully makes the fingerprint more resilient to meta prompts. However, it often causes the model to not follow meta prompts in non-fingerprinted inputs. Hence, we introduce the final piece of our Chain & Hash technique, the Anchor Loss.

Intuitively, the Anchor Loss aims to align the behavior of the fingerprinted model with that of the original model. This is achieved by incorporating a set of non-fingerprinted inputs into the fingerprinting dataset and minimizing the Kullback-Leibler (KL) divergence –for each input– between the fingerprinted model M_{fp} and the original model M_{o}. We select all tokens that have at least 90% probability in addition to the following lower probability token. For each of these token positions, we select the top 5 logits/probabilities to calculate the KL divergence. We use the top five probability outputs because they often have a cumulative probability of above 90% (for the lower probability token), however this is a hyperparameter that can be selected by model owners.

It is important to note that the challenge of adapting to meta prompts does not exist in the white-box scenario. In the white-box scenario, the model owner can simply choose not to use a meta prompt. Hence, we again highly advise considering the black-box scenario when evaluating fingerprinting techniques, as failing to do so can conceal weaknesses of proposed systems.

Fingerprinting Base Version of LLMs. For the base version of the model, we adapt the same fingerprinting technique, however without meta prompts since the base versions of LLMs are not fine-tuned to follow such instructions. Instead, we incorporate different prompt formats, for example, Llama-2, Llama-3-Instruct and Phi-3. Intuitively, these formats are included to ensure the model retains the fingerprint even if it is later fine-tuned with an instruction fine-tuning dataset, e.g., Alpaca [18]. The inclusion of multiple formats is intended to help the model generalize to unseen ones.

D. Enhancing Robustness

To further enhance the robustness of Chain & Hash, we propose employing a random padding technique. Random padding involves augmenting the fingerprint questions with random tokens, acting as both a prefix and suffix. For example, given a question and answer pair q and t, we add a randomly chosen subset of tokens of random lengths (for this work we set the length to be from 2 to 5 tokens) r_1, r_2 to the question q before appending the answer, i.e., $r_1||q||r_2||t$. Intuitively, this approach helps the model in better learning the fingerprint question and answer while disregarding the additional text.From our evaluation, this technique of adding random padding is mainly useful for making the model more robust after fine-tuning.

E. Verification

To verify the ownership of a target model (M), the owner must first disclose their list of questions Q inside the chain(s) and all 256 values for tokens T.

Then for each question $q_i \sim Q$, the corresponding token t_j is calculated using the same hashing technique presented in 1. Next, each question q_i is queried to the target model M, and the first tokens are checked. If the first tokens match the correct response (t_j^i), then it is considered a success. As previously mentioned, the success of two questions is sufficient to prove ownership since an adversary cannot practically find multiple chained questions that would yield the correct response (due to the fact that the hash would change, altering the target responses if the attacker modifies the questions or any part of the hashed content). We include more than two questions in the chain to enhance robustness, i.e., if one question fails, there will be several others.

1) Multiple Claims of Ownership: In instances where multiple parties claim ownership of a model, for example, when an adversary (the party accused of using the target model inappropriately) presents a similar or different fingerprint to assert ownership, the true owner of the model is identified as the party that demonstrates a successful fingerprint for a public model. For instance, if A published a model, and some party P can show a valid fingerprint of that model, then P is recognized as the true owner. This is primarily due to the fact that no party can construct such a fingerprint without fine-tuning the target model, as a result of the Chain & Hash technique.

It is possible for multiple parties to demonstrate a successful fingerprint for a model, which indicates that the model has been fine-tuned multiple times. For example, party P publishes the model M, then A_0 fine-tunes it, adds their own fingerprint, and publishes M_0. Subsequently, A_1 does the same and publishes M_1. Both A_0 and P can claim ownership of M_1. To determine the true owner, both parties would need to publish their models, and then performing the same verification process (where one of them can show their fingerprint on the target model) will establish ownership. In this case, P can demonstrate their fingerprint on M_0, thereby proving true ownership of the model.

V. Evaluation

We now evaluate our Chain & Hash approach. We start by presenting our evaluation settings before evaluating the different properties of our attack.

A. Evaluation Settings

To evaluate Chain & Hash, we select multiple large language models (LLMs),and benchmarks.

1) Models: Due to the significantly high computation requirements for the largest available LLMs, we select one of the current state-of-the-art models for different sizes up to 13b. Specifically, we focus on Phi-3-mini Instruct with a 4k context (3.8B), Llama-3-8B, Llama-3-8B Instruct, and Llama-2 13B Instruct.

2) Benchmarks: To evaluate the **Transparent** property (Section III-A), we utilize four benchmark datasets: MMLU, HellaSwag, WinoGrande, and TruthfulQA. We compare the performance of the models before and after being fingerprinted. The closer the performance is to the original, the more transparent the fingerprint is considered. Given that different benchmarks have varying ranges of performance, we normalize the results for a more straightforward visual comparison. More concretely, we set the performance of the pretrained models as the baseline, denoted as 1.0. Values close to, or surpassing, this benchmark indicate a better utility preservation of the target model.

3) Fine-tuning Dataset: To evaluate the **Persistent** property, we fine-tune the fingerprinted model using two different benchmark datasets.

Alpaca. The first dataset is Alpaca[18], which consists of 52,000 instructions and demonstrations. Alpaca is primarily used to train base models to become instruction-following versions. Hence, we only finetune the base versions of the models with Alpaca and not the instruct version.

ChatDoc. The second dataset is a domain-specific one, namely HealthCareMagic-100k[13]. HealthCareMagic-100k contains 100,000 records from the medical domain. It is a conversational dataset including interactions between doctors and patients. We employ this dataset to fine-tune the instruction-following versions of the models. Moreover, to further evaluate the **Persistent** property of Chain & Hash, we conduct double fine-tuning. We first start by fine-tuning the model on the Alpaca dataset to transition the base model into an instruction-following version, and then we fine-tune the resulting model with the HealthCareMagic-100k dataset.

Finally, for both the Alpaca and ChatDoc datasets, we use three epochs for fine-tuning.

B. Metrics

To evaluate the performance of Chain & Hash, we use the following metrics:

Fingerprint Success. Following previous works [20], [5], we measure fingerprint success by querying each fingerprint question to the target model and inspecting the generated tokens. More concretely, if the first generated token matches the specific response (as selected in Section IV-A), we consider it a success; otherwise, it is deemed a failure—even if the correct response appears later in the sequence of generated tokens. Due to the inherent randomness of large language models, this process is repeated multiple times to estimate the success rate. In this work, rather than estimating the success rate, we directly calculate its probability. More concretely, we compute the probability of the specific response being generated. Since the response may consist of multiple tokens, we calculate the product of the probabilities of all tokens in the response and consider this value as the success probability. A perfect fingerprint should have a perfect success probability (1.0).

Utility. Secondly, we assess the utility of the fingerprint by comparing the performance of the fingerprinted models to that of the pre-fingerprinted models, which we refer to as pretrained models. We employ all four benchmark datasets (Section V-A2) for this comparison. As previously discussed, we compare the performance of models using the normalized versions of benchmark performance (Section V-A2). Therefore, a perfect fingerprint should achieve a score of 1.0 or above. This metric directly evaluates the *Transparency* property.

Required Trials. As mentioned earlier (Section IV-E), proof of ownership of the model can be established with only two questions. Consequently, we also calculate the number of required trials of generations for the fingerprint questions to achieve a 99% success probability for at least two different fingerprint questions. The fewer the required trials, the more effective the fingerprint is. This metric directly evaluates the *Efficient* property.

C. Llama-3-Base

We first evaluate the base version of the Llama-3-Base model. To this end, we independently fingerprint three different models. We follow the methodologies previously presented in Section IV-C for the base models, i.e., different prompt formats are used instead of meta prompts when fingerprinting, and add random padding to increase robustness are presented in Section IV-D to create the fingerprinting dataset and fine-tune the target model. We repeat the fingerprint records in the fingerprinting dataset ten times to increase the speed of training and convergence. We stop the fine-tuning of the model when all fingerprints achieve at least a 90% success probability. We then benchmark the three different models using all four datasets and compare them with the benchmark results of the target model without any fingerprinting.

To adversarially evaluate Chain & Hash, we set ten testing meta prompts that the model has never seen during the fingerprinting process. These meta prompts include challenging prompts that cause a complete change in the model's output, e.g., instructing the model to talk like a pirate or in a snarky tone, adding specific prefixes to the answer, e.g., to always start the response with "ANSWER:", and even completely refusing to engage with the user if the input deviates from a specific topic, e.g., to only respond if the question is about weather; otherwise, reject.

1) Random Questions: We first evaluate the performance of the randomly constructed questions. We calculate the average probability across all three models, i.e., for each meta prompt, we determine the success probability for each model independently and then average the results. It is important to note that each of the three models has a different fingerprint; thus, this is an average of three distinct models, each with its unique fingerprint. We also display the performance of the fingerprints on the pre-fingerprinted model (the pretrained column). As the figures show, all fingerprints have approximately zero probability of occurring in the pretrained model. Figure 2a shows the results, where each bar represents the average cumulative probability for all 10 fingerprints. As the figure clearly demonstrates, Chain & Hash is able to maintain a cumulative probability of above 95% for most of the cases, while the lowest one is always above 80%. We believe this difference in performance of the models is mainly due to the different meta-prompts sampled and used during fingerprinting.

Figure 2b shows the utility of the fingerprinted models. As indicated by the performance across various benchmarks, the fingerprinted models exhibit a reduction of a maximum of 4% for the HellaSwag and WinoGrande benchmarks. However, there is a slight improvement for the MMLU and, surprisingly, a significant improvement for TruthfulQA. We believe this improvement is attributable to the incorporation of different prompt formats during the fingerprinting process.

2) Natural Language Questions: Secondly, we evaluate the natural language questions using the same evaluation settings presented in Section V-C1. Figure 3 presents the performance of natural language questions. As expected, using natural questions results in a reduced fingerprint strength compared to random ones, because random questions are easier for the model to overfit to, or in other words, remember. However, it still achieves strong performance for most of the meta prompts. The utility of the fingerprinted models follows a similar trend as when using random questions for fingerprinting, where most benchmarks exhibit comparable performance to the non-fingerprinted models, except for TruthfulQA, which is significantly improved.

D. Phi-3

Second, we evaluate the Phi-3 model. We follow the same evaluation settings as in Section V-C, with the only difference being the underlying model and using meta prompts instead of different prompt formats when fingerprinting (as discussed in Section IV-C). We set the number of meta prompts to be 60 which are randomly sampled for each model from a fixed 100 meta prompts (that we generated with GPT-4). These meta prompts are independent from the ones used during testing. Moreover, to show the difference of including the meta prompts we also evaluate the fingerprint performance when not including meta prompts.

E. Random Questions

We first start with the random questions. Figure 10a and Figure 4a display the results for using no meta prompts and 60 meta prompts, respectively. Compared to the base version, the instruct models are, as expected, more sensitive to meta prompts. As shown, the fingerprint strength in Figure 10a is significantly weak for all test meta prompts. Nonetheless, including meta prompts enhances the probabilities to almost 1.0 for all fingerprints and models. Benchmarking the datasets with Chain & Hash also indicates negligible degradation in these settings, less than 0.5% for both scenarios, with and without using meta prompts.

F. Natural Language Questions

Next, we evaluate the natural language questions. Similar to the previous evaluations, the natural language questions are more challenging for the model to learn, which is seen when comparing Figure 4 (random) with Figure 5 and Figure 11 (natural language). However, similar to the Llama-3-Base version, the performance of Chain & Hash is significantly improved when including meta prompts in fingerprinting, as shown in Figure 5a. Like the other models, benchmarking the phi-3 fingerprinted models results in no drop in most cases, with only three models experiencing a drop between 1-1.2% on the TruthfulQA dataset, which again demonstrates the effectiveness of Chain & Hash.

G. Llama-3-Instruct

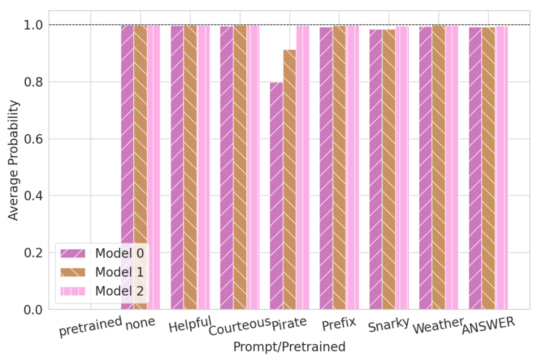
We now evaluate the LLama-3-Instruct model. We follow the same evaluation settings of Section V-D but set the target model to Llama-3-Instruct 7b.

1) Random Questions: The same trend of results extends to Llama-3-Instruct as well. As shown in Figure 6, augmenting the fingerprinting datasets is necessary to enhance the performance of the fingerprint and make it robust against changes in the meta prompts. Similarly, the benchmark evaluation does not exhibit a drop of more than 1.5% for TruthfulQA, while the performance on other benchmarks remains nearly identical (with less than a 0.1-0.2% difference) to the non-fingerprinted model.

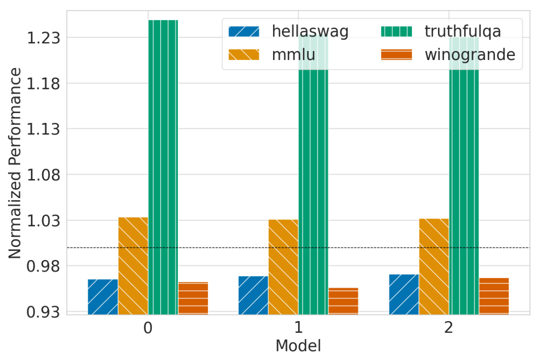
2) Natural Language Questions: Similar to other models, using natural language is more challenging for fingerprinting. The "Pirate" meta prompt keeps being particularly difficult to bypass robustly as other models. Furthermore, the TruthfulQA benchmark is the most impacted, with one model's performance dropping to 5% (Model 2 with 60 meta prompts), while another, in the same exact settings, maintains no drop (Model 1 with 60 meta prompts).We believe this difference can be attributed to the different sampled meta-prompts during fingerprinting. This is a hyperparameter that can be optimized by the model owner to maximize performance. Overall, the performance of Chain & Hash with natural language questions is strong for the llama-3-Instruct model, with all but the "Pirate" and "weather" meta prompt achieving a 80% success probability. This indicates that querying each question once would result in over 99% likelihood of having two questions succeed, thereby proving the ownership of the model.

H. Llama-2 13b

Finally, we evaluate the next larger version of the model (limited computational resources prevented us from evaluating the largest 70b version). The trend of fingerprints being bypassed when changing meta prompts continues for this model as well when not including meta prompts during the fingerprinting, hence we only present the results for including 60 meta prompts during fingerprinting.

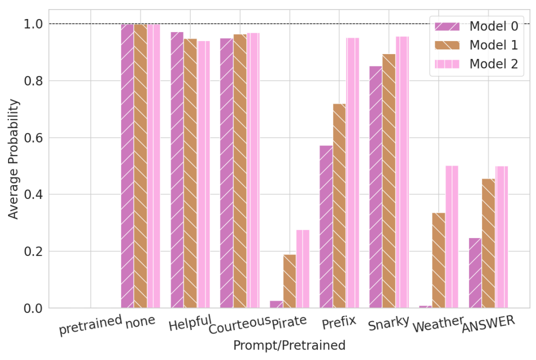


(a) Fingerprints Strength.

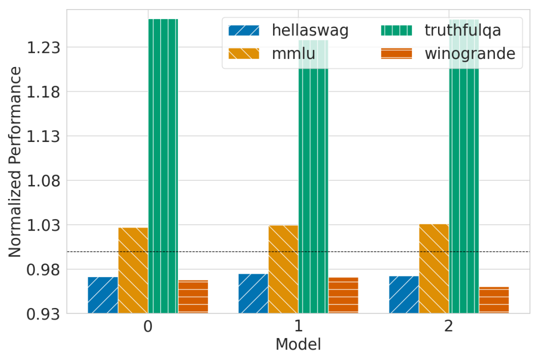


(b) Benchmark Performance.

Fig. 2: Performance of Chain & Hash, with no meta prompts used during fingerprinting, on Llama-3-Base 8b with random questions.

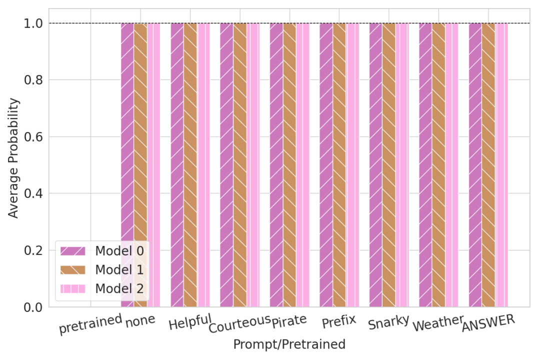


(a) Fingerprints Strength.

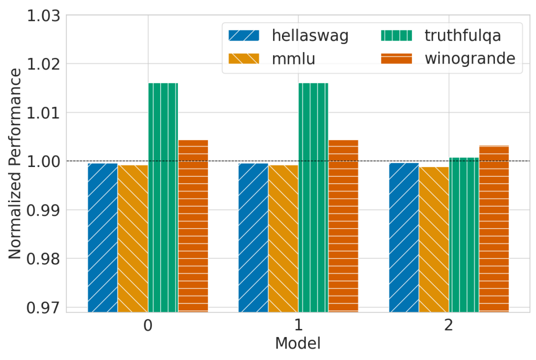


(b) Benchmark Performance.

Fig. 3: Performance of Chain & Hash, with no meta prompts used during fingerprinting, on Llama-3-Base 8b with the natural language questions.



(a) Fingerprints Strength.



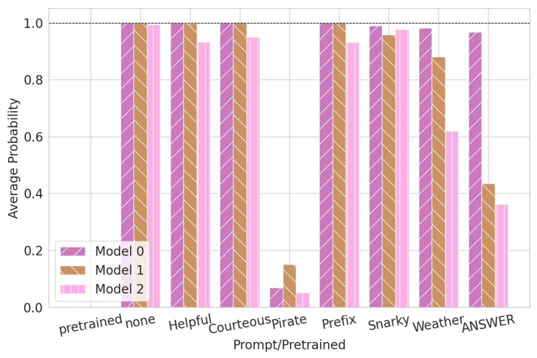
(b) Benchmark Performance.

Fig. 4: Performance of Chain & Hash, with 60 meta prompts used during fingerprinting, on the instruct version of Phi-3 with random questions.

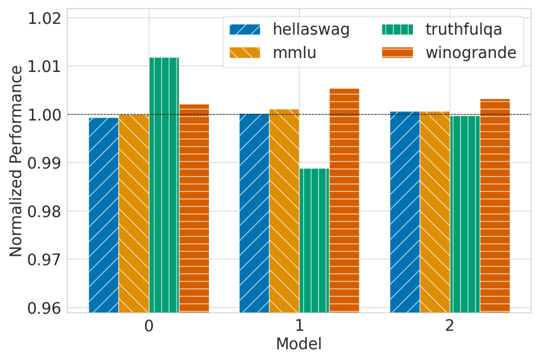
1) Random Questions: Similar to other models, random fingerprints achieve stronger performance as shown in Figure 8a, while achieving virtually no drop (less than 0.5%) as indicated in Figure 8b.

I. Natural Language Questions

Figure 8c shows the performance of Chain & Hash on the Llama-13b models. As the figure demonstrates, the results for this model are better than those for the previous smaller

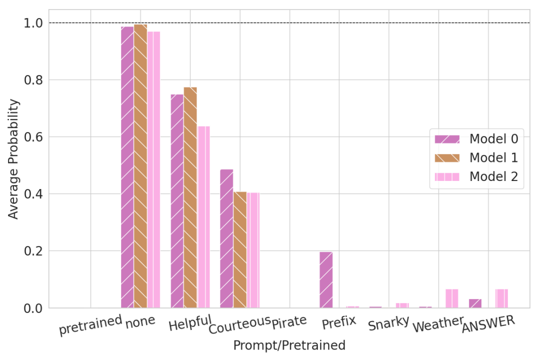


(a) Fingerprints Strength.

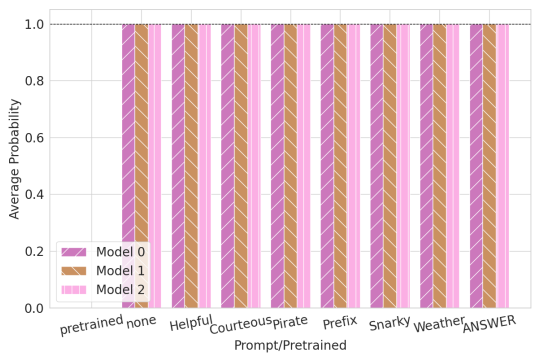


(b) Benchmark Performance.

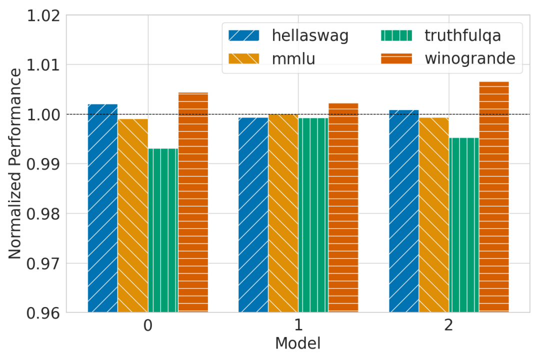
Fig. 5: Performance of Chain & Hash, with 60 meta prompts used during fingerprinting, on the instruct version of Phi-3 with natural language questions.



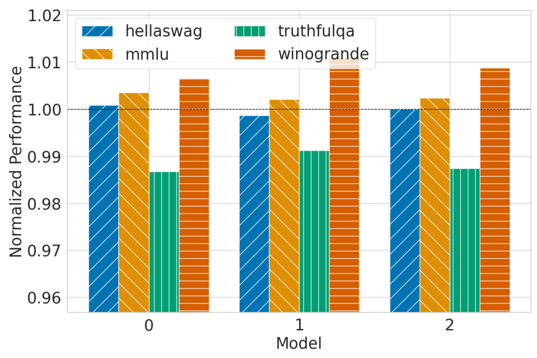
(a) Fingerprints performance with no meta prompts



(b) Fingerprints performance with 60 meta prompts.



(c) Benchmark Performance with no meta prompts.

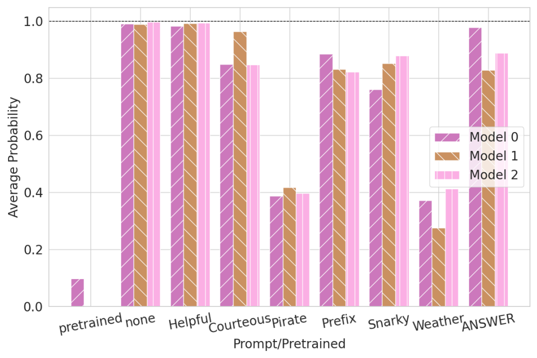


(d) Benchmark Performance with 60 meta prompts.

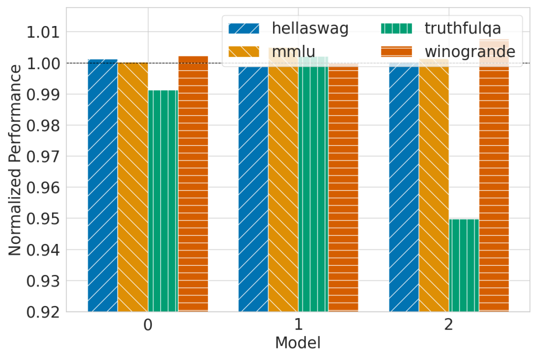
Fig. 6: Performance of Chain & Hash of Llama-3-Instruct with random questions.

ones. We hypothesize that the larger the model, the easier it is to learn the fingerprints. This could be attributed to having more parameters, which can facilitate overfitting to the fingerprints. Benchmarking this model also reveals that it experiences less performance drop with the more challenging benchmarks (such as TruthfulQA), always achieving the same or slightly better results than the pretrained model. The other benchmarks also show less than a 1% drop in performance, which is an improvement over the other smaller models.

It is important to note that the success of the final fingerprint is dependent on the average success rate of all fingerprints. As the figures show, querying the 10 fingerprints would almost guarantee to get two fingerprint responses, hence proving ownership (just two fingerprints are enough to cryptographically prove ownership).

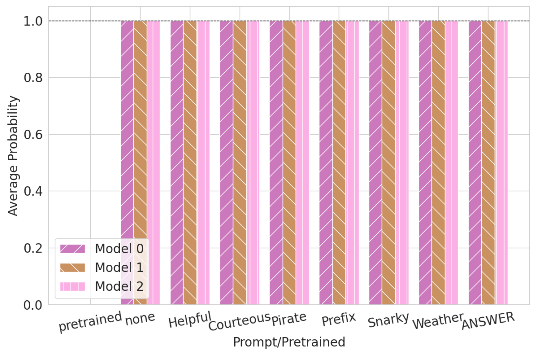


(a) Fingerprints Strength.

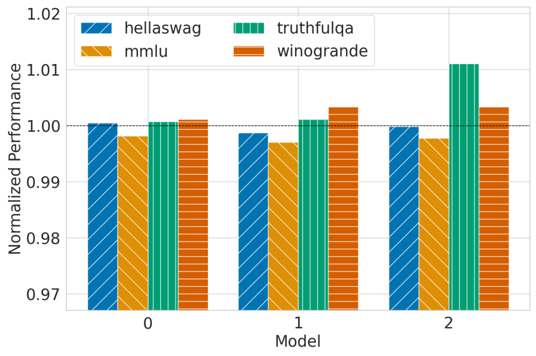


(b) Benchmark Performance.

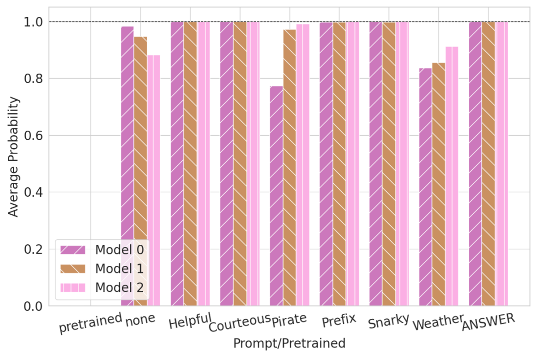
Fig. 7: Performance of Chain & Hash, with 60 meta prompts used during fingerprinting, of Llama-3-Instruct with natural language questions.



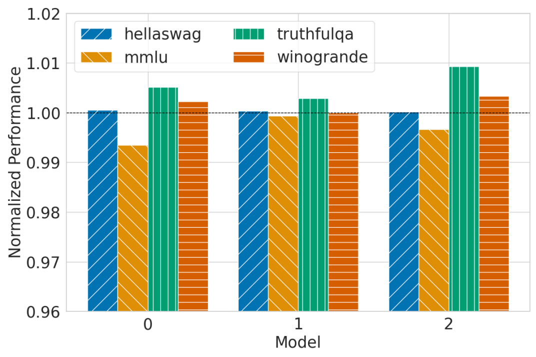
(a) Fingerprints Strength (Random Questions)



(b) Benchmark Performance (Random Questions)



(c) Fingerprints Strength (Natural Language Questions)



(d) Benchmark Performance (Natural Language Questions)

Fig. 8: Performance of Chain & Hash, with 60 meta prompts used during fingerprinting, on the instruct version of Llama-2 13b with random and natural language questions.

J. *Efficiency*

As previously introduced (Section III-A), fingerprints should be *efficient* when implemented. For Chain & Hash, all models required between one and ten epochs on the fingerprinting dataset to converge. As expected, random questions consistently took fewer epochs to converge compared to the natural language ones.

K. *Robustness and Persistence*

Next, we assess the robustness and persistence of Chain & Hash. As introduced in Section III-A, fingerprints need to be evaluated *adversarially*. To this end, we have first evaluated them against different meta prompts, and now we evaluate their resilience against finetuning (the *Persistent* property).

Finetuning. Due to the computational cost of fine-tuning, we only consider Llama-3, as it includes both the base and instruct versions. For the base version, we fine-tune it using Alpaca and then evaluate the effect of double fine-tuning it with ChatDoc, i.e., initially with Alpaca and then with ChatDoc. As for the instruct version, we only fine-tune with ChatDoc since it is already instruction-tuned.

For the Llama-3-Instruct model, we assume a full black-box assumption. During fine-tuning, we use the Llama-3-Instruct formatting since the model is already tuned for instructions. However, for the Llama-3-Base model, as it is not instruction-tuned, we adhere to Alpaca's formatting when fine-tuning with Alpaca and ChatDoc. Evaluating the fingerprint with the fine-tuned Llama-3-Base models under the black-box assumption revealed that the fingerprint strength is significantly degraded, especially when meta prompts were employed. We believe the main reason for this degradation is the alteration in prompt format post-fine-tuning. For instance, Alpaca enforces a different prompt format from the original Llama-3-Base one. Hence, we relax the assumption for this scenario and consider gray-box access to the model, which presumes that the model owner can set the prompt formatting during inference. However, it is important to note that we do not presume any knowledge of the prompt formatting used in fine-tuning.

We report the number of queries/generations required to achieve a 99% success probability for at least two questions. We do not report this metric for the non-finetuned versions of the model since they are consistently approximately one, i.e., the model owner would need to query each question approximately only once to have a 99% chance that two of them will yield the correct response. We consider any fingerprint which require more than $1,000$ trial to be removed, i.e., the model is not fingerprintted anymore. A trial here is trying each question in the fingerprint once.

Figure 9 illustrates the performance of both versions of Llama-3 fine-tuned with Alpaca, ChatDoc, and both datasets when using random and natural language questions. The figures demonstrate that using random questions for fingerprinting is more robust against fine-tuning. For instance, a model owner would require a maximum of 25 trials to prove ownership of a Llama-3-Instruct model fine-tuned with Alpaca. Using normal questions instead of random ones increases the maximum number of trials to 117 (which occurs when using the pirate meta prompt). Another interesting observation is that the random fingerprint actually performs better with meta prompts than without, unlike with normal questions. We believe this behavior is due to the fact that random tokens overfit more to the context, hence when provided with more context they achieve better performance.

For the Llama-3-Base model, as shown in Figure 9, using normal questions is already sufficient to achieve strong robustness. This efficiency highlights the effectiveness of Chain & Hash with gray-box access. For example, it requires only one and three trials without using any meta prompts and fingerprinting the model with Alpaca and Alpaca and ChatDoc, respectively. Similarly, the required number of trials has a maximum of 161 for Alpaca and 35 for the ChatDoc. It is interesting to note that fine-tuning with ChatDoc on top of Alpaca does not weaken the fingerprint's strength in most cases; in fact, it sometimes enhances it.

Quantization. Another robustness aspect we evaluate Chain & Hash against is quantization. To this end, we evaluate the performance of Chain & Hash after quantizing the fingerprinted model. We quantize the models to INT8 and evaluate the performance of the fingerprints. Our results show approximately no drop (most models less than 0.5%, with the maximum difference being less than 2.5%) in the fingerprint's strength, hence demonstrating the resilience of the Chain & Hash technique.

L. *Hyperparameters*

We evaluate some of the hyperparameters for Chain & Hash. Due to space restrictions, we only summarize our findings here. For all our hyperparameter experiments we fix Llama-3-Instruct as the target model.

Number of Meta Prompts. The first hyperparameter is the number of meta prompts included during fingerprinting. For random questions, we were able to reduce the number of meta prompts to only six, and it still produced similar performance, as shown in Figure 12. However, reducing the meta prompts for natural language questions can make the fingerprint less robust against changes in meta prompts. We believe this is because the fact that the model can easily overfit to the random tokens in random questions, learning to ignore the rest of the text regardless of the used meta prompt, which is not possible or harder with natural language questions.

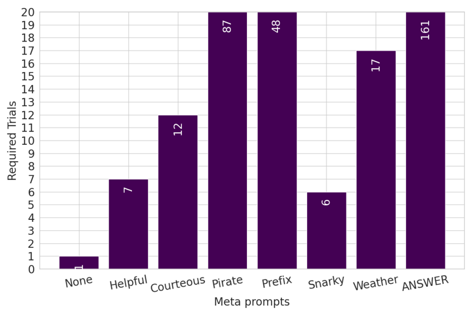
Number of Questions. The second hyperparameter we explore is the number of questions in the fingerprint. We evaluate Chain & Hash with ten times more questions, i.e., 100 fingerprint. As Figure 13 indicates, having 100 natural language questions fingerprint still results in the same performance while nearly maintaining benchmark performance. Moreover, using more questions makes the fingerprints more robust against meta prompts, as the model is exposed to more data. It also takes fewer epochs to converge; however, each epoch is more costly due to the increased size of the dataset. We repeat the experiment using random questions and observe similar trend.

VI. Discussion

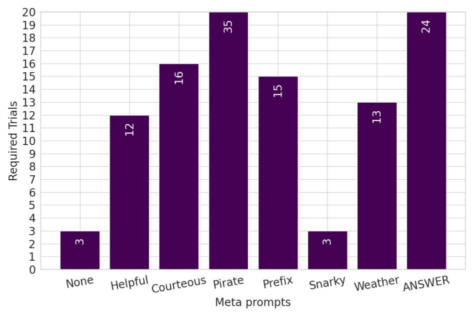
We now discuss some general properties for fingerprints and the limitations of our approach Chain & Hash.

A. *Question Generation*

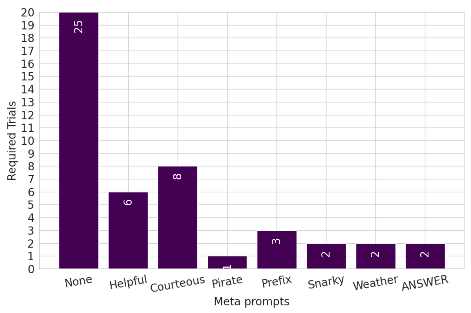
As previously discussed, there are multiple techniques for generating fingerprint questions. Evaluations show that using random tokens to construct the questions can make the fingerprint efficient and more robust compared to using natural language questions. However, natural language questions are more stealthy. We believe that both approaches are complementary and should be chosen based on the application or domain of the target model. For instance, if the model is a general-purpose one, where filtering its inputs is challenging due to its ability to encode/decode base64 and handle links that may resemble random text, random questions are suitable. Conversely, for a domain-specific model where an adversary may heavily filter inputs, domain-specific questions would be



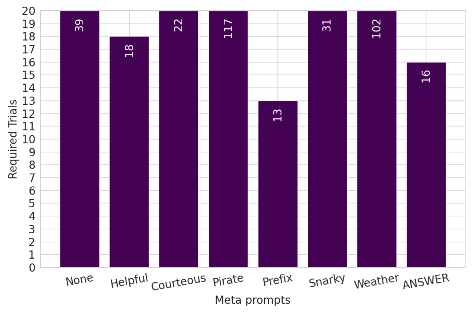
(a) Llama-3-Base finetuned with Alpaca (Natural Language Questions)



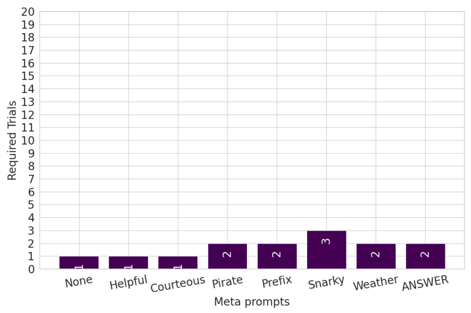
(b) Llama-3-Base finetuned with Alpaca then ChatDoc (Natural Language Questions)



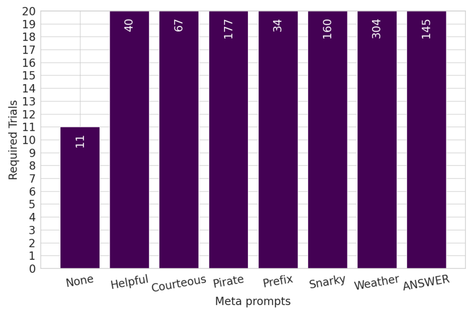
(c) Llama-3-Instruct finetuned with ChatDoc (Random Questions)



(d) Llama-3-Instruct finetuned with ChatDoc (Natural Language Questions)



(e) Llama-3-Base finetuned with Alpaca (Random Questions)



(f) Llama-3-Base finetuned with Alpaca then ChatDoc (Random Questions)

Fig. 9: The robustness of Chain & Hash when finetuning Llama-3-Base and instruct versions.

the optimal choice for fingerprinting to minimize the chances of an adversary filtering them out.

Another distinction between these approaches is the degree to which the model overfits the questions. A random sequence of tokens can result in the model overfitting on a subset of them. For example, we were able to reduce the complete question/prompt of the publicly fingerprinted models [20] to just a couple of tokens instead of the full prompt, which we believe is why random fingerprints are more robust. Models learn to focus on these subsets of tokens and completely ignore the rest of the input.

Finally, there is the potential of combining both approaches during fingerprinting, which includes both random and natural language questions for fingerprinting the model. We leave the exploration of this hybrid approach for future work.

B. *Response Selection*

It is crucial to differentiate between random questions and random responses. Random questions enhance the robustness of the fingerprint because they train the model to disregard all other text, such as meta prompts, and to concentrate solely on the random sequence. However, a random response might not be particularly helpful since it is not part of the input but rather produced by the model. Therefore, we believe that random responses have more disadvantages than advantages. Intuitively, random responses simplify adversarial attacks through filtration and post-processing. For instance, adversaries could use output filters that only allow English characters or words. Additionally, random responses might cause the model to learn specific sequences of tokens that it can generate with significantly higher probability. For example, we generate 10,000 outputs from the fingerprinted model published in [20], and the fingerprint response appeared three times. Furthermore, generating just 1,000 outputs with a prefix of "#" led to even more frequent occurrences of the fingerprint response. Hence, an adversary could easily generate 10,000 samples, search for any text that appears random or infrequent, and filter it out from the model.

Therefore, we recommend using responses that are neither unique nor random, yet still improbable for that specific question (or fingerprint in general). This approach has the advantage that even if a response is leaked, the adversary cannot filter it out without compromising the model's utility. For example, the response "'yes, actually" is nearly impossible for an adversary to recognize if encountered, as it is a common phrase, and even if it becomes known, preventing the model from ever saying such words would affect its utility or at least smoothness of the output.

Finally, we believe that a potential future direction for strengthening the fingerprint's strength is to optimize the response.

C. Access To Multiple Fingerprinted Models

As previously mentioned, we propose a new threat model in which the adversary has access to multiple fingerprinted models. Under this scenario, the adversary can input the same queries to both models and monitor any discrepancies in the outputs. That is, while both models should behave identically on standard inputs, they should generate distinct fingerprinted responses when presented with fingerprint questions.

Although we cannot use the same fingerprint across all models, since a leaked model would not reveal the specific version of the original model (assuming here that the target model is distributed to different parties and the goal is to identify which specific one leaked), we propose the use of intersecting sets of fingerprints. For example, as Chains of only two questions are already cryptographically hard to forge. The model owner can create multiple two-question Chains in their model and fingerprints different models with intersecting Chains such that even if the adversary has access to multiple models, they cannot completely filter all Chains, i.e., the intersected chains will have the same response, hence not filtered. That approach will be robust against access to multiple fingerprinted models up to a point. In other words, it is similar to the secure multi-party computation property of security against c colluding users; where a colluding user here is a fingerprinted model. The design of the intersection of the chains is done with respect to the required level of c.

D. Optimizing Meta Prompts

As demonstrated in the different figures, changing the meta prompts included in the construction of the fingerprinting dataset can further enhance the performance of Chain & Hash. For all results shown in this paper, we randomly sample the meta prompts. Therefore, we do not cherry-pick any results or meta prompts to boost the fingerprint's performance. When using the Chain & Hash, we recommend optimizing the meta prompts dataset to further improve the fingerprint's robustness.

The same applies to question and response generation. We randomly generate these; however, they can be further optimized to improve performance. For instance, questions spanning different topics can be used as fingerprints with responses that are more related to the questions yet still improbable. This makes it easier for the model to memorize.

E. Limitations

Finally, regarding the limitations of Chain & Hash, first, there is an inherent limitation for any black-box fingerprints, which is that it is impossible to create a perfectly secure fingerprint without linking it to a target utility. Since an adversary could simply take a model and make it output a constant response, thereby circumventing any black-box fingerprint.

Second, as meta prompts can affect the performance of Chain & Hash on the instruct versions of models, the inclusion of meta prompts in the fingerprinting process is necessary. The amount of meta prompts and non-fingerprinted inputs depends on the model. For example, using the same hyperparameters for Phi-3 as for the other models results in a lower average success rate. However, increasing the number of meta prompts can increase the success rate.

Third, sometimes the response leaks when using some of the meta prompts on non-fingerprinted questions. However, due to the use of typical phrases as responses, we believe such leaks are acceptable as removing these responses can degrade the model's utility. Furthermore, the majority of the fingerprint responses do not leak; the leaks are approximately less than 0-10% of the fingerprints. This means that even if filtered, the majority of the other fingerprints remain intact.

Finally, it is important to note that Chains can be constructed to provide robustness against both leakage and meta prompts. For example, to achieve two successful fingerprints (the minimum to cryptographically confirm fingerprint success) when considering a chain of size 10, the average fingerprint success rate needs to be around 41% to reach a fingerprint success threshold of 95%, i.e., the percentage of two different questions resulting in the correct response. This result assumes only querying the question once; querying the question twice would reduce the required average success rate of questions even further. This demonstrates that, despite such limitations, Chain & Hash still achieves strong fingerprinting performance.

VII. Conclusion

We present Chain & Hash, the inaugural fingerprinting technique for Large Language Models (LLMs) that can cryptographically boost the uniqueness of fingerprints, regardless of the method used to construct the underlying fingerprints. Furthermore, we introduce two distinct methods for constructing fingerprints: the first method utilizes a random sequence of tokens, while the second employs natural language. Both construction techniques are complementary and depend on the specific application at hand. Additionally, we illustrate how to adapt Chain & Hash to both the instruct and base versions of the target model. Lastly, we assess Chain & Hash across multiple state-of-the-art LLMs and demonstrate its efficacy and robustness, even when the models are subjected to fine-tuning.

References

[1] K. Chen, Y. Meng, X. Sun, S. Guo, T. Zhang, J. Li, and C. Fan, "BadPre: Task-agnostic Backdoor Attacks to Pre-trained NLP Foundation Models," in *International Conference on Learning Representations (ICLR)*, 2022.

[2] X. Chen, A. Salem, M. Backes, S. Ma, Q. Shen, Z. Wu, and Y. Zhang, "BadNL: Backdoor Attacks Against NLP Models with Semantic-preserving Improvements," in *Annual Computer Security Applications Conference (ACSAC)*. ACSAC, 2021, pp. 554–569.

[3] M. Christ, S. Gunn, and O. Zamir, "Undetectable watermarks for language models," 2023.

[4] P. Fernandez, G. Couairon, T. Furon, and M. Douze, "Functional invariants to watermark large transformers," 2024.

[5] C. Gu, C. Huang, X. Zheng, K.-W. Chang, and C.-J. Hsieh, "Watermarking pre-trained language models with backdooring," 2023.

[6] X. He, Q. Xu, L. Lyu, F. Wu, and C. Wang, "Protecting intellectual property of language generation apis with lexical watermark," 2021.

[7] D. Hendrycks, C. Burns, S. Basart, A. Critch, J. Li, D. Song, and J. Steinhardt, "Aligning ai with shared human values," *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[8] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt, "Measuring massive multitask language understanding," *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[9] E. Hubinger, C. Denison, J. Mu, M. Lambert, M. Tong, M. MacDiarmid, T. Lanham, D. M. Ziegler, T. Maxwell, N. Cheng, A. Jermyn, A. Askell, A. Radhakrishnan, C. Anil, D. Duvenaud, D. Ganguli, F. Barez, J. Clark, K. Ndousse, K. Sachan, M. Sellitto, M. Sharma, N. DasSarma, R. Grosse, S. Kravec, Y. Bai, Z. Witten, M. Favaro, J. Brauner, H. Karnofsky, P. Christiano, S. R. Bowman, L. Graham, J. Kaplan, S. Mindermann, R. Greenblatt, B. Shlegeris, N. Schiefer, and E. Perez, "Sleeper agents: Training deceptive llms that persist through safety training," 2024.

[10] N. Kandpal, M. Jagielski, F. Tramèr, and N. Carlini, "Backdoor attacks for in-context learning with language models," 2023.

[11] J. Kirchenbauer, J. Geiping, Y. Wen, J. Katz, I. Miers, and T. Goldstein, "A watermark for large language models," 2023.

[12] P. Li, P. Cheng, F. Li, W. Du, H. Zhao, and G. Liu, "Plmmark: A secure and robust black-box watermarking framework for pre-trained language models," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 12, pp. 14 991–14 999, Jun. 2023. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/26750

[13] Y. Li, Z. Li, K. Zhang, R. Dan, S. Jiang, and Y. Zhang, "Chatdoctor: A medical chat model fine-tuned on a large language model meta-ai (llama) using medical domain knowledge," 2023.

[14] S. Lin, J. Hilton, and O. Evans, "Truthfulqa: Measuring how models mimic human falsehoods," 2022.

[15] M. MacDiarmid, T. Maxwell, N. Schiefer, J. Mu, J. Kaplan, D. Duvenaud, S. Bowman, A. Tamkin, E. Perez, M. Sharma, C. Denison, and E. Hubinger. (2024) Simple probes can catch sleeper agents. [Online]. Available: https://www.anthropic.com/news/probes-catch-sleeper-agents

[16] X. Pan, M. Zhang, B. Sheng, J. Zhu, and M. Yang, "Hidden Trigger Backdoor Attack on NLP Models via Linguistic Style Manipulation," in *USENIX Security Symposium (USENIX Security)*. USENIX, 2022, pp. 3611–3628.

[17] K. Sakaguchi, R. L. Bras, C. Bhagavatula, and Y. Choi, "Winogrande: An adversarial winograd schema challenge at scale," 2019.

[18] R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, and T. B. Hashimoto, "Stanford alpaca: An instruction-following llama model," https://github.com/tatsu-lab/stanford_alpaca, 2023.

[19] R. Wen, T. Wang, M. Backes, Y. Zhang, and A. Salem, "Last one standing: A comparative analysis of security and privacy of soft prompt tuning, lora, and in-context learning," 2023.

[20] J. Xu, F. Wang, M. D. Ma, P. W. Koh, C. Xiao, and M. Chen, "Instructional fingerprinting of large language models," 2024.

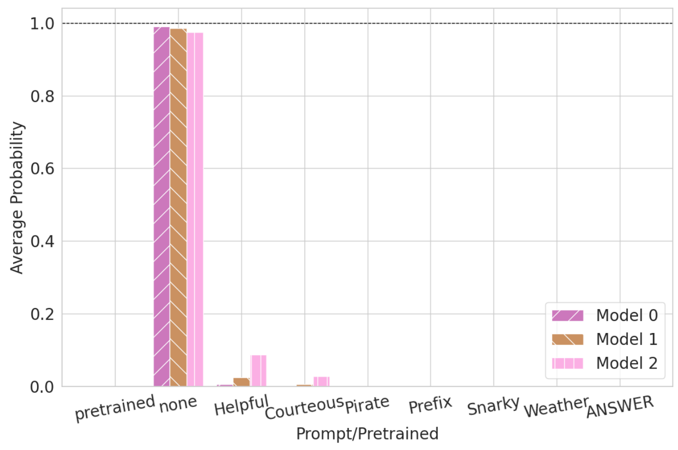
[21] X. Yang, K. Chen, W. Zhang, C. Liu, Y. Qi, J. Zhang, H. Fang, and N. Yu, "Watermarking text generated by black-box language models," 2023.

[22] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi, "Hellaswag: Can a machine really finish your sentence?" in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

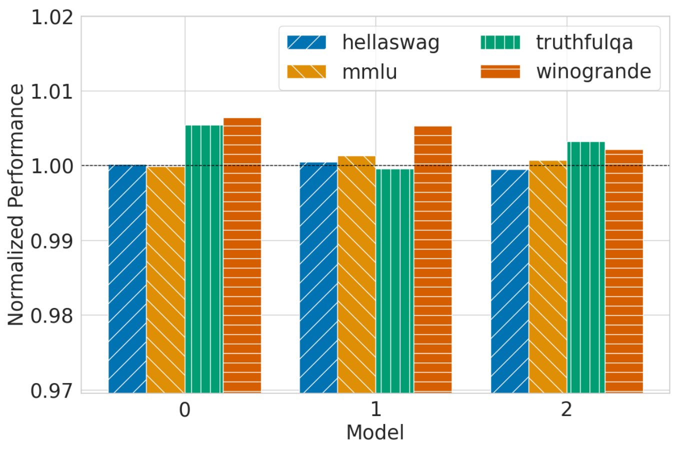
[23] B. Zeng, C. Zhou, X. Wang, and Z. Lin, "Human-readable fingerprint for large language models," 2024.

APPENDIX

A. *No Meta Prompts Plots*

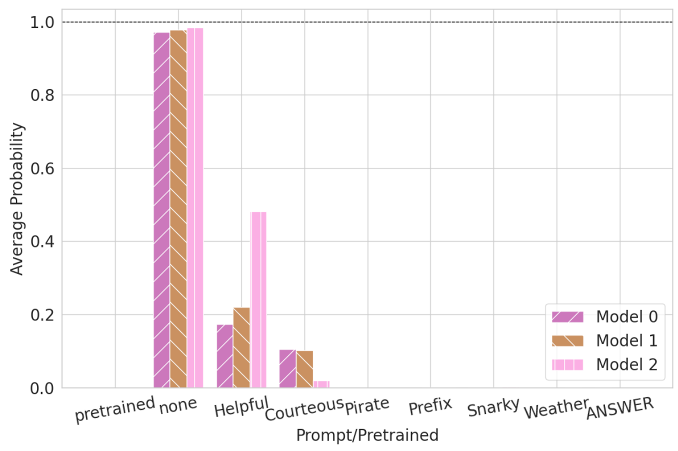


(a) Fingerprints performance with no meta prompts.

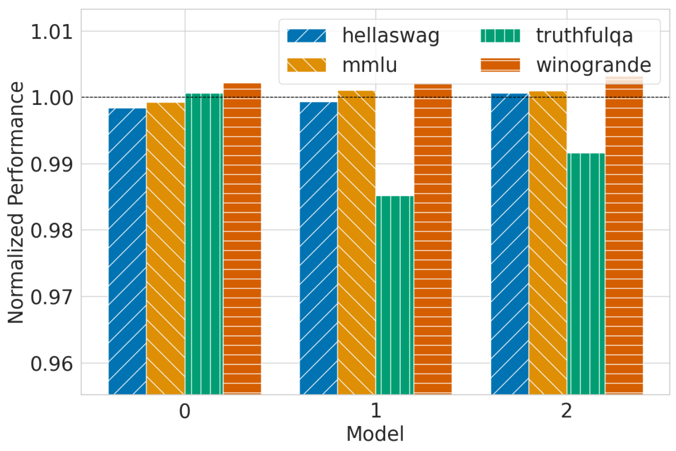


(b) No Meta Prompts (Benchmark)

Fig. 10: Performance of Chain & Hash on the instruct version of Phi-3 with random questions.



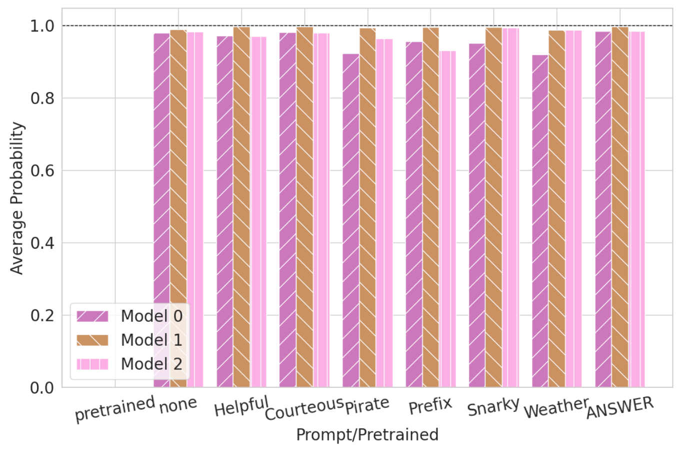
(a) Fingerprints performance with no meta prompts



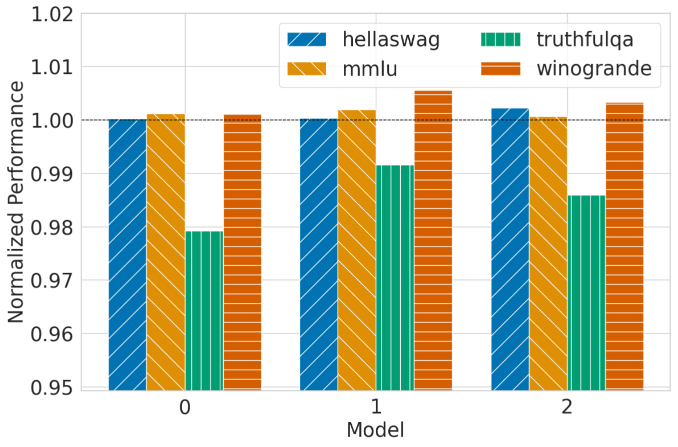
(b) Benchmark Performance with no meta prompts

Fig. 11: Performance of Chain & Hash on the instruct version of Phi-3 with natural language questions.

B. Hyperparameter Plots

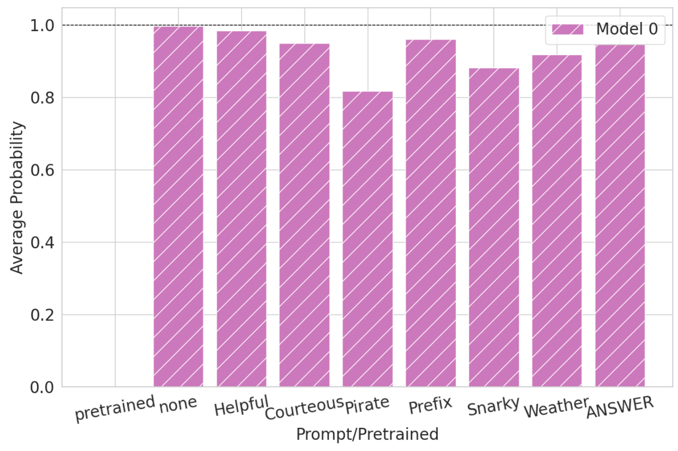


(a) Fingerprint Performance with six meta prompts (Random Questions).

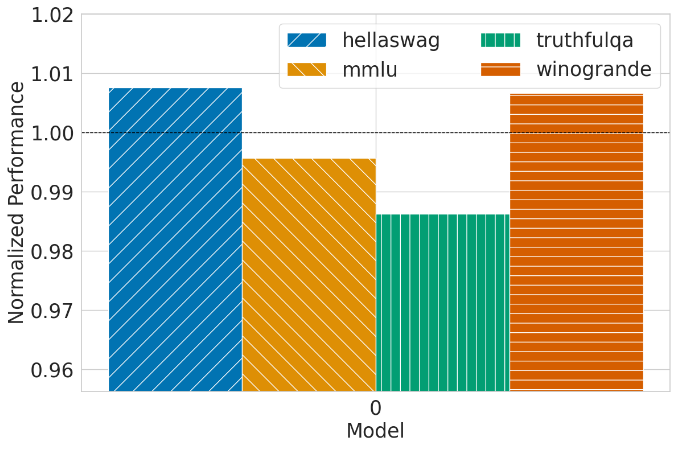


(b) Benchmark Performance with six meta prompts (Random Questions).

Fig. 12: Chain & Hash with only six meta prompts on Llama-3-Instruct version.



(a) Fingerprint Performance with 100 fingerprints (Natural Language Question).



(b) Benchmark Performance with 100 fingerprints (Natural Language Question).

Fig. 13: Chain & Hash with 100 fingerprints on Llama-3-Instruct version.