

AI-Identities: **Fingerprinting &** **Cataloguing LLMs** **Through Their** **Generated Responses**

Ahmed, Babur, Mark, Sani, Stephen, Zayd

Table of Contents

Summary (Ahmed, Mark)	3
Project Background and Context (Sani, Zayd)	4
Problem Definition (Stephen, Zayd)	5
Problem 1 (Ensuring transparency with LLMs).....	5
Problem 2 (Lack of a collaborative repository to centralize LLM information).....	6
Solution (Stephen, Babur)	7
LLM Fingerprinting.....	7
The “Fingerprint Library”.....	7
Identification Procedure.....	10
How Fingerprinting Impacts LLM Transparency.....	11
Wiki.....	13
Data Analysis and Identification Development Process (Babur, Stephen)	13
Classification methods.....	14
Word frequency classifiers.....	15
Embedding vector classifier.....	16
Datasets.....	16
Base dataset experiment.....	16
System prompt dataset.....	18
System prompt dataset experiment.....	19
Final CLF Dataset.....	23
Final CLF Dataset Experiment.....	24
Library LLM analysis.....	27
Impact of Model Training Data, Training Processes, and Architecture on Model Grouping	28
Impact of Thinking Process on Model Grouping.....	29
Ethical, Social, or Community Implications (Zayd, Sani, Ahmed)	30
Recommendations and Future Work (Ahmed, Mark, Babur)	31
Recommendations.....	31
Accessibility.....	31
User Guides.....	32
LLM Fingerprint and Identification Tool.....	33
Next Steps.....	33
Accessibility: Colors.....	33
Accessibility: Semantic HTML.....	35
User Guides: GitHub Pages.....	35
Continued Exploration.....	35
References	37

Summary (Ahmed, Mark)

Our team developed **LLMDetective**, a web application designed to both identify large language models (LLMs) from their responses and serve as a collaborative knowledge hub for LLM-related information. The project was motivated by two challenges: the difficulty of reliably distinguishing between LLMs when given only black-box access to their outputs, and the lack of a centralized, community-driven repository for information about LLMs and their applications.

The first half of the project was focused on fingerprinting and categorizing LLMs. We developed a procedure that queried an unknown LLM with a series of carefully constructed open-ended identification questions. The responses were then vectorized into numerical representations through TF-IDF, averaged into a "fingerprint," and compared to a database of known LLM fingerprints using cosine similarity. When the similarity value was above a calibrated level, the system classified the model; otherwise, the system returned "unknown," which helped with out-of-distribution detection. We evaluated a number of classification approaches ranging from word frequency-based to embedding-based classifiers on 36 LLMs, across varying prompts, temperature settings, and system prompt variations. Our experiments showed that the TF-IDF trigram with cosine similarity performed the best across the board, with near perfect accuracy (~97% top-1) under controlled scenarios and remaining robust even under prompt variation.

The second feature of LLMDetective is the LLM Wiki, which targeted the dispersed segments of information surrounding LLMs. This feature allowed users to author and edit wiki-like pages summarizing critical information such as release dates, authors, architectural features (e.g., dense vs. mixture-of-experts), ancestors, and known security vulnerabilities. Through this feature, the platform not only aggregates information but also highlights relationships between models, such as shared vulnerabilities and tendencies to hallucinate.

Furthermore, we observed key characteristics in the behaviour of LLMs. We learned that open-ended prompts constrained with some speaking styles (e.g., "speak like a professor") significantly improved classification accuracy, as trying to neutralize system prompts had a tendency to drop performance. In addition, comparison analysis of model answers picked out large clusters: e.g., Qwen3 models grouped together tightly with DeepSeek due to collaborative mixture-of-experts and "thinking" architectures while the Qwen2.5 devoid of these features diverged from its successors.

Holistically, LLMDetective demonstrates that interpretable, lightweight classifiers are appropriate for language model detection with small data and are fit for deployment in real-world scenarios, even in API rate-limited contexts. When coupled with the LLM Wiki, the system not only supports accurate model detection but also has the foundation for cooperative community-based knowledge sharing about the rapidly evolving LLM space.

Project Background and Context (Sani, Zayd)

LLMs are being rapidly integrated into applications across industries, powering tools like customer support chatbots, content summarizers, and productivity assistants. This growth has outpaced the frameworks meant to ensure transparency and disclosure. In many cases, companies do not reveal which model their app is built on, and some even combine multiple LLMs without making this clear to users. This lack of disclosure raises concerns about security, privacy, and accountability. For instance, flaws in a specific LLM family or the fact that certain providers may collect and sell user data are risks that users should be aware of before trusting an application.

Researchers and communities have made progress toward transparency, but the focus has mostly been on models themselves rather than applications. Platforms like Hugging Face provide centralized hosting for models and datasets, while frameworks such as Model Cards (Mitchell et al., 2019) promote standardized documentation of model performance and limitations. More recently, the Foundation Model Transparency Index

has benchmarked how openly leading developers disclose information about their models (Bommasani et al., 2023). These initiatives mark important steps forward, but they do not address the gap of app-level disclosure involving information about which LLMs power which applications.

In this context, our project is motivated by the need to bridge that gap. By combining a classifier tool with a community-maintained wiki, the system aims to help identify which LLMs power different applications and collect that information in one place. This not only gives users a way to stay informed about possible security or ethical concerns but also contributes to broader goals of transparency and accountability in the AI ecosystem.

Problem Definition (Stephen, Zayd)

Problem 1 (Ensuring transparency with LLMs)

LLMs are being used more and more, but it's not always clear how they actually work. This lack of transparency makes it harder to know whether their outputs are fair, how decisions are made, or if hidden errors and biases are affecting results. Without some way of understanding these systems, it becomes difficult for researchers, companies, or everyday users to trust them.

One reason this problem exists is that many of the most advanced LLMs are closed-source. Companies don't release details about their training data or design, which means independent researchers can't fully evaluate them. Even with open-source models, there's no standard way to "fingerprint" or compare systems, so transparency varies a lot depending on which model is being used. It is hard to know what biases and vulnerabilities a newly-released LLM might have until after extensive testing.

The impacts of this lack of transparency are significant. Developers may not know whether a model has the right capabilities for their application. Companies risk

reputational or financial damage if they deploy biased, faulty, or vulnerable systems, while policymakers face challenges in holding organizations accountable and LLM creators lack straightforward means to enforce licensing for the LLMs they create. End-users of LLM powered applications can't be sure if the functionality they're paying for is what they're getting.

Problem 2 (Lack of a collaborative repository to centralize LLM information)

As AI applications built on LLMs spread across industries, it becomes clear that there is no central repository that tracks which models are being used. While some platforms such as Hugging Face provide partial repositories for models, and frameworks like Model Cards (Mitchell et al., 2019) promote transparency in model documentation, there is still no comprehensive, standardized system that discloses which LLMs power which applications. Information remains scattered across technical papers, blog posts, developer forums, or left undisclosed altogether. This lack of a collaborative, standardized space makes it difficult to build a shared understanding of how LLMs are deployed in practice.

The problem exists partly because companies often avoid disclosing which model powers their apps. Sometimes this is to protect proprietary information, and sometimes it's because no standard exists to encourage disclosure. Even when researchers or developers identify models, the information is solely within individual tools, datasets, or community efforts, rather than being collected in a single place.

The impacts of this fragmentation are significant. Developers have no straightforward way to compare which models perform best in different applications. Researchers struggle to validate claims or replicate findings. Companies risk credibility when their AI tools appear as "black boxes." End users, meanwhile, are left in the dark about whether the model generating their content is reliable or biased. At a larger scale, this lack of

centralization slows innovation and makes it harder for policymakers to set clear accountability standards.

Solution (Stephen, Babur)

Given the two main problems above, we developed a web application with two main features.

First, an LLM fingerprinting and identification tool which identifies unknown LLMs based on their responses to the following "identification prompt":

Invent a new taste unknown to humans. Describe (in less than 800 words) how it feels, what foods or cuisines feature it, and how it could transform food culture and impact health. Speak like a professor and only use vocabularies, wordings, etc professors use.

The rationale as to why we chose this particular prompt will be explained in the "Development Process" section.

Additionally, the web application provides a wiki where users can view and contribute information on LLMs and LLM-powered applications.

LLM Fingerprinting

Our process of LLM fingerprinting involves the following two components.

The "Fingerprint Library"

For each LLM in our "base dataset" (covered in depth in "Technical Implementation Summary"), we generated 4 "fingerprints" for the LLM by using its responses to the "identification prompt". We generate these "fingerprints" via the following process:

1. We divide up the LLM’s responses to the “identification prompt” into groupings based on which of the following 3 temperature bins the response falls into:
 - a. Low temp: between 0 inclusive and 0.4 exclusive
 - b. Medium temp: between 0.4 inclusive to 0.8 exclusive
 - c. High temp: between 0.8 and 1.2 inclusive

A response falls into a given temperature bin if the temperature of the LLM when the response was generated is in the range of the temperature bin shown above.

2. For each response within the temperature bins, we take the TF-IDF trigram vector of each response (TF-IDF trigram vectorization details in the “word frequency classifiers” section of “Technical Implementation Summary”) and calculate the average vector of those responses. The average vector for each temperature bin grouping becomes one of the LLM’s “fingerprints”, giving us our “low temp”, “medium temp”, and “high temp” fingerprints.

Here is an example of what this step looks like:

Temperature Bin	Response Vectors	Averaged Response Vector
Low: [0.0, 0.4)	[0.100, ... 0.109, 0.102]	[0.059, ... 0.0985, 0.1085] ("low temp fingerprint")
	⋮	
	[0.018, ... 0.088, 0.115]	
Medium: [0.4, 0.8)	[0.077, ... 0.059, 0.062]	[0.0405, ... 0.0870, 0.0435] ("medium temp fingerprint")
	⋮	
	[0.004, ... 0.115, 0.025]	
High: [0.8, 1.2)	[0.044, ... 0.054, 0.097]	[0.0255, ..., 0.0670, 0.100] ("high temp fingerprint")

	⋮	
	[0.007, ... 0.080, 0.103]	

3. For the fourth “fingerprint”, we use the TF-IDF vector of every single one of the LLM’s responses to calculate the average of those vectors, giving us the 4th “fingerprint” for the LLM.

Here is an example of what this step looks like:

Response Vectors	Averaged Response Vector (“overall fingerprint”)
[0.100, 0.084, ... 0.109, 0.102]	[0.04166, 0.04, ... 0.08416, 0.084]
⋮	
[0.007, 0.085, ... 0.080, 0.103]	

The set of all of these “fingerprints” from each LLM becomes our “library”. This is what that “library” would look like:

	LLM A	LLM B	...
“Low temp fingerprint”	[0.0591, 0.0437, ..., 0.1085]	[0.0103, 0.0998, ..., 0.3091]	[...]
“Medium temp fingerprint”	[0.0401, 0.0188, ..., 0.0435]	[0.1211, 0.1903, ..., 0.0091]	[...]
“High temp fingerprint”	[0.0250, 0.0589, ..., 0.1008]	[0.0322, 0.2210, ..., 0.0818]	[...]


“Overall fingerprint”	[0.0410, 0.0401, ..., 0.08401]	[0.4301, 0.0101, ..., 0.2010]	[...]
-----------------------	--------------------------------	-------------------------------	-------

Identification Procedure

To identify an unknown LLM, first we generate a “fingerprint” for that LLM. Our process of generating a “fingerprint” involves prompting the unknown LLM 14 times with the “identification prompt”. We then vectorize those responses into TF-IDF trigram vectors, and then calculate the average vector of those vectorized responses, which we will use as the “fingerprint” of the unknown LLM.

With the “fingerprint” of the unknown LLM, we then compare that “fingerprint” using cosine similarity to the “fingerprints” of LLMs in our library. Each LLM in our library is then ranked based on the cosine similarity value of its “fingerprint” which scored highest when comparing with the “fingerprint” of the unknown LLM. This is the LLM’s “most similar fingerprint”. The higher the cosine similarity value for the LLM’s “most similar fingerprint”, the higher the LLM is ranked.

Temperature Bins	Library LLM A		Library LLM B		...		Unknown LLM
	Fingerprint	Cosine Similarity to Unknown LLM's Fingerprint	Fingerprint	Cosine Similarity to Unknown LLM's Fingerprint	Fingerprint	Cosine Similarity to Unknown LLM's Fingerprint	Fingerprint
Low	[0.0591, 0.0437, ..., 0.1085]	0.23	[0.0103, 0.0998, ..., 0.3091]	0.65	[...]	...	N/A
Medium	[0.0401, 0.0188, ..., 0.0435]	0.4	[0.1211, 0.1903, ..., 0.0091]	0.34	[...]	...	N/A
High	[0.0250, 0.0589, ..., 0.1008]	0.57	[0.0322, 0.2210, ..., 0.0818]	0.94	[...]	...	N/A
Overall	[0.041, 0.0401, ..., 0.08401]	0.88	[0.4301, 0.0101, ..., 0.2010]	0.7	[...]	...	[0.033, 0.21, ..., 0.084]



LLM Identification Cosine Similarity Ranking		
Rank	LLM	Cosine Similarity
1	LLM B	0.94
2	LLM A	0.88
...

Figure 1: Visualization of the LLM identification process

We demonstrate this procedure with the example shown at Figure 1. Let us call the highest ranked LLM $m1$ and the cosine similarity of its “most similar fingerprint” $s1$. If $s1 \geq 0.65$, then the unknown LLM is identified as $m1$. Otherwise, the unknown LLM is labeled as “unknown”, meaning that we cannot identify it due to the LLM not being in our “library”.

Our web application displays the top 3 highest ranked LLMs and the cosine similarity of their “most similar fingerprint” by default when using our identification functionality. Users can additionally download the full rankings in JSON format like so:

```
{  
  "LLM S": 0.88,  
  "LLM T": 0.79,  
  "LLM U": 0.65,  
  "LLM V": 0.58,  
  ...  
}
```

The keys are the names of the LLM and are in descending order of ranking. The value associated with each key is the cosine similarity of the “most similar fingerprint” for the LLM.

How Fingerprinting Impacts LLM Transparency

The LLM fingerprinting functionality of the web application helps encourage LLM transparency in a few different ways.

For a closed-sourced, LLM-powered application, e.g. a chatbot, we can predict the LLM that is likely generating the text responses. Identifying the LLM allows potential users to look up its capabilities and vulnerabilities. Users can get a sense of what they should

expect from the application, allowing them to decide whether they should continue to use it. Therefore, fingerprinting can supplement users' reviews for an LLM-application, especially in scenarios where there are few or no user reviews available to read. It is not too time-consuming to prompt an LLM 14 times to collect the requisite data for the identification procedure. In addition, being able to identify LLMs like this will help enable policymakers to make regulations on which LLMs can be used and how they can be used. It will also help with enforcing LLM licensing with users of LLMs.

For a given LLM accessible via API, by applying the "identification procedure" (albeit modified so that the LLM is prompted in the same way as when collecting data points for a given LLM and user prompt combination in the base dataset), users can quickly find out which LLMs in our dataset are most similar to the given LLM. Our data analysis (refer to "Library LLM analysis") has shown that model fingerprint similarity comparison can be used to cluster together models that share similar training data, training methods, and architectures. Additionally, for every cluster examined in the analysis, each model within a cluster had a cosine similarity value greater than 0.65 when compared with any of the other models in the group. Hence, if the given LLM can cluster with any of the LLMs in our library, this implies that the LLM likely shares similar architectures, training methods, and training data with the models in that cluster. This can be used to help verify whether the claims about a closed-source LLM's training data, methods, and architecture are likely legitimate or not. Additionally, by looking at the biases and vulnerabilities of the models that the given LLM had clustered with, we can know to be vigilant for these same vulnerabilities and biases in the given LLM. In particular, if the given LLM is likely a fine-tune of one of the models in our library based on our data analysis, we then know that these vulnerabilities and biases are very likely present in the given LLM. If you know which LLM in the library the given LLM is likely a fine-tune of, it will also give some idea of what to expect in terms of the LLM's performance. For example, if the fine-tuned model is based on a model that's known to be terrible at programming tasks, you can reasonably expect the fine-tuned model to also be terrible at coding. Knowing if a model is a fine-tune also helps with enforcing

LLM licensing e.g. if a model's license prohibits passing off a fine-tune of the model as your own model.

Wiki

To solve the problem of a lack of a collaborative central repository of LLM information, we have decided to integrate wiki functionality into the aforementioned web application. Users can create wiki pages containing various information on LLMs, such as their release date, creator, predecessors, e.g. GPT-3.5 is the predecessor of GPT-4, vulnerabilities, and other attributes, e.g. whether the LLM has a mixture-of-expert architecture. Users can also create wiki pages documenting LLM-powered applications e.g. ChatGPT, to track which applications use what LLMs with the help of our LLM fingerprinting functionality.

Data Analysis and Identification Development Process (Babur, Stephen)

In the last iteration of this project, we learned that classical machine learning based classifiers did not perform well since they rely heavily on the set of features that can be found in the training data. Given the success and performance of word frequency based classifiers in the last iteration of the project, we decided to further explore non-parameteric (i.e., does not perform model weight update through training) classifiers. Thus, we decided to limit our experiments to the following two classification approaches; word frequency based and embedding vector based. We explored these classification approaches with the assumption of black-box access to the unknown LLMs that we are trying to identify.

First, we explored existing literature and classification approaches. We examined the approach of Nikolić et al. (2025) and tried to see if its approach can be applied to our use cases. Given that the algorithms employed by Nikolić et al. (2025) relies on a few thousand prompts, we concluded that this approach is not useful in our web application

given that the rate limit of most LLM websites becomes a bottleneck when collecting LLM responses for the user prompts. The classifier employed in the previous iteration of the project also relied on thousands of user prompt and LLM response pairs which makes it difficult to use in production systems.

Thus, we decided to instead utilize user prompts that are more open-ended and are likely to result in the LLMs generating long responses where we at most collect dozens of LLM response data points for each user prompt. Hence, all the datasets used in our experiments were collected using these sort of user prompts.

Classification methods

We explored variations of the two classification approaches above. All of our classifiers share a common process where we convert the raw LLM responses into some sort of response vector where all the entries are numerical values. The main difference of the classifiers we explored is how they create this numerical response vector. Using these response vectors, we aim to construct fingerprints for LLMs which can be compared quantitatively. We compute the averaged response vector for an unknown LLM and use this vector as the unknown model's fingerprint. For our library of known LLMs, we categorize the response vectors into low (within the range $[0.0, 0.4)$), medium (within the range $[0.4, 0.8)$), and high temperature ranges (within the range $[0.8, 1.2)$) and use the averaged response vectors of each bin as fingerprints. Additionally, we compute an overall averaged response vector (based on all response vectors) and also use this as a fingerprint. All the fingerprints we compute can be visualized as Table 1. Thus, we compare the fingerprint of an unknown LLM to the fingerprints of known LLMs that we have stored in our library/database. All the classifiers make predictions by finding a known LLM in our library whose fingerprint is the most similar to the fingerprint of the unknown LLM that we are trying to identify. This known LLM becomes the classifiers' prediction. As for the word frequency approach, we explored raw frequency (using [CountVectorizer from Scikit-Learn](#)) and TF-IDF (using [TfidfVectorizer from Scikit-Learn](#)) methods with both unigrams and trigrams. Additionally, we explored the [potion-multilingual-128M embedding model](#) (using SentenceTransformer). Note that for

all the classifiers, we explored two versions of classifier; one where we used euclidean distance as a comparison metric (when comparing response vectors) and another where we used cosine similarity as a comparison metric.

	Library LLM 1	...	Unknown LLM
Low Temp Bin	[0.059, 0.04, ..., 0.10]	[...]	N/A
Medium Temp Bin	[0.04, 0.01, ..., 0.1]	[...]	N/A
High Temp Bin	[0.02, 0.05, ..., 0.1]	[...]	N/A
Overall	[0.04, 0.04, ..., 0.08]	[...]	[0.0322, 0.0527, ..., 0.0377]

Table 1: Visualization of the all the fingerprints used for LLM identification

Word frequency classifiers

The Scikit-Learn vectorizers are built on the Bag of Words (BoW) model. The BoW model ignores word order and grammar, treats a document as a “bag” (or multiset) of its words, and focuses only on word occurrence and frequency. First, the vectorizers split the document (text string) into a list of words (or sub-words) called tokens, for instance by using white-spaces and punctuation as token separators. Next, the vectorizer analyzes all documents in the training set to create a master list of all unique tokens which is called the vocabulary. The CountVectorizer then simply counts how many times each token in the vocabulary appears in the current document. On the other hand, the TfidfVectorizer takes the counts (raw frequency) from CountVectorizer and transforms them to reflect how important a word is to a document in a corpus. It does this by computing the TF-IDF score which is the product of the Term Frequency (TF) and the Inverse Document Frequency (IDF). The term frequency measures how often a word appears in a specific document while the inverse document frequency measures how rare a word is across all documents. Thus, a high TF-IDF score is achieved when a term is frequent in a specific document but rare in the overall corpus. Therefore, the

TfidfVectorizer outputs a vector which is the result of applying TF-IDF weighting to the raw frequency vector created by CountVectorizer.

Embedding vector classifier

The potion-multilingual-128M is an embedding model developed by Minish Lab that is distilled from the BAAI/bge-m3 Sentence Transformer (embedding model). It is trained on 101 languages and produces 256 dimensional embeddings. It utilizes static embeddings which enables much faster computation on both CPU and GPU than other embedding models of the similar size. Although static embedding models lack contextual awareness, the potion-multilingual-128M model is a static embedding model that was trained to mimic a contextual/non-static embedding model (i.e., the BAAI/bge-m3 model). Thus, the potion-multilingual-128M enables real-time usage in our production system while having better accuracy than other embedding models that offer similar speed and efficiency. However, it generates and outputs 256-dimensional embeddings while still fundamentally being a static embedding model which means that it will not have the best performance when used as part of our classifier.

Datasets

We collected three datasets: the base dataset, the system prompt dataset, and the final classifier (clf) dataset. For each dataset, we collected the LLM response data at varying LLM temperatures ranging from 0.0 to 1.2. We created three LLM temperature bins where the low temperature bin ranges from 0.0 (inclusive) to 0.4 (exclusive), the medium temperature bin ranges from 0.4 (inclusive) to 0.8 (exclusive), and the high temperature bin ranges from 0.8 (inclusive) to 1.2 (exclusive).

Base dataset experiment

The base dataset contains LLM responses for (LLM, user prompt) combinations where we used a set of LLMs containing 36 LLMs across 6 families and a set of 20 distinct user prompts. For each (LLM, user prompt) pair, we collected 12, 24, and 48 data points for the low, medium, and high temperature bins respectively. This dataset was split into

the non-held-out set and the held-out set such that the non-held-out set contains all the data for 30 LLMs and the held-out set contains all the data for the remaining 6 LLMs. We then split the non-held-out set into train and test sets using a split ratio of 5:1 while stratifying by the combinations (LLM, user prompt, temperature bin). Lastly, we performed a 5-fold cross validation on the train set where all the 4 train splits were used as the library of the known LLMs and the single validation split was treated as the set of unknown LLMs we are trying to identify. The results were as follows:

z

Classification Method	Top-1 Identification Accuracy ↑	Top-3 Identification Accuracy ↑
raw_freq_unigram_cosine	0.959	0.998
raw_freq_unigram_euclidean	0.959	0.998
tfidf_unigram_cosine	0.960	1.0
tfidf_unigram_euclidean	0.960	1.0
tfidf_trigram_cosine	0.968	1.0
tfidf_trigram_euclidean	0.968	1.0
potion_emb_cosine	0.962	0.998
potion_emb_euclidean	0.962	0.998

Table 2: 5-fold cross validation results on the train set of base dataset

As we can observe from Table 2, all of the classifiers achieved a near-perfect performance while also having very similar metric values. This 5-fold cross validation on the train set (of the base dataset) does not allow us to distinguish the classifiers and get a good understanding of their performance differences. Hence, we need to construct a

more difficult dataset such that our classifiers would have a noticeable difference in their performance. This is one of the motivations for constructing and experimenting on the system prompt dataset.

System prompt dataset

The system prompt dataset contains LLM responses for (LLM, user prompt, system prompt, neutralization technique) combinations where we used a set of LLMs containing 17 LLMs across 6 families and a set of 20 distinct user prompts. The set of 17 LLMs used here is a subset of the set of LLMs used for the base dataset while the set of user prompts is the same as the one used for the base dataset. For each (LLM, user prompt, system prompt, neutralization technique) pair, we collected 2, 4, and 8 data points for the low, medium, and high temperature bins respectively. Note that the neutralization techniques are applied to the base user prompts as an attempt to minimize the influence of the system prompt on the LLM responses. The behavior and response style of LLMs can change drastically depending on the system prompt used for the LLM. Additionally, most LLMs used in production systems use system prompts to control the behavior of the LLMs. To be specific, system prompts are usually used to ensure ethical usage of the LLM, to ensure the LLM provides responses for specific topics or fields, to prevent misuses of the LLM, etc. Since our classifiers rely on the similarity of the textual features of the LLM responses to make predictions, it is crucial for our classifiers to be able to make accurate predictions on LLM responses regardless of the system prompt used with the LLM. Thus, we collected the system prompt dataset to compare and determine the best classifier method, to determine the most discriminative (i.e., best) user prompt, and the most effective system prompt neutralization technique. We explored 5 neutralization techniques: none, multiple response styles, linguistic uniqueness, localization, and Best-of-N text augmentation. The “none” technique means that we don’t apply any technique to the base user prompt. The “multiple response styles” technique is where we prepend the base user prompts with a short text asking to respond in three different styles. The “linguistic uniqueness” technique is where we prepend the base user prompts with a short text asking to prioritize its linguistic uniqueness over everything else. The “localization” technique is where we prepend the

base user prompts with a short text asking to respond normally and also in using linguistic patterns common in a country that is randomly selected from a set (the set contains 10 countries across all continents). The “Best-of-N text augmentation” technique randomly swaps words, randomly swaps characters in words, randomly capitalizes characters in words, and randomly alters characters by adding or subtracting 1 from the ASCII index.

System prompt dataset experiment

The entire system prompt dataset was treated as the set of unknown LLMs we are trying to identify whereas the entire training set of the base dataset was used as the library of known LLMs and their response vectors. The table below summarizes the performance of the classifiers across all the user prompts, system prompts, and neutralization techniques.

Classification Method	Top-1 Identification Accuracy ↑	Top-3 Identification Accuracy ↑
raw_freq_unigram_cosine	0.2389	0.4069
raw_freq_unigram_euclidean	0.1824	0.3061
tfidf_unigram_cosine	0.2989	0.4567
tfidf_unigram_euclidean	0.1968	0.3399
tfidf_trigram_cosine	0.3141	0.4881
tfidf_trigram_euclidean	0.2252	0.3685
potion_emb_cosine	0.2033	0.3854
potion_emb_euclidean	0.1967	0.3821

Table 3: System prompt dataset experiment - overall classifier performance results

As we can observe from Table 3, the `tfidf_trigram_cosine` classifier has a much better performance compared to the other classifiers. (Add a bit more detailed analysis later). Next, the following table contains the average effectiveness of each system prompt neutralization technique across all classifiers, user prompts, and system prompts.

Neutralization Technique	Accuracy \uparrow	Variance \downarrow	Count
none	0.321838	0.218260	190400
response styles	0.253382	0.189181	190400
localization	0.251471	0.188234	190400
linguistic uniqueness	0.221544	0.172463	190400
best-of-n text augmentation	0.111985	0.099445	190400

Table 4: Average neutralization technique effectiveness

First, note that the “count” column in Table 4 is the total number of evaluations(i.e., response data) that were processed to compute the accuracy across all classifiers. The “variance” column represents the variance of the correct predictions across all classifiers, user prompts, and system prompts. We can observe that applying no neutralization technique (i.e., the “none” technique) leads to the highest accuracy and hence is the most effective technique. We believe that the “response styles” and “localization” techniques do not work as well as the “none” technique since they result in LLM responses that are longer and much more diverse in terms of the raw frequency and rareness of words that appear in the responses. It would be interesting to use these two neutralization techniques as a part of a base user prompt and measure their effectiveness as a base user prompt instead of a neutralization technique. The “linguistic uniqueness” technique basically tries to make the LLMs behave in a certain manner simply by asking in a longer, and somewhat abstract manner. This technique not performing well suggests that the LLMs follow their system prompts quite faithfully

and also that the LLMs likely do not understand their own linguistic uniqueness, habits, and patterns. Lastly, the “best-of-n text augmentation” technique performs the worst. While the larger and more powerful LLMs were able to understand and heavily modified user prompts, the smaller and less powerful LLMs misunderstood the user prompts and their responses were about an unrelated topic. Additionally, most of the smaller LLMs struggled to output longer responses likely due to getting confused by the random and noisy nature of the words and characters in the user prompt. While the “best-of-n text augmentation” technique did not perform well, it is worth trying this technique with lower probabilities for the random character swaps, random character capitalizations, and random word swaps. Next, the following table contains the average prediction accuracy for each user prompt across all classifiers, system prompts, and neutralization techniques.

Base User Prompt	Accuracy ↑	Variance ↓	Count
Make a guess about what the treasure “one piece” ... Speak like a mathematician ...	0.404706	0.240924	47600
Invent a new taste unknown to humans ... Speak like a professor ...	0.399412	0.239887	47600
Imagine a world where gravity pulls sideways ...	0.270588	0.197374	47600
...
Invent a new taste unknown to humans ...	0.252353	0.188675	47600
Make a guess about what the treasure “one piece” ...	0.240000	0.182404	47600

...
Animaliek hauteskundeetan bozkatu ahal ... (Swahili)	0.166176	0.138565	4760 0
Imagine a world where human emotions can be recorded and traded ...	0.163235	0.136592	4760 0

Table 5: Average user prompt discriminativeness

Note that Table 5 contains the top 3 and bottom 2 (by accuracy) user prompts and also the other variations of the top 2 base user prompts. First, note that top 2 user prompts have a much higher accuracy than all the other base user prompts (including the user prompt with the third highest accuracy of 0.270588). Additionally, note that the top 2 user prompts have another version where they don't enforce any speaking styles. Whenever we enforce a speaking style on a user prompt, it improves the accuracy by approximately 15% which is a noticeable improvement. The reason for this improvement might be that enforcing speaking styles leads to the base user prompt becoming more specific and resulting in the LLMs to answer an open-ended question in a very constrained, restricted and specific manner. Since the base user prompts are all open-ended, it is likely that adding some sort of constraint (e.g., enforcing a certain speaking style) assigns a new and important task or persona to the LLM that can mitigate the effect of the system prompt on the LLM responses. When we consider the accuracy of the top 2 base user prompts for the most effective neutralization technique (i.e., "none") and for the best classifier method (i.e., `tfidf_trigram_cosine`), they both still have very similar accuracies. We ended up using the prompt "Invent a new taste unknown to humans ... Speak like a professor ..." as our single best user prompt but the prompt "Make a guess about what the treasure "one piece" ... Speak like a mathematician ..." should also give very similar performance in the next experiment.

Final CLF Dataset

The final CLF dataset contains LLM responses for (LLM, best user prompt, system prompt) combinations where we used a set of LLMs containing 36 LLMs across 6 families and a set of 15 distinct system prompts. The set of LLMs is the same as the one used for the base dataset while the set of system prompts is different from the one used for the system prompt dataset. For each (LLM, best user prompt, system prompt) pair, we collected 4, 8, and 16 data points for the low, medium, and high temperature bins respectively. Then, we split the dataset into the non-held-out set (containing full data for 30 LLMs which are the same exact 30 LLMs that are used in our library) and the held-out set (which contains full data for 6 LLMs). Next, we split both sets into half (i.e., using split ratio 1:1) while stratifying by the (LLM, best user prompt, system prompt, temperature bin) combinations. Hence, the entire final CLF dataset is split into 4 sets: non-held-out tuning set, held-out tuning set, non-held-out test set, and held-out test set. While we determined the best classifier (i.e., `tfidf_trigram_cosine`), the best user prompt (i.e., “Invent a new taste unknown to humans ... Speak like a professor ...”) and the best neutralization technique (i.e., “none”), our classifier needs to use a threshold on the cosine similarity values to make the decision whether it should actually make a prediction (i.e., selecting one of the known LLMs in the library as the prediction) or it should predict “unknown”. In other words, if the highest cosine similarity value is greater than or equal to the threshold, our classifier should make predictions based on the LLMs in our library. Otherwise, it should predict “unknown” if the highest cosine similarity value is less than the threshold. This process is crucial since otherwise the classifier will always wrongly make predictions on unseen LLMs whose data is not in our library of known LLMs. Thus, our classifier should be able to accurately predict seen LLMs (i.e. LLMs whose data is in our library) while it should also accurately predict “unknown” for unseen LLMs (this is called out-of-distribution detection). Thus, the 2 tuning sets are used to evaluate the selected/best classifier (i.e., `tfidf_trigram_cosine`) on various threshold values (0.05, 0.10, 0.15, ..., 0.95) to search for the optimal threshold value which maximizes the prediction accuracy on the non-held-out set while also maximizing the OOD detection accuracy on the held-out set.

Final CLF Dataset Experiment

Figure 2 below visualizes the result of the final CLF dataset experiment. It is based on the best classifiers performance on the non-held-out tuning set and held-out tuning set.

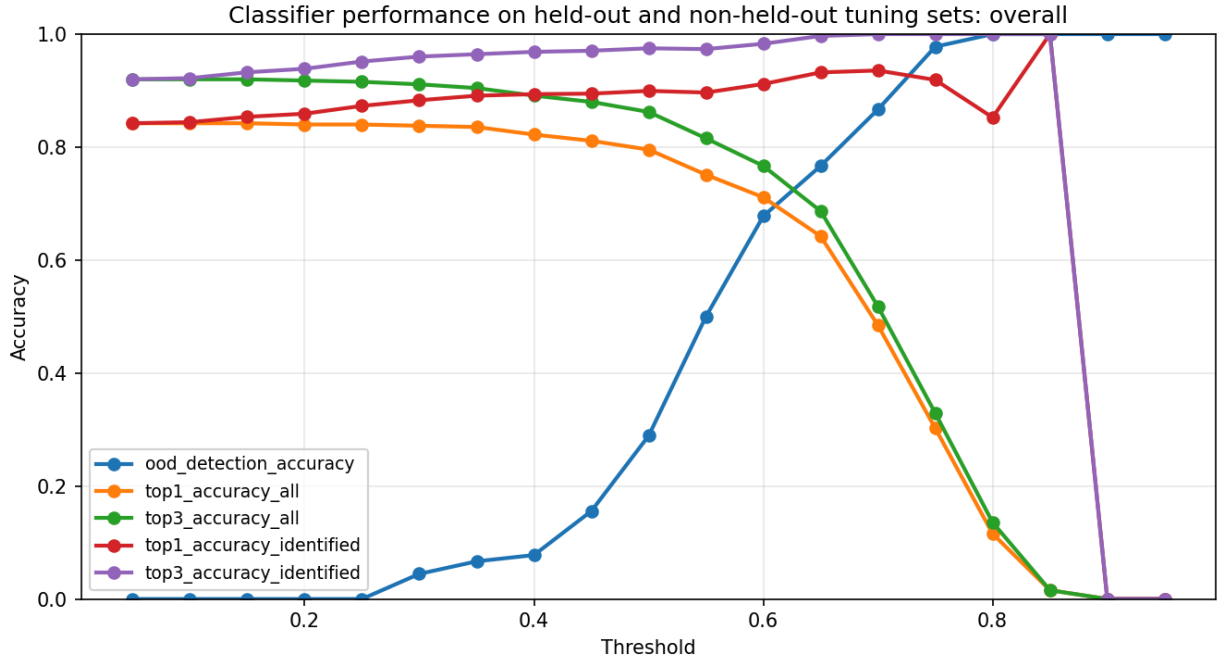


Figure 2: Best classifier’s performance on non-held-out and held-out tuning sets

As we can observe from Figure 2, setting the threshold to a larger value will improve the LLM identification accuracy for the models where the classifier actually makes predictions. However, a higher threshold value leads to the classifier only making predictions for the unknown LLMs that it is quite confident about. As a result, the classifier predicts “unknown” for models that are actually in the library. Thus, a higher threshold value leads to a sharp drop in the overall identification accuracy (including “unknown” predictions). The threshold value 0.65 strikes a good balance and achieves high identification accuracy as well as high OOD detection accuracy so we decided to use this value as the threshold value for our classifier. Using this selected threshold value, the performance of our final version of the selected classifier (i.e., the version of the classifier that will be used in the production system) is summarized in Table 6 and Table 7 below.

Metric	overall	refusal	non_refusal
top1_accuracy_all ↑	0.6378	0.0357	0.6777
top3_accuracy_all ↑	0.6822	0.0357	0.7251
top1_accuracy_identified ↑	0.9199	1.0	0.9196
top3_accuracy_identified ↑	0.984	1.0	0.9839
unknown_rate	0.3067	0.9643	0.263
misidentification_rate ↓	0.0556	0	0.0592
identified_error_rate ↓	0.0801	0	0.0804
total_predictions	450	28	422
total_identified	312	1	311

Table 6: Final classifier's performance on non-held-out test set

Metric	overall	refusal	non_refusal
top1_family_accuracy_identified ↑	1.0	0.0	1.0
top3_family_accuracy_identified ↑	1.0	0.0	1.0
top1_branch_accuracy_identified ↑	0.3333	0.0	0.3333
top3_branch_accuracy_identified ↑	0.4444	0.0	0.4444
top1_family_accuracy_all ↑	0.9556	0.6	0.9765
top3_family_accuracy_all ↑	0.9667	0.6	0.9882

top1_branch_accuracy_all ↑	0.4667	0.4	0.4706
top3_branch_accuracy_all ↑	0.6444	0.4	0.6588
ood_detection_accuracy ↑	0.8	1.0	0.7882
total_predictions	90	5	85
total_identified	18	0	18

Table 7: Final classifier’s performance on held-out test set

As we can observe from Table 6 and 7, our classifier achieves high LLM identification accuracy, LLM family identification accuracy, and OOD detection accuracy. Thus, it can be said that the classifier is very accurate when it makes predictions. Additionally, if an unknown LLM is not in the library, then the classifier detects this accurately (given the 0.8 OOD detection accuracy) and avoids wrongly making predictions for unseen LLMs that are not in the library. However, the branch accuracy is nowhere as high as the other metrics. This happens because even within the LLM branch, the size of the models can differ significantly which leads to LLMs within the branch having dissimilar fingerprints. Additionally, it can be observed from Table 6 that the classifier has really low identification accuracy for LLMs where their responses are refusal responses (i.e., responses where the LLMs refuse to answer the user prompt due to their policy set through the system prompt). Predicting “unknown” (i.e., out-of-distribution) for almost all of the refusal responses in the non-held-out set shows room for improvement in our classifier. While we can add more fingerprints to the library where the fingerprints are based on the refusal responses of each LLM, this might not be the most robust approach since different system prompts can trigger different styles in the refusal responses. We discuss further extensions for the classifier in the Recommendations and Future Work section that aims to tackle these two shortcomings of the current classifier.

Library LLM analysis

The dendrogram in Figure 3 visualizes the hierarchical clustering of the LLMs in the library. The hierarchical clustering algorithms start with each item (i.e., LLM) as its own cluster and progressively merge the most similar items until all the items become part of a single large cluster. The y-axis of the dendrogram quantifies dissimilarity where smaller values indicate closer (more similar) LLMs. We used one minus the cosine similarity value of an LLM’s “overall fingerprint” to another LLM’s “overall fingerprint” as the dissimilarity metric (linkage distance). Additionally, different colors are used to highlight distinct clusters. Analyzing the dendrogram, we can see a number of patterns emerge.

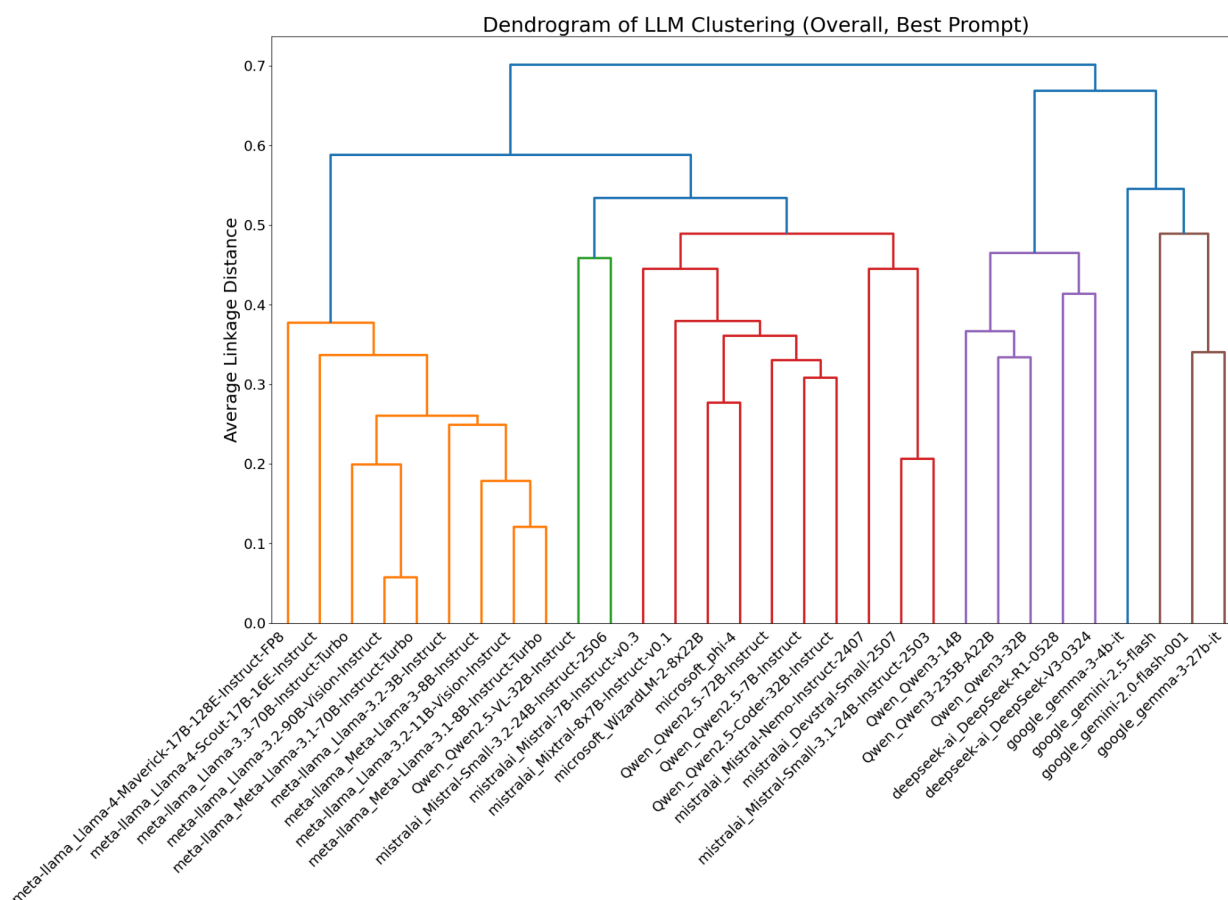


Figure 3: Dendrogram visualizing the hierarchical clustering of library LLMs

Impact of Model Training Data, Training Processes, and Architecture on Model Grouping

We can see that the Qwen2.5 (except Qwen2.5 VL), Qwen3, and Llama 3.x models each form distinct clusters in the dendrogram. Examining these clusters (Llama 3.x, Qwen3, Qwen2.5) further, we can see that each model in a cluster has a cosine similarity value greater than 0.65 when comparing with any other model in the group, based on the data in

response_classifier/results/library_data_analysis/inter_llm_similarity_matrix_overall.txt.

The LLMs in each of these clusters had similar training processes (at least for the process of responding to text prompts) and similar architectures compared to other models in their cluster. They also shared training data (or at least likely did for the Llama 3.x models) with other models in their cluster (Meta AI, 2024a; Meta AI, 2024b; Meta AI, 2024c; Meta AI, 2024d; Yang et. al, 2025a; Yang et. al, 2025b). In contrast, the Qwen2.5-VL model is very distant from both the other Qwen2.5 and Qwen3 models, and utilized weights from Qwen2.5 (that are also then used to process text input) that were then modified when the model was trained on a wide variety of multimodal image data, alongside adding architectural features to support image and video inputs (Bai, 2025). Meanwhile, the Llama 3.2 vision models add a vision adapter on top of a Llama 3.1 model, with no modification made to the Llama 3.1 LLM parameters of the Llama 3.2 models outside of fine-tuning during the post-training phase (Meta AI, 2024c). Thus, these results imply that model fingerprint similarity can be used to group together models which likely share similarities in their training data, training processes, and architecture.

Examining the dendrogram more closely, we can observe that the Llama 3.2 vision models are each very similar to their closest-sized Llama 3.1 model. The 90B vision model has a similarity score of 0.94 when comparing with Llama 3.1 70B Instruct Turbo, and the 11B vision model has a similarity score of 0.87 when comparing with meta-llama_Meta-Llama-3.1-8B-Instruct-Turbo, based on the data in *response_classifier/results/library_data_analysis/inter_llm_similarity_matrix_overall.txt.* The other Llama models, which the Llama 3.1/3.2 models were not directly derived from

and are not fine-tuned from the the Llama 3.1/3.2 models, max out at a similarity score of 0.84 when comparing with the LLM most similar to them.

The Llama 3.2 vision models, in terms of text functionality, are effectively just a fine-tune of their corresponding Llama 3.1 models in our library. This is because the Llama 3.1 LLM parameters used for the Llama 3.2 models were only modified during the fine-tuning phase of creating the Llama 3.2 models. Thus, these results imply that when two models have a cosine similarity score greater than 0.85 when comparing “overall fingerprint”, this means that one of the models is likely a fine-tune of the other.

Impact of Thinking Process on Model Grouping

Examining the dendrogram, we can see that the Qwen2.5 (non-VL) models (which are all non-thinking) and Qwen3 models (which are all thinking) are separated into distant clusters despite belonging to the same “Qwen” family of models. In fact, the Qwen3 models are more closely clustered with the Deepseek models, which are also thinking (DeepSeek R1) or at least support a thinking mode (DeepSeek V3; its thinking mode wasn’t used during data collection) (Deepseek AI, 2025). Compared to the distance between the Qwen2.5 (non-VL) and Qwen3 clusters, the Llama 4 models are more closely clustered with the Llama 3.x models, with neither the Llama 4 or Llama 3.x models being “thinking” models (Meta AI, 2025). In addition, when looking at *response_classifier/results/library_data_analysis/inter_llm_similarity_matrix_overall.txt*, the cosine similarity value (0.586116) when comparing DeepSeekV3 (which was prompted in non-thinking mode) to DeepSeek R1 (which always has a thinking process when prompted) is noticeably lower than the cosine similarity value when comparing either Llama 4 model to the three most similar Llama 3.x. The results imply that the process of “thinking” for LLMs and the capacity to “think” has a significant influence on LLM response writing style, and that model fingerprint similarity can be used to group together models that specifically possess “thinking” capabilities.

Ethical, Social, or Community Implications (Zayd, Sani, Ahmed)

Our project “fingerprints” LLMs to encourage their safe and fair usage. As a result, we believe the primary implication of our project will be ethical: as users accurately identify the LLMs powering different services, these services will be motivated to utilize LLMs that were tuned for safety and privacy concerns. We discuss these ethical implications to show the impact of our project on general LLM-usage.

The mainstreaming of LLMs is viewed by some as akin to the “transistor moment”. Many businesses are pushing to incorporate LLMs into their workflows. Similarly, many developers are motivated to learn how to develop meaningful applications with LLMs. However, the sparks of such monumental change can cloud the ethical implications surrounding LLM use. As shown by several of our wiki entries, LLMs can be prone to prompt injection attacks (composing malicious prompts to extract or amend sensitive data) and generating harmful content. These vulnerabilities need to be taken into account when creating LLM-powered applications. Moreover, some LLM service endpoints (APIs) have raised cautionary flags around their not-so-private privacy policy. DeepSeek, for example, clearly states that “we may collect your text input, prompt, uploaded files, feedback, chat history, or other content that you provide to our model and Services” (2025). This implies that business-critical data should not be sent to their APIs. Whether it’s safety or privacy concerns about user data, our project operates to educate users on the various concerns associated with LLM services.

Since our project offers a way to identify LLMs, we believe users will feel empowered to understand more about the LLMs powering the services they currently use. With this information, users can demand changes to LLM-based applications that will protect them from harmful responses, and protect the privacy of their data. This can transform how LLM-based applications like conversational agents, AI notetakers and transcribers engage with users, as they transition towards open guidelines about their LLM usage. Holistically, our open-source project can help users find out, demand and change how

LLM services operate and use their data. Further, our project being open-source allows users to verify our methods, building trust and accountability. This also allows for collaboration, encouraging improvement on fingerprinting methods as the world of LLM design and tuning continues to evolve.

Recommendations and Future Work (Ahmed, Mark, Babur, Stephen)

Recommendations

As an open-source project, there are a variety of recommendations that can be implemented to transform it into a full-fledged community-driven web-app. To ensure that a multitude of users share a similar experience accessing the project, we focus on recommendations that build upon general user accessibility. Initially, we discuss how accessibility can be improved to accommodate visually impaired and colorblind users. Moreover, we recommend the addition of a set of user-guides to improve the experiences of public contributors.

Accessibility

Accessibility is an integral part of user experience (UX). Especially for open-source projects, it's important to serve all users - who share different capabilities - with a similar UX. As stated by W3C Web Accessibility Initiative (WAI):

Accessibility is essential for developers and organizations that want to create high-quality websites and web tools, and not exclude people from using their products and services (2024).

With this sentiment in mind, we recommend improvements to accessibility through 3 key areas: the web-app's color palette, the web-app's behaviour at different zoom-levels, and the web-app's level of integration with screen-readers.

To begin with, the web-app's color palette can be adjusted to better cater towards color-blind individuals. As shown by Sajek et al., "colour in interface designs is crucial for ensuring high-quality usability" (2025). By offering users a simple way to configure the *theme* of our web-app, we can enhance how users access, develop with and understand the results of our LLM-identification service.

Additionally, another way our project can improve its accessibility is by making the web-app responsive at different zoom levels. That is, individuals who are visually impaired may need to increase the level of a page's zoom. As practically explained by W3S, "[s]ome people just need to enlarge the text a little if they forgot their reading glasses" (2024). Currently, our web-app - especially the CSS animations - may fail to display information around zoom levels of 200% or larger. Therefore, improving the zoom experience is an integral step as the project progresses forward.

Finally, by improving the semantic nature of our HTML, we can offer better integration with screen-readers. In a Google guide about *Semantics and Screen Readers*, Rob Dodson states that technologies like screen-readers depend on "developers marking up their pages with semantic HTML" (2018). As such, augmenting our pages with semantically meaningful HTML helps ensure that users can effectively use screen-readers to traverse our web-app.

User Guides

Furthermore, as we continue to develop the open-source project, it will become increasingly important to facilitate the contributions of code and documentation. That is, users who are not initially associated with the project should find it easy to contribute features to the web-app, along with information about existing LLMs and LLM-apps. So, we recommend the addition of a set of guides that aid users in discovering the different subset of the project. Guides can explore how users can contribute code, identify unknown models and edit wiki entries relating to LLMs and LLM-apps.

LLM Fingerprint and Identification Tool

It can be observed from Table 6 and 7 that the LLM branch identification accuracy is nowhere as high as the other metrics (LLM identification accuracy, LLM family identification accuracy, OOD detection accuracy, etc). Thus, to improve the LLM branch identification accuracy, we recommend creating 4 new fingerprints for each LLM branch based on the 4 fingerprints of the known LLMs in the library that belongs to the branch. Additionally, it can be observed from Table 6 that the classifier has really low identification accuracy for LLMs where their responses are refusal responses (i.e., responses where the LLMs refuse to comply with the user prompt). While we can add more fingerprints to the library where the fingerprints are based on the refusal responses of each LLM, this might not be the most robust approach since different system prompts can trigger different styles in the refusal responses. Thus, more experimentation and research should be done to find a way to match open-ended prompts (such as our identification prompt) to LLMs or LLM-based applications so that the context of the user prompt is within the topics that are supported by the LLM or the LLM-based application (and their system prompts). Creating a process of picking an identification prompt based on the use cases of the LLM or LLM-based application should decrease the number of refusal responses which will allow our classifier to actually make predictions and not specify it as “unknown”. In addition, we should collect responses from many more models which are fine-tunes of other models and conduct an experiment with that data in order to calculate a more precise threshold for when a model is similar enough to another model to be considered a fine-tune to improve our ability to specifically identify fine-tunes.

Next Steps

Following through with our recommendations, we lay down some groundwork on how accessibility and user-guides can be practically implemented in our project.

Accessibility: Colors

Starting with accessibility, we aim to include configurable UI color-palette options that accommodate color-blind individuals. To do so, we aim to develop color-themes that feature *triadic colours*. As shown by Sajek et al., triadic colour palettes “are the most versatile and practical option for all types of colour blindness” (2025). A triadic color palette consists of 3 colours that are equally spaced from each other on a color-wheel (Figma, n.d.). They’re versatile because they can be adjusted into many different color combinations. Triadic color-palettes are also practical due to their bold colors and high contrast.

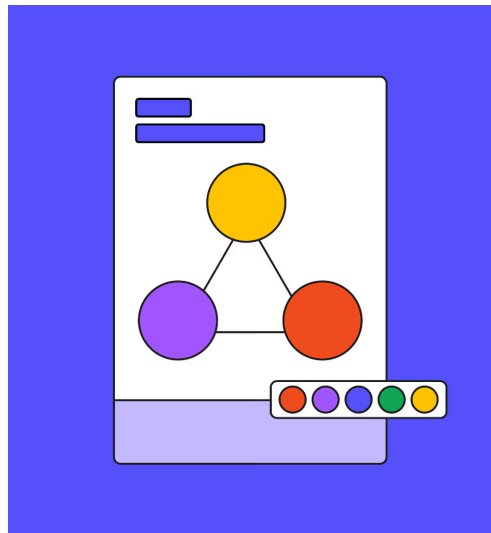


Figure 3: Triadic Colors (Source: Figma, n.d.)

To generate a triadic palette, we begin by choosing our primary color - our web-apps main theme color - to be purple. Then, we will generate the rest of the color scheme by opting for 2 more colors that are equally spaced from purple. As such, the generated triadic color palette is shown below.

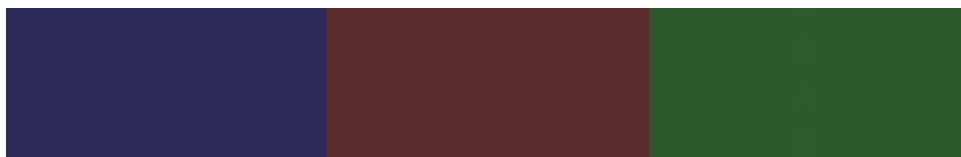


Figure 4: Triadic Color Palette, with Purple as the Primary Color

Thus, by adding the aforementioned color-palette as a configurable theme for our project, we can ensure that individuals with varying color-blindness can comfortably access and utilize our LLM-identification services.

Accessibility: Semantic HTML

Adding semantics to our web-pages is a simple, yet effective way of improving the performance of screen-readers on our web-app. Semantic HTML involves using semantically meaningful HTML tags like `<main>`, `<article>` and `<aside>` to show how the different bits of a webpage relate to each other. In our case, this will involve replacing many of our `<div>`'s with their semantic equivalent. By beginning with the homepage and moving towards secondary pages, we can add semantics to our web-apps in a practical and relevant manner.

User Guides: GitHub Pages

User guides are critical for allowing developers to understand, access and contribute to our project. As a mainly static medium, user-guides can be hosted as a static web-app. Moreover, user-guides should also be hosted with a cloud-hosting solution that is both easily accessible and amenable. Thus, GitHub Pages emerges as a widely-accessible, static web-app hosting service. By composing a series of markdown pages, we can host various guides that detail the LLM-identification and wiki services that our project offers. To append or amend any of the guides, users can expectedly submit pull-requests to do so.

As such, in terms of user-guides, our next step would be to create a GitHub repository containing a series of Markdown user-guides that detail our main services. As a GitHub Pages application, we will look to continuously add more guides as features are integrated into our project.

Expanding LLM library

We should work towards collecting response data from many more LLMs and adding them to our library of LLMs in order to make the library much more comprehensive. This

would allow us to identify many more LLMs, significantly improving the utility of our LLM fingerprinting process.

Continued Exploration

A key area of user experience that we haven't explored is community forums. Many of the features in our project cater towards developers - individuals that develop or research with LLMs. Subsequently, our project can better serve non-developers and general LLM enthusiasts by allowing user-comments, Q&A's and general community posts about the current use-cases of LLMs. This helps us engage with users from a variety of backgrounds while encouraging the cataloguing and safe usage of LLMs.

References

- Bai, S., Chen, K., Liu, X., Wang, J., Ge, W., Song, S., Dang, K., Wang, P., Wang, S., Tang, J., Zhong, H., Zhu, Y., Yang, M., Li, Z., Wan, J., Wang, P., Ding, W., Fu, Z., ... Lin, J. (2025). *Qwen2.5-VL technical report*. arXiv.
<https://arxiv.org/abs/2502.13923>
- Bommasani, R., Li, K., Yan, A., Hudson, D. A., Holla, R., Krass, M., ... Liang, P. (2023, October 19). The foundation model transparency index. arXiv.
<https://arxiv.org/abs/2310.12941>
- Data Science Society. (2024, November 27). Enhancing AI transparency: The role of LLM observability in data science. Data Science Society.
<https://www.datasciencesociety.net/enhancing-ai-transparency-the-role-of-llm-observability-in-data-science/>
- Deepseek AI. (2025, March 27). *deepseek-ai/DeepSeek-V3*. HuggingFace.
<https://huggingface.co/deepseek-ai/DeepSeek-V3>
- Dodson, R. (2018, November 18). Semantics and screen readers. web.dev. Google.
<https://web.dev/semantics-and-screen-readers/>
- Figma. (n.d.). What are triadic colors? Figma. Retrieved August 30, 2025, from
<https://www.figma.com/resources/learn-design/what-are-triadic-colors/>
- Hangzhou DeepSeek Artificial Intelligence Co., Ltd. (2025, July 4). *DeepSeek privacy policy*. DeepSeek. <https://www.deepseek.com/privacy>
- Hugging Face, Inc. (n.d.). Hugging Face. In Wikipedia. Retrieved August 31, 2025, from
https://en.wikipedia.org/wiki/Hugging_Face
- Liao, Q. V. (2023, September 20). AI transparency in the age of large language models: A human-centered research roadmap. Human-Centered AI. Medium.
<https://medium.com/human-centered-ai/ai-transparency-in-the-age-of-large-language-models-a-human-centered-research-roadmap-19bd3a55914a>
- Meta AI. (2024a, April 18). *Meta-Llama-3-8B-Instruct* [Large language model]. Hugging Face. <https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct>
- Meta AI. (2024b, July 23). *Llama-3.1-8B-Instruct* [Large language model]. Hugging Face. <https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct>

Meta AI. (2024c, September 25). *Llama 3.2: Connect 2024 vision: Edge and mobile devices*. Meta.

<https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/>

Meta AI. (2024d, December 6). *Llama-3.3-70B-Instruct* [Large language model].

Hugging Face. <https://huggingface.co/meta-llama/Llama-3.3-70B-Instruct>

Meta AI. (2025, July 23). *The llama 4 herd: The beginning of a new era of natively multimodal AI Innovation*. Meta.

<https://ai.meta.com/blog/llama-4-multimodal-intelligence/>

Mitchell, M., Wu, S., Zaldivar, A., Barnes, P., Vasserman, L., Hutchinson, B., ... Gebru, T. (2019, January). Model cards for model reporting. Proceedings of the Conference on Fairness, Accountability, and Transparency (FAT* '19), 220–229. ACM. <https://doi.org/10.1145/3287560.3287596>

Nikolić, I., Baluta, T., & Saxena, P. (2025). Model Provenance Testing for Large Language Models (arXiv:2502.00706). arXiv.

Sajek, D., Korotenko, O., & Kyrychok, T. (2025). Research on the Accessibility of Different Colour Schemes for Web Resources for People with Colour Blindness. *Journal of Imaging*, 11(8), 268. <https://doi.org/10.3390/jimaging11080268>

W3C Web Accessibility Initiative (WAI). (2024, March 7). Introduction to web accessibility. In S. L. Henry (Ed.), *Education and Outreach Working Group (EOWG)*. World Wide Web Consortium (W3C).

<https://www.w3.org/WAI/fundamentals/accessibility-intro/>

W3C Web Accessibility Initiative (WAI). (2024, March 19). Easy checks – Zoom. In K. White (Ed.), *Education and Outreach Working Group (EOWG)*. World Wide Web Consortium (W3C). <https://www.w3.org/WAI/test-evaluate/easy-checks/zoom/>

Yang, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Li, C., Liu, D., Huang, F., Wei, H., Lin, H., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Yang, J., Zhou, J., Lin, J., ... Qiu, Z. (2025a). *Qwen2.5 technical report*. arXiv.

<https://arxiv.org/abs/2412.15115>

Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., Zheng, C., Liu, D., Zhou, F., Huang, F., Hu, F., Ge, H., Wei, H., Lin, H., Tang, J., Yang, J., ... Qiu, Z. (2025b). *Qwen3 technical report*. arXiv.

<https://arxiv.org/abs/2505.09388>