information

MDPI

*Article*

# Benchmarking 21 Open-Source Large Language Models for Phishing Link Detection with Prompt Engineering

**Arbi Haza Nasution [1,*], Winda Monika [2], Aytug Onan [3], Yohei Murakami [4]**

[1] Department of Informatics Engineering, Universitas Islam Riau, Pekanbaru 28284, Indonesia
[2] Department of Library Information, Universitas Lancang Kuning, Riau 28266, Indonesia; windamonika@unilak.ac.id
[3] Department of Computer Engineering, College of Engineering and Architecture, Izmir Katip Celebi University, Izmir 35620, Turkey; aytug.onan@ikcu.edu.tr
[4] Faculty of Information Science and Engineering, Ritsumeikan University, Kusatsu 525-8577, Japan; yohei@fc.ritsumei.ac.jp
* Correspondence: arbi@eng.uir.ac.id

**Abstract:** Phishing URL detection is critical due to the severe cybersecurity threats posed by phishing attacks. While traditional methods rely heavily on handcrafted features and supervised machine learning, recent advances in large language models (LLMs) provide promising alternatives. This paper presents a comprehensive benchmarking study of 21 state-of-the-art open-source LLMs—including Llama3, Gemma, Qwen, Phi, DeepSeek, and Mistral—for phishing URL detection. We evaluate four key prompt engineering techniques—zero-shot, role-playing, chain-of-thought, and few-shot prompting—using a balanced, publicly available phishing URL dataset, with no fine-tuning or additional training of the models conducted, reinforcing the zero-shot, prompt-based nature as a distinctive aspect of our study. The results demonstrate that large open-source LLMs ($\geq$27B parameters) achieve performance exceeding 90% F1-score without fine-tuning, closely matching proprietary models. Among the prompt strategies, few-shot prompting consistently delivers the highest accuracy (91.24% F1 with Llama3.3_70b), whereas chain-of-thought significantly lowers accuracy and increases inference time. Additionally, our analysis highlights smaller models (7B–27B parameters) offering strong performance with substantially reduced computational costs. This study underscores the practical potential of open-source LLMs for phishing detection and provides insights for effective prompt engineering in cybersecurity applications.

**Keywords:** large language models; phishing link detection; prompt engineering



Academic Editor: Heming Jia

Received: 31 March 2025
Revised: 23 April 2025
Accepted: 29 April 2025
Published: 29 April 2025

**Citation:** Nasution, A.H.; Monika, W.; Onan, A.; Murakami, Y. Benchmarking 21 Open-Source Large Language Models for Phishing Link Detection with Prompt Engineering. *Information* **2025**, *16*, 366. https://doi.org/10.3390/info16050366



## 1. Introduction

Phishing attacks remain one of the most pervasive cybersecurity threats, causing significant financial and data losses worldwide. For example, recent reports estimate that 3.4 billion phishing emails are sent each day, and phishing-related data breaches cost, on average, around $4.9 million per incident [1]. Malicious URLs embedded in emails or messages are a primary vector for these attacks, luring users into disclosing credentials or downloading malware. Detecting phishing URLs quickly and accurately is thus of paramount importance to protect individuals and organizations.

Traditional phishing detection approaches have relied on machine learning models trained on handcrafted features extracted from URLs and websites (e.g., domain reputation,

URL length, and the presence of certain keywords). These classical models—such as Decision Trees, Naive Bayes, and Random Forests—have achieved high accuracy on curated datasets (often exceeding 90%). For instance, one study reported the Naive Bayes model achieving 95.86% accuracy on phishing URL classification, while the Decision Tree model reached 97.02% accuracy and the Random Forest model reached the highest accuracy (97.98%) [2]. However, such results often come with limitations. They require extensive feature engineering or the use of metadata and may not generalize well to evolving phishing tactics. High performance on static benchmarks can mask weaknesses against novel or obfuscated attacks. Moreover, some traditional models can be brittle, with noise or feature variations negatively impacting performance, and their decision criteria can be opaque.

In recent years, large language models (LLMs) have emerged as powerful tools for a variety of Natural Language Processing (NLP) tasks. LLMs excel in generating coherent and contextually relevant text, understanding complex verbal patterns, and performing tasks such as content summarization, sentiment analysis, and conversational AI [3–6]. LLMs have been applied to specialized fields such as medical NLP, where they assist in tasks like medical information extraction, diagnosis, and personalized treatment plans [4]. They also show promise in the telecommunications domain [7], oil and gas industry [8], and even in the Quranic studies [9]. Recent studies have compared human and LLM-generated annotations across multilingual NLP tasks, revealing that LLMs can produce competitive annotations in simpler tasks like sentiment analysis [10]. LLMs pre-trained on vast text corpora (e.g., GPT-3/4, PaLM, and LLaMA) possess an inherent ability to understand linguistic patterns and context, which can be harnessed for security tasks like phishing detection. By formulating URL detection as a language understanding problem, we can prompt an LLM to assess whether a given URL is likely to be phishing or legitimate. Early explorations in this vein have shown promise. A study investigated using LLMs directly via prompt engineering for phishing URL classification [11]. In their case study, prompting closed-source chat models (OpenAI's GPT-3.5-turbo and Anthropic's Claude 2) yielded up to a 92.74% F1-score on a benchmark phishing dataset, demonstrating that LLMs can accurately discern malicious URLs from benign ones even without task-specific training. Notably, they found Claude 2 consistently outperformed GPT-3.5 on this task (Claude reached 92.74% F1 vs. GPT-3.5's 88.54%, under the best prompt), though both models benefited from carefully engineered prompts. These results are remarkable, considering no explicit feature engineering was used—the LLMs learned to detect phishing cues from the URL strings and their own knowledge.

Despite these advancements, there are critical gaps in our understanding. Prior studies mostly used proprietary "black-box" LLMs, accessible only via APIs. Relying on closed models poses practical limitations: they may apply content filters that block certain inputs (indeed, GPT-3.5 sometimes refused to classify some URLs due to internal filters), their internal reasoning is not transparent, and they incur ongoing API costs and latency that make deployment challenging. For instance, deploying GPT-4o for phishing detection at scale could cost approximately $0.60 per 1000 URLs (assuming 100 input tokens and 5 output tokens per request), whereas self-hosted open-source models incur only electricity and hardware amortization. Moreover, the rapid development of open-source LLMs offers an opportunity to build self-hosted, transparent phishing detectors, but these open models have not been systematically benchmarked for this task. Questions remain on whether open-source LLMs (which users can fine-tune or run on-premise) can approach the accuracy of GPT-4/Claude, and how their inference efficiency compares. Likewise, while prompt engineering has proven effective, we lack comparative evaluations of different prompting strategies (e.g., zero-shot vs. few-shot) on a wide range of models. The prompt formulation can significantly influence LLM performance, and an inappropriate prompt might even

reduce model accuracy in some cases. This challenge echoes similar findings in computer vision, where few-shot classification has shown that performance can be greatly enhanced by better feature representation and metric learning techniques, even under data-scarce conditions [12].

Our work aims to address these gaps by providing a comprehensive benchmarking of 21 open-source LLMs for phishing URL detection using various prompt engineering techniques. We evaluate models spanning 1.5 billion to 70 billion parameters from diverse model families, using four distinct prompting strategies: (1) zero-shot prompting, (2) role-playing prompting, (3) chain-of-thought (CoT) prompting, and (4) few-shot prompting. We measure each model's detection performance (accuracy, precision, recall, and F1) on a standard balanced phishing URL dataset and also record inference execution times to assess efficiency. Through this study, we seek to identify which open models (and prompt approaches) are most effective for phishing detection and to elucidate the trade-offs between detection accuracy and computational cost. Ultimately, our findings inform how one might design a phishing detection system powered by LLMs—weighing the choice of model and prompt to achieve high accuracy while remaining practical for real-world deployment.

This study provides a novel comprehensive benchmarking of 21 state-of-the-art open-source LLMs specifically for phishing URL detection, contrasting sharply with prior research that primarily utilized classical machine learning techniques or focused solely on proprietary closed-source models accessed via APIs. Unlike previous studies, our work systematically evaluates a diverse set of open-source models, illuminating the effectiveness of various prompt engineering techniques—particularly the practical advantages of few-shot prompting—and demonstrating that open-source LLMs can achieve comparable or superior performance to proprietary models. Furthermore, we provide new insights into the strengths and weaknesses of different prompting strategies, highlighting practical considerations for cybersecurity applications.

In the following, we discuss relevant prior work and the motivation for our benchmarking (Section 2), describe the experimental setup, including models, data, and prompt techniques (Section 3), present and analyze the results in terms of both performance and efficiency (Section 4), and finally discuss implications, limitations, and future directions (Section 5).

## 2. Related Works

Prior research in phishing URL detection traditionally relied heavily on supervised machine learning models using handcrafted features extracted from URLs or associated metadata. These approaches achieved high accuracy but often required extensive manual feature engineering and struggled to generalize to evolving phishing tactics [2,13].

More recently, large pre-trained neural language models (LLMs) such as GPT-3.5 and Claude have demonstrated impressive capabilities for various natural language tasks, including cybersecurity-related classification problems [14,15]. Initial studies applying these proprietary models to phishing detection have shown promising results without explicit training, highlighting the potential of prompt engineering [16]. However, such models present challenges, including API access constraints, lack of transparency, and cost.

Recent advancements in open-source LLMs, such as LLaMA, Gemma, Qwen, Phi, DeepSeek, and Mistral, have opened possibilities for transparent, self-hosted phishing detection systems. Nonetheless, these open models have not been extensively benchmarked for phishing detection. Our work directly addresses this gap by providing the first extensive benchmarking of open-source LLMs using multiple prompt engineering methods, aiming to clarify their practical applicability and performance relative to proprietary counterparts.

### *2.1. Phishing URL Detection with Traditional Machine Learning*

Phishing detection has long been studied as a classification problem using various machine learning techniques. Early approaches focused on extracting features from the URL string or associated webpage (HTML content, WHOIS data, etc.) and training classifiers to distinguish phishing links from legitimate ones [11]. Common lexical features include the URL length, presence of suspicious substrings (like "login" or numerals in domains), entropy of the URL string, use of HTTPS, and so on. Classifiers such as SVM, Logistic Regression, Decision Trees, Random Forests, and Multi-Layer Perceptrons have all been applied, often yielding high performance on benchmark datasets. For example, Sahingoz et al. and others reported that Naive Bayes achieves 95.86% accuracy on phishing URL classification, while a Decision Tree reaches 97.02% accuracy and a Random Forest reaches the highest accuracy (97.98%) in controlled experiments [2]. However, these traditional methods have notable limitations. They typically require manual feature engineering or feature selection to be effective. As phishing techniques evolve (e.g., using homograph attacks, adding benign-looking tokens to URLs to fool filters), fixed feature sets may miss new patterns. Models like Random Forest also offer limited explainability in terms of which features drive decisions. More importantly, the impressive accuracy numbers reported in the literature can be misleading—models may overfit to the specific dataset or distribution of phishing samples. A study highlighted the need for diverse, representative datasets for phishing detection. Without continuous updating, an ML model trained on past phishing data might fail against novel URLs crafted to evade those very features [17]. Thus, maintaining and retraining such models becomes an ongoing effort as attackers adapt.

### *2.2. Deep Learning and Large Language Models in Security*

In search of more robust solutions, researchers have turned to deep learning and more recently, to large pre-trained models. Neural networks can automatically learn feature representations given enough data. There have been attempts at using CNNs or RNNs on URL strings or email text for phishing detection, which removes the need for manual features [18]. Nonetheless, purely supervised training of deep networks still requires sizable labeled datasets, and performance gains have been incremental. The advent of LLMs pre-trained on billions of tokens introduces a powerful new paradigm: using the knowledge encoded in an LLM for security tasks. LLMs are not explicitly trained on phishing data, but they ingest vast amounts of Internet text, which likely includes many URL examples and patterns of malicious versus safe links. An LLM might "know" certain common phishing domains or be able to infer from context (e.g., a URL with a misleading domain name) that it is suspicious.

Another approach is to use the LLM directly to make the classification decision via prompting. This treats the LLM as a standalone phishing detector: we feed it the URL (possibly with some instruction or context) and it outputs a label or explanation. A case study was conducted comparing this prompt-based approach against fine-tuning smaller models for phishing URL detection [11]. They experimented with prompt engineering on two state-of-the-art closed LLMs (GPT-3.5-turbo and Claude 2). They designed a few prompt templates to coax the models into performing a binary classification of URLs: a straightforward zero-shot prompt asking, "Is this URL phishing or legitimate?", a role-playing prompt instructing the model to act as a cybersecurity expert, and a chain-of-thought prompt asking the model to reason, step by step, before giving an answer. Their findings showed that prompt design significantly affected performance. GPT-3.5, in a naive zero-shot setting, achieved around 77.9% F1 on their test set, but with an improved prompt, this rose to 88.5% F1. Claude 2 performed strongly even with zero-shot (90.7% F1) and reached 92.7% F1 with the chain-of-thought prompt. The chain-of-thought strategy,

which had the model provide a rationale before the verdict, gave the highest boost for GPT-3.5 (indicative that reasoning helped it catch more phishing cues). An interesting observation was that across all prompt types and both models, precision was higher than recall – meaning the LLMs were conservative, more likely to correctly identify legitimate URLs (low false positive rate) but at the expense of missing some phishing URLs (false negatives). In the security context, missing phishing (false negative) is arguably more dangerous; thus, methods to increase recall (catch more phishing) are valuable.

While closed models responded well to prompt engineering, they are "black-boxes" with hidden drawbacks for operational use. The authors noted the opaque nature of GPT-3.5/Claude made it unclear why Claude outperformed GPT-3.5, and they experienced minor issues like GPT-3.5 occasionally refusing to process some inputs due to content filters (though only for 1–2 URLs). Moreover, reliance on third-party APIs raises concerns of cost, rate limits, data privacy, and lack of offline capability. This motivates the exploration of open-source LLMs that can be run locally. The open-source AI community has produced numerous LLMs (e.g., Meta's LLaMA 2, MosaicML's MPT, EleutherAI's GPT-J/Neo, etc.) with competitive performance. By fine-tuning such models on domain-specific data, one can create specialized detectors. Trad and Chehab pursued this by fine-tuning several small open models (GPT-2 variants, Baby-Llama, and Bloom) on the phishing data. The fine-tuned models achieved outstanding results—up to 97.29% F1 and 99.56% Area Under the Curve (AUC) on the test set, exceeding the prior state-of-the-art for that dataset. This confirms that, given training data, even relatively small open models can excel at phishing detection. However, fine-tuning requires training infrastructure and expertise, and the resulting model is task-specific. In contrast, prompt use of a general LLM is "zero-cost" in terms of training, albeit potentially lower in peak accuracy [11].

Other related works further confirm the promise of LLMs in phishing detection [14,19,20]. Ref. [14] developed ChatSpamDetector, utilizing GPT-4 to detect phishing emails, achieving a remarkable accuracy of 99.70% and providing detailed reasoning to assist users. Ref. [19] demonstrated that LLMs such as GPT-4, Claude, PaLM, and LLaMA effectively detected phishing emails, sometimes surpassing human-level detection. Additionally, Ref. [20] proposed a multimodal LLM-based system that efficiently identifies phishing webpages, showcasing high accuracy and interpretability, as well as robustness against adversarial attacks.

In summary, prior work indicates the following: (i) classical ML can detect phishing well under controlled conditions but has maintenance and generalization issues, (ii) LLMs, even without fine-tuning, have an innate capability to identify phishing URLs when prompted effectively, and (iii) open-source models offer a path to deployment without the downsides of closed APIs, but their performance has not been benchmarked in depth for this task. Our work builds on these insights by evaluating a broad array of open LLMs with prompt-based phishing detection. We aim to see if we can attain high accuracy comparable to closed models like Claude 2 (90+ F1) using open models, and to understand the impact of prompt strategies across different model architectures. Additionally, we explicitly measure inference speed, which is a critical factor for real-world use (scanning millions of URLs daily). While chain-of-thought prompting improved accuracy in prior work [11], it also entails the model generating more text (reasoning), potentially slowing down inference dramatically. Indeed, recent research has found that for certain tasks, prompting an LLM to "think step-by-step" can reduce performance and efficiency when the task is straightforward [21]. Thus, we hypothesize there is a trade-off: complex prompts might boost accuracy for some models, but simpler prompts or few-shot examples might achieve similar results with less overhead.

## 3. Experimental Setup

To investigate these questions, we designed a comprehensive evaluation involving 21 open-source LLMs and 4 prompting techniques on a standard phishing URL dataset. In this section, we describe the models tested, the dataset and preprocessing, the prompt formulations, and the evaluation metrics and procedure.

### *3.1. Open-Source LLMs Evaluated*

We benchmarked 21 open-source LLMs, carefully selected to represent a diverse range of model families, parameter sizes (from 1.5B to 70B), design objectives, and development backgrounds. The goal was to capture a representative cross-section of the current open-source LLM landscape while enabling meaningful comparisons across model scales and prompting strategies. The evaluated models include the following:

- Meta LLaMA Family: Llama3 (8B and 70B), Llama3.1 (8B and 70B), Llama3.2 (3B), and Llama3.3 (70B)—hypothetical successive improvements of Meta's LLaMA model series. Parameter counts are indicated by numerical suffixes (e.g., 8B denotes 8 billion parameters).
- Google Gemma 2: Gemma2 variants (2B, 9B, and 27B)—models introduced by Google, designed for efficient multilingual task performance, with the 27B model demonstrating particularly strong capabilities.
- Alibaba Qwen Family: Qwen-7B, Qwen2-7B, and Qwen2.5-7B—iterations of Alibaba's Qwen model, progressively enhanced for broader general-purpose tasks. Each numerical suffix indicates the parameter size, and successive versions incorporate iterative improvements in architecture and training methodologies.
- Microsoft Phi Family: Phi-3 14B and Phi-4 14B—two models from Microsoft's Phi series, notable for their relatively smaller size yet exceptional reasoning capabilities. Phi-4, in particular, has been reported to achieve competitive performance relative to significantly larger models due to advanced training methods and synthetic data integration.
- Mistral Family: Mistral-small 24B—based on the Mistral architecture, which garnered attention for its efficient yet strong performance. This 24B variant is evaluated as a representative of mid-scale models.
- DeepSeek Series: DeepSeek R1 variants (1.5B, 7B, 8B, 14B, 32B, and 70B)—a series of open-source models designed for general-purpose tasks, emphasizing scalability and open-source model advancements. The range from very small (1.5B) to very large (70B) allows for the examination of scaling effects within the same family.

### *3.2. Hardware and Environment*

Inference was conducted on a system comprising four NVIDIA RTX A6000 GPUs, each offering 49 GB of memory, running with CUDA version 12.6 and NVIDIA Driver version 560.35.05. For most inference tasks, a single GPU was used actively, while the remaining GPUs remained idle. The level of GPU utilization varied depending on the model's parameter size and the applied quantization strategy. On average, each model instance utilized around 2–3 GB of GPU memory. Power usage ranged from 15 W during idle periods to approximately 278 W under full computational load, with GPU temperatures reaching up to 76 °C during intensive inference sessions.

All inference tasks were executed using Python-based scripts, with Ollama employed to efficiently serve quantized models on GPU. For models that either exceeded available GPU memory or lacked compatibility with CUDA-based libraries, execution was automatically redirected to CPU, which considerably increased inference latency. This setup reflects

realistic deployment conditions for local open-source LLMs, emphasizing the importance of memory efficiency, power consumption, and hardware resource management.

### *3.3. Dataset and Preprocessing*

We use the phishing URL dataset by Hannousse and Yahiouche [22] available for download at Mendeley Data (https://data.mendeley.com/datasets/c2gw7fy2j4/3 (accessed on 31 March 2025)) as the evaluation benchmark. This dataset contains a total of 11,430 URL samples, with an equal split of phishing and legitimate URLs (approximately 5715 each). The dataset is notable for being carefully curated and balanced, addressing some of the biases in earlier collections. Each URL in the dataset originally comes with a set of 87 engineered features (related to URL composition, domain info, etc.) and a label indicating phishing or legitimate. However, in line with prior LLM-based studies, we ignored the provided features and used only the raw URL text and its label. This allowed us to evaluate the LLMs' ability to classify based solely on the URL string, without additional structured inputs.

While the dataset offers a strong foundation for benchmarking, we acknowledge that it does not fully represent the complexity of real-world phishing threats. Modern phishing URLs often incorporate obfuscation techniques, dynamic tracker parameters, multilingual domain patterns, and evolving tactics that may not be fully captured in this static, curated dataset.

For our experiments, we focus on the same subset used by Trad and Chehab for testing their prompts. Specifically, we take a test set of 1000 URLs, comprising 500 phishing and 500 legitimate examples (this subset was originally sampled randomly from the full dataset, maintaining the 50/50 balance). We did not perform any further fine-tuning or training of the LLMs on the dataset; instead, the models are used in an out-of-the-box manner with prompts. The 1000 URLs were presented one by one to each model with the respective prompt template, and the model's output was interpreted as a phishing or legitimate prediction.

We focused exclusively on the raw URL string, intentionally excluding any contextual metadata (such as webpage content or email headers), to simulate a content-free detection setting that mirrors early-stage threat filtering. All URLs were provided to the models as plain text (e.g., "http://www.x.com/login/"). We performed minimal preprocessing, ensuring each URL was a standalone input (without additional context unless added by the prompt template), and escaping any special characters in the prompt as needed. No browsing or content retrieval was performed—the models see only the URL string, not the actual webpage content, focusing the task strictly on URL-based phishing detection. This setup mimics a common real-world scenario where a security filter must judge a URL directly (e.g., in an email) without fetching the page.

### *3.4. Prompt Engineering Techniques*

To perform phishing URL classification, we employed four distinct prompt engineering techniques: zero-shot, role-playing, chain-of-thought, and few-shot prompting. Each model was explicitly prompted to output a concise decision ("Phishing" or "Legitimate"). For CoT prompts, we structured instructions explicitly to produce reasoning followed by a clear, single-word verdict. The goal was to see how different prompt styles might elicit better or worse performance from the model. The prompting strategies are as follows:

- Zero-shot prompting: This is the simplest approach where the model is directly asked to classify the URL with no examples or role context. We formulated a straightforward instruction, and the model is expected to output a label or a brief answer (e.g., "Phishing" or "Legitimate"). This method leverages the model's learned knowl-

edge directly, with no additional guidance beyond the question. Listing 1 shows a zero-shot prompting template.

**Listing 1.** Zero-shot prompting template

```
Classify the following URL as either phishing or legitimate.
Respond with only one word: Phishing or Legitimate.
Provide no explanation, no additional text, and no formatting.

URL: "{url}"

```

- Role-playing prompting: In this approach, we asked the model to adopt a specific role or persona relevant to the task. We prompted the model as if it were a cybersecurity analyst or phishing detection expert. Listing 2 shows the role-playing prompting template. By role-playing, we hope the model will internalize the instruction and provide a more informed and context-aware response, potentially improving accuracy. The model might respond with something like: "As a cybersecurity expert, I notice the URL has an IP address instead of a domain, which is suspicious... Therefore, this looks like a phishing URL".

**Listing 2.** Role-playing prompting template

```
Act as a seasoned cybersecurity expert who has been hired by a company to
 examine and analyze URLs for potential phishing threats.
Classify the following URL as either phishing or legitimate.
Respond with only one word: Phishing or Legitimate.
Provide no explanation, no additional text, and no formatting.

URL: "{url}"
```

- Chain-of-thought prompting: Chain-of-thought prompting encourages the model to think step by step and articulate its reasoning before arriving at a final answer. Listing 3 shows the CoT prompting template. Our CoT prompt asked the model to first enumerate a reasoning process, "Let's reason this out. Is the URL <URL> phishing? Think step by step", and then after the reasoning, provide a final verdict (phishing or legitimate). The expectation, based on prior work, is that some models might perform better when they explicitly reason about features of the URL (e.g., "The domain looks like paypal.com but has an extra token, which is a known phishing trick, so..."). We included instructions for the model to output the final answer clearly (such as prefixing it with "Final answer:"). This technique has yielded gains for powerful models on various tasks that benefit from intermediate reasoning [23]. However, it also introduces the risk of the model "overthinking" a simple decision, and it certainly increases the amount of text the model must generate for each query, which can slow down inference.

**Listing 3.** Chain-of-thought prompting template

```
You are a cybersecurity analyst specializing in phishing detection.

Your task is to analyze the following URL and determine whether it is a
phishing link or a legitimate website.

Let us reason this out carefully, step by step. Consider indicators such
as:
- Suspicious or misspelled domains
- Use of IP addresses instead of domain names
```

```
    - Extra tokens or misleading subdomains
    - Use of HTTP instead of HTTPS
    - Known brand impersonation patterns
    - Unexpected or uncommon URL structures

    URL: "{url}"

    Think step by step:

    [Model will now generate reasoning]

    After your reasoning, provide your final classification in this format:

    Final answer: [Phishing or Legitimate]
```

- Few-shot prompting: Few-shot prompting provides the model with a couple of example cases (input–output pairs) before asking it to handle the new input [23]. We constructed a prompt that included a small number of example URLs along with their correct classification as shown in Listing 4. The few-shot examples were chosen to be illustrative of various phishing tactics (like deceptive subdomains, HTTP vs. HTTPS usage, and IP address links) and legitimate URLs. The expectation is that by seeing these examples, an LLM can induce the pattern of the task (much like how GPT-3 demonstrated few-shot learning abilities [23]), potentially improving its accuracy on the test URL. Few-shot prompting essentially primes the model with a mini training set in the prompt itself.

**Listing 4.** Few-shot prompting template

```
    Classify the following URL as either phishing or legitimate.
    Respond with only one word: Phishing or Legitimate.
    Provide no explanation, no additional text, and no formatting.

    Here are some examples:

    URL: "https://support-appleld.com.secureupdate.duilawyeryork.com/ap/89
    e6a3b4b063b8d/?cmd=_update&dispatch=89e6a3b4b063b8d1b&locale=_"
    Response: phishing
    URL: "http://www.budgetbots.com/server.php/Server%20update/index.php?
    email=USER@DOMAIN.com"
    Response: phishing
    URL: "https://www.facebook.com/Interactive-Television-Pvt-Ltd-Group-M
    -100230523435650/photos/?ref=page_internal"
    Response: legitimate
    URL: "http://www.mypublicdomainpictures.com/"
    Response: legitimate

    URL: "{url}"
```

All prompts were designed to elicit a concise answer (ideally just a label). During inference, we parsed the model's output to determine the classification. Most models would output a sentence or phrase; we checked for keywords like "phishing", "legitimate", "safe", etc., to map to the binary label. We also had to ensure consistency—for some models, we explicitly instructed the format of the answer (e.g., "Answer with just 'Phishing' or 'Legitimate'."). In cases where the model's answer was unclear or verbose, we applied simple rules to interpret it (for example, if the output said "This appears to be a phishing link because...", we count that as phishing).

### *3.5. Evaluation Metrics and Procedure*

We evaluate each model's performance on the 1000-url test set using standard classification metrics: accuracy, precision, recall, and F1-score. These are defined in the usual way, treating "phishing" as the positive class (true positives = correctly identified phishing URLs, false positives = legitimate URLs incorrectly flagged, etc.). In context, the following applied:

- Accuracy is the overall percentage of URLs correctly classified (phishing identified as phishing and legitimate as legitimate).
- Precision (Positive Predictive Value) = TP/(TP + FP)—the fraction of URLs the model flagged as "phishing" that were actually phishing. High precision means few false alarms (false positives).
- Recall (True Positive Rate) = TP/(TP + FN)—the fraction of actual phishing URLs that the model successfully caught. High recall means few phishing attempts slip past undetected (low false negatives).
- F1-score is the harmonic mean of precision and recall: $2 * \text{Precision} * \text{Recall}/(\text{Precision} + \text{Recall})$. F1 is a balanced measure that is useful when one wants to account for both error types and the classes are balanced (as in our dataset).

Given our equal class distribution, accuracy can be a bit less informative (since 50% is the baseline random accuracy), so we focus on precision, recall, and especially F1 when comparing methods. A high F1 indicates the model is doing well on both precision and recall fronts.

For each combination of model and prompting method (21 models × 4 prompts = 84 runs), we collected the model's predictions for all 1000 URLs and computed these metrics. We also recorded the inference time: essentially, how long (in seconds) it took from the start of processing the first URL to finishing the last URL for that model+prompt. The timing includes the model loading and prompt processing overhead for each URL (but since the model is loaded once and kept in memory, load time is negligible relative to the per-sample inference cost, especially for large models). We ensured that each run was performed in isolation on the machine to avoid resource contention. If a model failed to produce an output for a particular prompt (which happened rarely, e.g., one model crashed on the chain-of-thought prompt due to length), we handled that case (e.g., counting it as a miss or re-running with a shorter output limit).

The execution times are useful to gauge the efficiency of each model and prompt. A model that is highly accurate but takes an impractically long time to run would be hard to deploy at scale. We did note large differences in runtime especially for the chain-of-thought prompt, which we will detail in the results.

## 4. Results

In this section, we present the benchmarking results of the 21 LLMs, focusing first on phishing detection performance (accuracy, F1, etc.) under each prompt strategy, and then on execution time and efficiency. We include tables and figures to summarize these findings.

### *4.1. Classification Performance of Different Models and Prompts*

Table 1 summarizes the F1-scores of all 21 models under each prompting technique (zero-shot, role-playing, chain-of-thought, and few-shot). For example, the top block of Table 1 shows that with few-shot prompting, the best model (Llama3.3_70b) achieved 91.24% F1, whereas with chain-of-thought prompting the performances dropped significantly for most models. We use F1-score as a succinct indicator of overall classification effectiveness. (Accuracy, precision, and recall for each case were also computed; we will mention notable patterns in those as needed.) The models are ordered roughly by their size (parameter count).

**Table 1.** F1-score (%) of 21 open-source LLMs on the phishing URL test (500 phishing, 500 legitimate) under four prompting strategies. The best result for each model is highlighted in bold.

| Model | Zero-shot F1 (%) | Role-playing F1 (%) | Chain-of-Thought F1 (%) | Few-shot F1 (%) |
|---|---|---|---|---|
| deepseek-r1_1.5b | 69.94 | 67.13 | 67.59 | **71.74** |
| gemma2_2b | 84.19 | 84.90 | 75.37 | **85.42** |
| llama3.2_3b | 72.96 | 73.53 | 55.58 | **76.34** |
| qwen2_7b | 86.40 | 85.35 | 50.62 | **86.82** |
| deepseek-r1_7b | 73.14 | **74.23** | 56.98 | 71.33 |
| qwen2.5_7b | 81.98 | 83.26 | 74.44 | **87.22** |
| qwen_7b | 81.98 | **82.00** | 57.91 | 75.67 |
| deepseek-r1_8b | 73.04 | 73.29 | 67.35 | **76.81** |
| llama3_8b | 85.28 | 84.57 | 67.12 | **88.03** |
| llama3.1_8b | 87.45 | **87.74** | 64.40 | 87.50 |
| gemma2_9b | **87.67** | 86.48 | 77.67 | 86.86 |
| phi3_14b | **79.24** | 71.48 | 63.63 | 67.78 |
| phi4_14b | 73.89 | 62.32 | 52.73 | **77.22** |
| deepseek-r1_14b | 82.14 | **83.18** | 45.07 | 81.77 |
| mistral-small_24b | **88.32** | 82.27 | 7.03 | 88.13 |
| gemma2_27b | 87.32 | 88.24 | 80.34 | **90.31** |
| deepseek-r1_32b | 80.12 | 76.41 | 42.96 | **80.86** |
| deepseek-r1_70b | **86.15** | 85.98 | 51.62 | 85.34 |
| llama3.3_70b | 88.81 | 90.17 | 47.84 | **91.24** |
| llama3.1_70b | 88.15 | 89.46 | 61.98 | **89.71** |
| llama3_70b | **89.48** | 87.93 | 69.85 | 88.29 |

Several important observations can be made from Table 1:

- Few-shot prompting emerged as the best overall strategy. For the vast majority of models, the highest F1-score is achieved with the few-shot prompt (as indicated in bold for a few notable models). The average F1 across all models in the few-shot setting was 82.6%, slightly higher than zero-shot ( 82.3%) and substantially higher than role-playing ( 80.9%). Every model of moderate to large size (9B and above) reached its peak performance with the few-shot prompt, often gaining 1–3 percentage points in F1 over zero-shot. For example, the 70B models ("llama3.3_70b" and others) all exceeded 89–90% F1 with few-shot, whereas they were in the high 80s with zero-shot. Llama3.3_70b (70B) in particular achieved the top F1 of 91.24% with few-shot prompting, which is an excellent result on this task—approaching the performance of Claude 2 (92.7% F1) reported by prior work [11]. The few-shot examples likely provided helpful context, effectively teaching the model the concept of phishing detection on-the-fly [23].
- Zero-shot prompting is surprisingly strong and often nearly as good as few-shot for many models. Several models (e.g., llama3_70b with 89.48% F1 zero-shot vs. 88.29% F1 few-shot) actually performed slightly better in zero-shot than few-shot, though by a negligible margin. The differences between zero-shot and role-playing for large models are also small in many cases. This indicates that the largest models might already have sufficient knowledge to perform well without needing examples or elaborate role context. For instance, Llama3_70b led in zero-shot with 89.48% F1. On the other hand, smaller models (below 10B) generally benefited more from few-shot—e.g., llama3.2_3b improved from 72.96% to 76.34% F1 with few-shot, a noticeable jump for a 3B model.
- Role-playing prompts had mixed results. On average, the role persona prompt (acting as a security expert) yielded slightly lower F1 than zero-shot. Some models did improve with role-playing—e.g., llama3.1_8b went from 87.45% (zero-shot) to 87.74% (role), and llama3.3_70b from 88.81% to 90.17%. Claude 2 in the closed-model study similarly saw a benefit from the role-playing prompt [11]. In our case, the largest models seem to respond well to role hints, but a few models were hurt by it. One extreme case is mistral-small_24b, which dropped from 88.32% F1 zero-shot to 82.27%

with the role prompt. It appears that the role-playing instruction may have confused some models or caused them to output lengthier explanations (which might have made extracting the final answer harder). Overall, the role prompt was not consistently beneficial across the board, though for some top models it was on par with zero-shot performance.

- Chain-of-thought prompting was largely detrimental for this task and these models. This is a striking result: nearly every model saw a large drop in F1 when asked to produce a reasoning chain. The average F1 plummeted to 59% with CoT, compared to 82% with other methods. Some models completely failed in this mode. The most dramatic was again mistral-small_24b, which fell to an abysmal 7.03% F1—essentially failing to correctly classify almost any phishing instances with CoT. We suspect that certain models might not have properly followed the prompt to give a final answer after reasoning, or their reasoning text confounded our simple output parsing. On the flip side, a few models handled CoT relatively well: the Gemma2 series (2b, 9b, and 27b) models stand out, achieving 75–80% F1 with CoT. In fact, gemma2_27b maintained a strong 80.34% F1 with CoT, only modestly lower than its 87–90% with other prompts. This suggests that some models (perhaps those fine-tuned to follow instructions or reason, like the Gemma2 family) can utilize chain-of-thought without completely losing accuracy. However, the largest LLaMA-based models (70B) all saw drops: e.g., llama3_70b went from 89 F1 zero-shot to 69.85% F1 with CoT. Interestingly, in those cases, the recall often remained high but precision dropped significantly, meaning the model would label almost everything as "phishing" when reasoning step-by-step (catching all true phishing but also flagging many legitimate as phishing, hence precision tanked). This aligns with the idea that forcing a model to explain can sometimes lead it to be overly cautious or to apply some memorized rule too broadly.

Figure 1 compares the average F1-score across the four evaluated prompt strategies: zero-shot, role-playing, chain-of-thought, and few-shot. Few-shot prompting clearly outperforms other strategies, suggesting its superior capability in guiding LLMs for phishing detection tasks. Figure 2 provides a scatter plot illustrating the relationship between precision and recall for each model and prompting strategy (assuming Precision $\approx$ Recall $\approx$ F1-score for simplicity), highlighting performance distribution among prompt methods. Higher-performing prompt methods cluster near the upper-right corner, demonstrating balanced detection capabilities. Figure 3 shows a log–log scatter plot of inference time versus model size, illustrating how larger models generally incur higher inference times. As illustrated in Figure 4, the heatmap provides a visual overview of performance across all models and prompt types, quickly highlighting which combinations achieve optimal performance, where warmer colors indicate higher F1-scores. One can quickly spot that the column corresponding to few-shot prompting has generally warmer colors (higher performance) across most models, confirming the superiority of few-shot prompting.

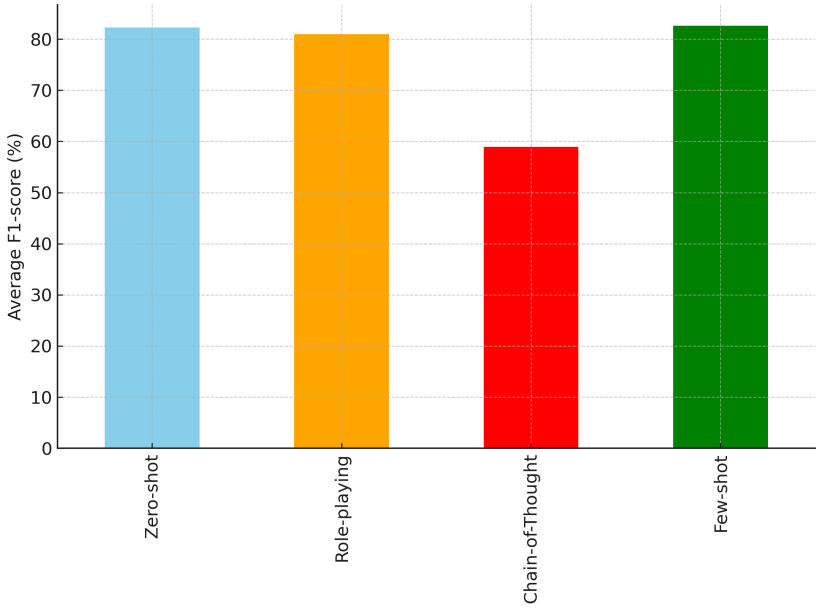


**Figure 1.** Average F1-score achieved by all models under each prompting method. This bar chart summarizes the mean performance of each prompt strategy—zero-shot, role-playing, chain-of-thought, and few-shot—across all evaluated models. Higher bars indicate the stronger overall effectiveness of the prompt method for phishing URL classification.

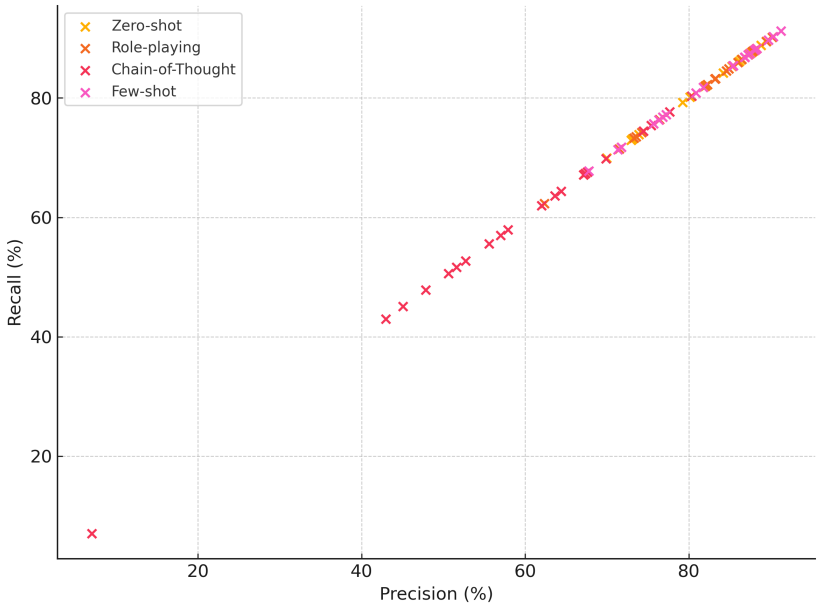


**Figure 2.** Precision–recall scatter plot for each model under different prompting strategies. Each point represents a model's performance (precision vs. recall) for a given prompt method; points clustered toward the upper-right indicate high balanced accuracy (F1-score).

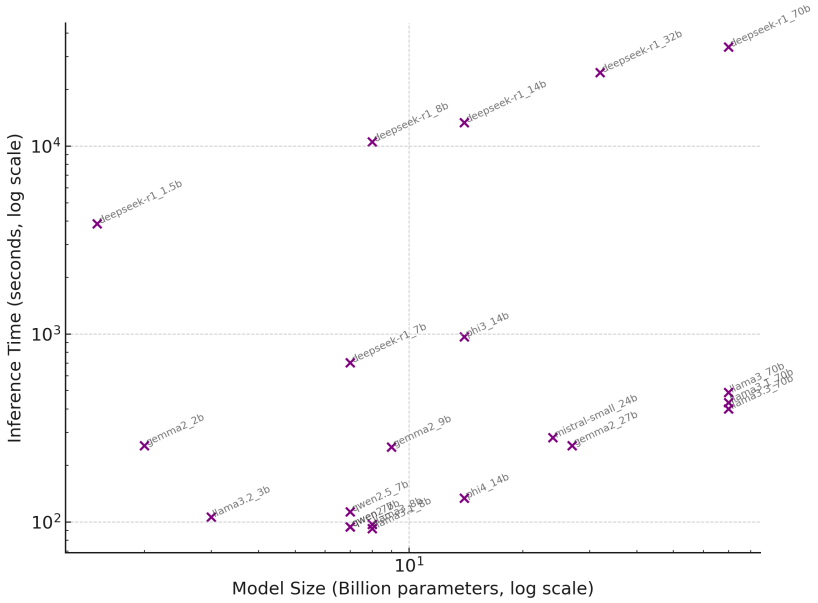


**Figure 3.** Inference time vs. model size plotted on a log–log scale. Each point represents a model, where the x-axis denotes the number of model parameters (in billions) and the y-axis shows the average inference time per URL (in seconds). This visualization highlights the computational trade-offs between model size and inference speed, with annotations for selected models to aid interpretation.

The poor performance of CoT prompting here is consistent with recent findings that chain-of-thought can hurt performance on tasks that do not fundamentally require multi-step reasoning [21]. Phishing URL detection might be such a task: it may rely more on pattern recognition (e.g., suspicious substrings or domain patterns) than on logic or arithmetic that truly benefits from step wise reasoning. Forcing a model to "explain itself" might introduce noise. This is a valuable insight: while CoT prompting has been a popular technique for boosting reasoning in math or logical tasks [21,23], it may be counterproductive for straightforward classification tasks like ours. In terms of statistical significance, the differences between CoT and the other prompts are large enough (20–30 point swings in F1) that we can be confident that CoT is suboptimal for most models here.

Beyond F1-scores, we also examined precision and recall trends. Generally, we observed (similar to Trad and Chehab's findings) that in the zero-shot, role, and few-shot settings, models tended to have precision a bit higher than recall [11]. That is, they produced relatively few false positives (mislabeling legitimate URLs as phishing), but sometimes missed some phishing URLs (false negatives). For example, llama3_70b in zero-shot had precision $\approx$ 85.7% and recall $\approx$ 93.6%, meaning it caught 93.6% of all phishing but also mistakenly marked some legitimate URLs (precision 85.7%). A security practitioner might prefer higher recall (catch all phish) even if precision drops, but raising recall often means the model starts flagging more legitimate URLs too. In our few-shot prompts, some models achieved a better balance. The few-shot examples likely helped models learn to be a bit more inclusive in identifying varied phishing tricks, boosting recall. For instance, gemma2_27b few-shot had precision 85.3%, recall 96.0%, an excellent balance (F1 90.31%). In contrast, under CoT prompting, many models flipped to a high-recall, low-precision regime or vice versa unpredictably. CoT often caused either very cautious behavior (e.g., one model marked almost everything as safe except very obvious phishing, leading to low recall but high precision, or the opposite).

Notably, the best model (Llama3.3_70b with few-shot) achieved about 91% F1 and 90.8% accuracy, with precision at 87.1% and recall at 95.8%. This indicates it only missed about 4.2% of phishing URLs and misclassified 12.9% of the legitimate ones. This perfor-

mance is close to the closed-source Claude 2's 92–93% F1 [11]. It is also far above what GPT-3.5 achieved in zero-shot (78% F1) [11], demonstrating that an open model (albeit a large one) with proper prompting can outperform a powerful API model that is not optimally prompted. Of course, Claude 2 and GPT-4 are capable of even higher performance if fine-tuned or given more sophisticated prompting; our point here is that open models are catching up in their ability to handle such tasks.

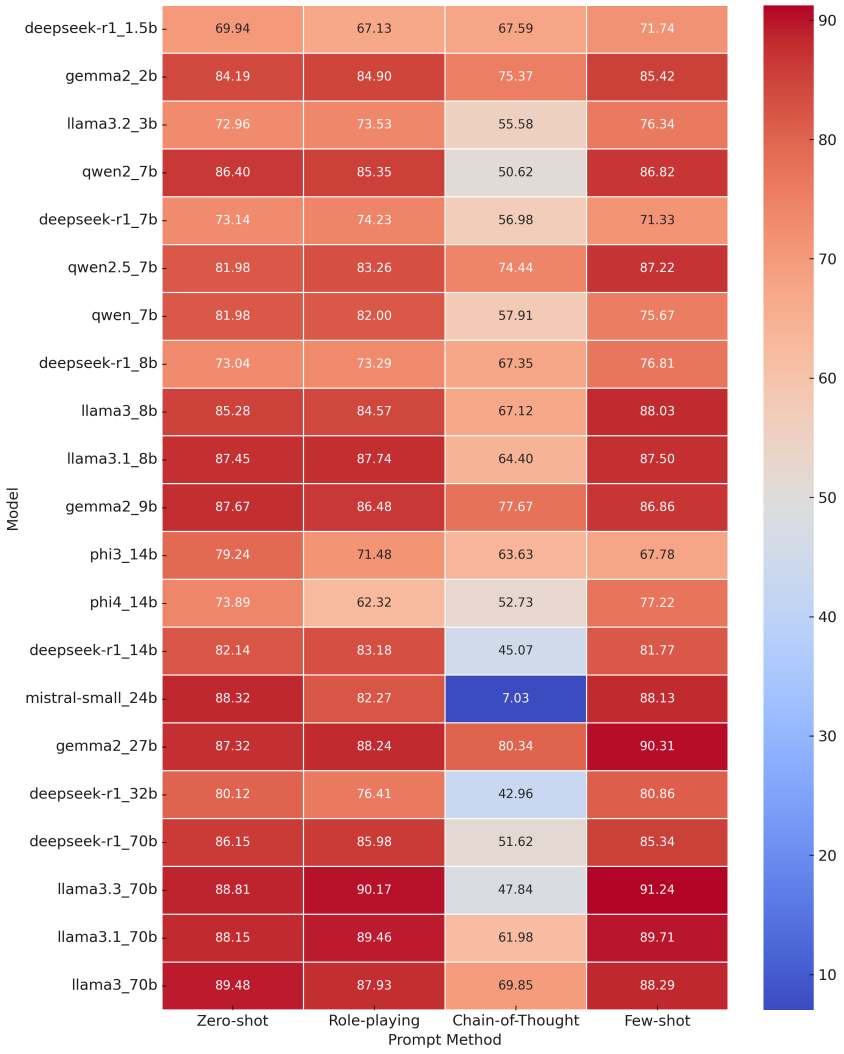


**Figure 4.** Heatmap of F1-scores for each model under different prompting strategies. Rows represent individual LLMs, and columns correspond to prompt methods. Cell color intensity indicates the F1-score (darker cells represent higher values). This matrix allows for a visual comparison of model performance across prompt strategies, helping to identify models that respond well to specific types of prompting.

One might wonder how these results compare to a fully fine-tuned model on the same data. As referenced earlier, fine-tuning even a small 117M GPT-2 on this dataset yielded over 95% F1, and fine-tuning a larger model achieved 97% F1. Those numbers still surpass our prompting results by a clear margin. This highlights that there is room for improvement—possibly by fine-tuning some of these open 70B models on phishing data, one could exceed 95% F1, combining the best of both worlds (knowledge and specialization). However, fine-tuning 70B models is non-trivial due to the computational cost. Our benchmarking provides insight into what can be achieved without any training, which is impressive in its own right: over 90% F1 with zero training using open models.

The following points summarize the performance findings:

- The best prompting method was few-shot, giving the top results around 90–91% F1. While few-shot prompting generally yielded the highest F1-scores across models, this trend was not universal. For example, Llama3_70b achieved a slightly higher F1-score with zero-shot prompting (89.48%) compared to its few-shot counterpart (88.29%). Similar patterns were observed with a few other large models, suggesting that for certain architectures, the inherent pre-training quality may be sufficient to generalize well without additional in-context examples. These exceptions underscore that prompt effectiveness is model-dependent, and few-shot prompting, while often beneficial, should not be assumed as optimal in all cases.
- Model size and quality matter—the 70B models and the well-tuned 27B model (Gemma2) dominated the top of the leaderboard. Some mid-size models like Mistral-small 24B and Gemma2 27B actually rivaled the 70B models in zero-shot and few-shot, showing that a well-designed 20–30B model can be as effective as a generic 70B model for this task. In fact, Mistral 24B had 88.3% F1 zero-shot, slightly above some 70B variants (like llama3.1_70b at 88.15%). This is encouraging as smaller models are cheaper to run. On the flip side, extremely small models (below 7B) struggled—e.g., DeepSeek-1.5B was around 70% F1, which is only slightly better than random guessing on this balanced set, missing many phishing cases. So, there is a parameter threshold (somewhere around 2–3B) beyond which the model has enough capacity/knowledge to perform this task reasonably well. The range of 7–13B models generally landed in the 80–88% F1 range with good prompting (e.g., Qwen-7B, 86–87% F1).
- The prompting strategy impacts are clear: unless a task truly needs step-by-step reasoning, chain-of-thought prompting might be unnecessary and even harmful. Few-shot example-based prompting appears to give consistent gains and should be part of the toolkit for deploying LLMs in phishing detection. Role-based prompting can be tried, but one should verify its effect on the particular model—it is not universally helpful.

### *4.2. Inference Efficiency and Execution Time*

Besides accuracy, a practical system must consider speed and computational cost. We measured how long each model took to process the 1000 URLs under each prompt strategy. Several trends emerged:

- Model size vs. speed: In general, larger models took longer, but the relationship was not strictly linear and was influenced by model optimizations. For example, in the zero-shot setting, a 7B model like Qwen-7B took about 94 s to run all 1000 URLs, whereas the 70B model (llama3_70b) took 488 s. Roughly, the 70B model was five times slower for 10× the parameter count, which is actually an efficient scaling (thanks to optimized matrix multiplication on our hardware). Another 70B variant (llama3.3_70b) was a bit faster at 400 s, possibly due to using 4-bit quantization. Mid-sized models like 27B and 24B took around 250–280 s, and 13–14B models ranged widely (one "phi4_14b" was quite fast at 134s, whereas "phi3_14b" took 967s, indicating differences in their implementations). Notably, one of the smallest models, DeepSeek-R1 1.5B, was extremely slow at 3854 s (over an hour) for 1000 URLs. This is likely due to the lack of optimization (perhaps it was run on CPU or under an inefficient configuration). In fact, all DeepSeek models were outliers: DeepSeek 8B took 10,540 s, 14B took 13,323 s, and the 70B a whopping 33,586 s ( 9.3 h) for just 1000 inferences. This suggests that the DeepSeek models were not using GPU acceleration effectively, or had very slow generation (possibly they were not instruction-tuned and thus took many tokens to produce an answer). By contrast, other models of similar size (e.g., LLaMA 7B, 13B, and 70B) clearly were leveraging optimized inference. In a deployment scenario, one

would avoid using a model that takes seconds per URL if a competitor model can complete it in 0.1 s with similar accuracy.

- Prompt complexity vs. speed: The length of the prompt and output affected runtime. Zero-shot and role-playing prompts are relatively short (just the instruction and URL, and the output is a one-word answer). Few-shot prompts are longer (they include multiple example URLs and labels), which means the model has to process more tokens for each inference, slowing it down somewhat. We found few-shot prompting to incur a minor overhead: on average few-shot runs were 5–10% slower than zero-shot for the same model. For instance, Qwen-7B zero-shot was 94 s, and with few-shot it was 99 s. This overhead is due to the extra tokens in the prompt that the model must read each time, plus possibly longer outputs if it mirrors the example format. Chain-of-thought prompting, however, had a drastic impact on speed. Because CoT prompts encourage the model to generate a reasoning sequence (several sentences) before the final answer, the number of output tokens per query balloons. As a result, inference time roughly doubled (or worse) under CoT prompting for every model. The average total time for 1000 URLs with CoT was 9896 s versus 4770 s in zero-shot (a 2× increase) across models. For example, Llama3.3-70B took 13,264 s with CoT compared to 400 s zero-shot—over 33× slower! Even small models saw their time shoot up: the 3B model went from 106 s with zero-shot to 968 s with CoT, a 9× slowdown. The reasoning text essentially multiplies the work. This clearly demonstrates a trade-off: while CoT might sometimes boost accuracy (as seen in GPT-3.5 earlier [11]), it comes at a heavy cost in speed. In our case, CoT did not even boost accuracy for open models, so it was all cost and no benefit.
- Fastest models: The absolute fastest run in our tests was by one of the LLaMA-derived 8B models (llama3.1_8b) at about 92 s for 1000 URLs. This equates to 0.092 s per URL on average, or about 11 URLs per second, which is quite usable. Qwen-7B was close at 94s. These models likely utilized GPU and had a relatively small context length (prompt + output) to process. Using a larger batch (feeding multiple URLs at once) could further increase throughput, but we performed this one by one for fidelity to the prompt method.
- Slowest models: As mentioned, DeepSeek models were anomalously slow (taking hours). Excluding those, the slowest was phi3_14b at 967 s with zero-shot—possibly that model had some inefficiency. Most others were within 500 s even up to 70B. This indicates that with proper optimization, even a 70B model can process two URLs per second on a high-end GPU. If we needed to scale to millions of URLs per day (10 per second continuously), a single 70B model instance might fall short, but a cluster or a smaller model could handle it.

Figure 5 visualizes the relationship between model size, accuracy, and speed. Figure 5a plots each model's best F1-score (y-axis) against its zero-shot inference time (x-axis, log scale for clarity). We see a general trend: as F1 increases (toward 90%), the time tends to increase (toward hundreds of seconds). The cluster of top-accuracy models (85–91% F1) includes the 70B models and the 27B model, which have moderate to high times. Meanwhile, models that were very fast (under 150s) include 7–8B models which achieved around 82–88% F1. There is a notable diminishing returns effect: to gain that last 5% F1 (going from 85% to 90% F1), one has to use models that are 3–5× slower. This is an important consideration for deployment—if 85% F1 is deemed acceptable, one could use a smaller model and benefit from faster processing (and lower memory usage). But if >90% F1 is required, one might need a large model and possibly to parallelize inference.

Figure 5b shows the dramatic slowdown caused by chain-of-thought prompting: it plots the factor by which each model's time increased under CoT vs. zero-shot. Most

models have a factor between 2× and 10×, with an average around 2.1× for large optimized models and much higher for some smaller ones. This reinforces that CoT should be used sparingly, if at all, in high-throughput systems, unless absolutely necessary for accuracy.

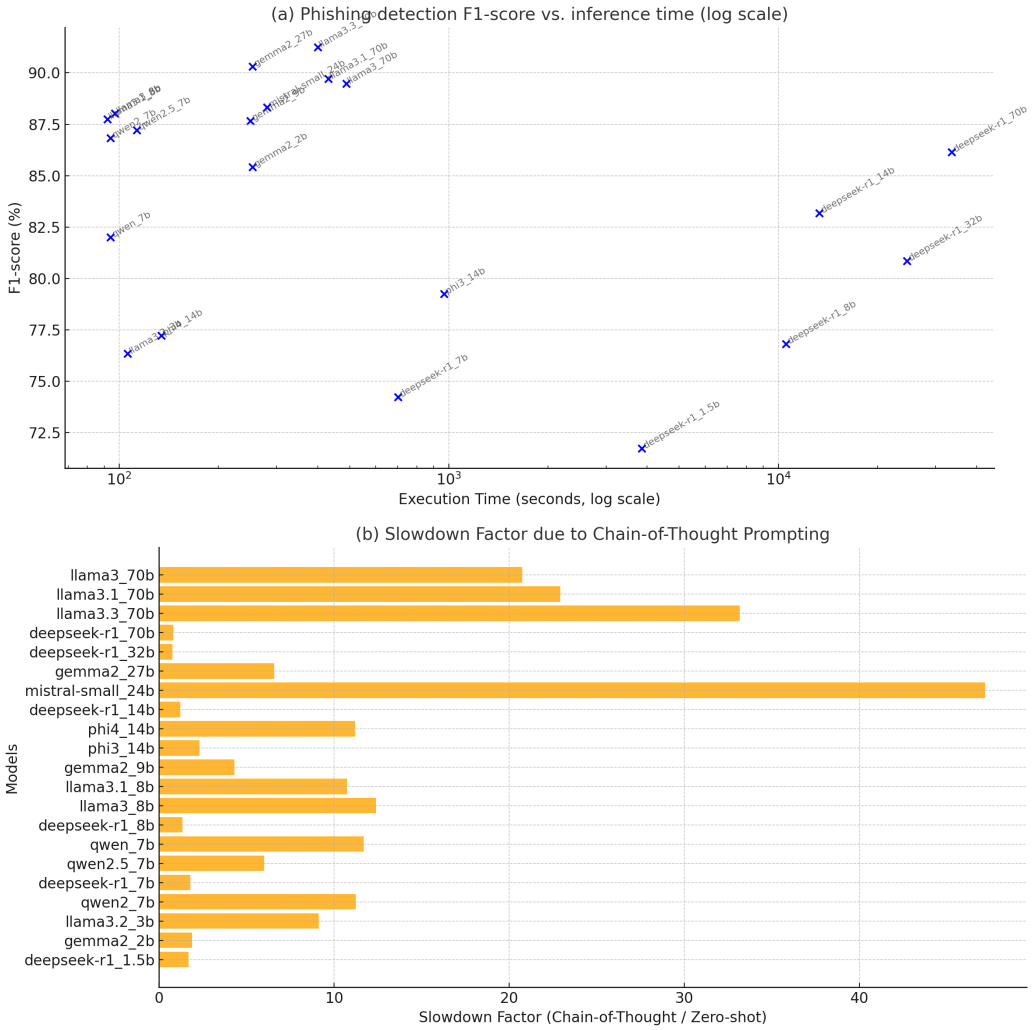


**Figure 5.** (**a**) Phishing detection F1-score versus inference time (log scale) for various models. Each point is a model (with its best-performing prompt). Larger models achieve higher F1 but incur higher runtime. (**b**) Slowdown factor due to chain-of-thought prompting for each model (ratio of CoT time to zero-shot time). Many models exhibit 2× to 5× slower inference with CoT, and some were even an order of magnitude slower.

To put the timings in perspective: the fastest model (around 0.1 s per URL) could potentially scan 600 URLs per minute, or 0.86 million URLs per day on one machine. The slowest useful model (70B at 0.4 s per URL) could scan 216 k per day on one machine. If an email provider needs to check billions of URLs per day, clearly multiple instances or further optimized versions would be needed. Another solution is to adopt a cascaded approach: use a fast lightweight model to filter obvious cases and only send ambiguous cases to a slower, more powerful model. Our data suggest, for example, using a 7B model to achieve 85% F1 quickly, then, maybe only 15% of URLs (the ones it is unsure about) go to the 70B model for a second opinion.

One particular observation from Table 1 combined with times is that Gemma2_27b had a very high F1 (90.31% with few-shot) while its zero-shot time was 254 s (0.254 s/URL)—a decent middle ground. This model seems to offer a sweet spot: it is roughly 5× faster than a 70B model but only 1–2% lower in F1. Similarly, Mistral-24B was extremely good in

zero-shot (88.3% F1) and took 281 s (about 0.28 s/URL). Thus, 20–30B-scaled models can be a great choice when balancing performance and cost.

Finally, we note that memory usage was another practical aspect ( not shown in tables). The 70B models in 4-bit precision consumed around 20 GB of GPU memory, whereas a 7B model can run in under 8 GB easily. So, deploying a large model might require expensive hardware (GPUs with tens of GB of VRAM or running on CPU, which is slow). This further emphasizes exploring smaller models or model compression.

## 5. Discussion

Our benchmarking study provides several key insights for utilizing LLMs in phishing link detection.

### *5.1. Open-Source LLMs Can Achieve High Phishing Detection Performance via Prompting*

We found that the best open models (70B LLaMA-based variants and a well-tuned 27B model) reached 90–91% F1 on a challenging balanced dataset of 1000 URLs. This is on par with, or even exceeded, the performance of some closed models (GPT-3.5) using similar prompts [11]. While the absolute best result (Claude 2 at 92.7% F1 [11]) was not surpassed, the gap is small. This is encouraging because it suggests organizations could use open models without needing API access to proprietary models, and still obtain very accurate results. Moreover, with techniques like fine-tuning or few-shot learning, open models could further improve. It is plausible that an instruction-tuned open 70B model (like LLaMA-2-Chat) fine-tuned on phishing data would even exceed Claude's performance, as hinted by prior fine-tuning results reaching 97% F1 [11].

### *5.2. Prompting Strategies Matter: Few-Shot Outperforms Chain-of-Thought*

Our experiments clearly showed few-shot > zero-shot > role-play > chain-of-thought in terms of average F1. The few-shot prompts likely help the model focus on relevant aspects of the URL by example. In practice, this means that if one is deploying an LLM for such tasks, investing time to craft a good few-shot prompt (with diverse examples of phishing tricks and legitimate cases) is worthwhile. It can especially yield a few extra points of recall. Role-based prompts (e.g., telling the model it is an expert) did not universally help and, in some cases, even hurt. This might depend on how the model was trained—some models might not have been tuned on such "persona" instructions. It may also be that role prompts cause more verbose outputs which complicate parsing. Therefore, we recommend testing a role prompt on a validation set before assuming it will help.

Chain-of-thought prompts, despite their popularity and being beneficial for tasks requiring step-by-step reasoning, performed poorly in phishing URL detection. They massively increased latency (2–5× slower) and decreased accuracy for most models. The intuition is that URL classification does not require heavy reasoning; it is more pattern recognition and memory of known bad strings. Coaxing a model to produce a rationale might distract it from those patterns or cause it to second-guess a correct gut feeling.

Our analysis indicates that this approach encouraged models to generate overly cautious and verbose explanations, often resulting in a high number of false positives (i.e., high recall but low precision), which significantly reduced the F1-score. Specifically, we observed that many models tended to label nearly every URL as "phishing" when asked to reason step by step. This may be because, while reasoning, the models encountered suspicious patterns and defaulted to a conservative stance. For instance, even legitimate URLs were often flagged as phishing, with model responses concluding with statements such as "Therefore, this appears to be a phishing link".

Interestingly, some models—particularly the Gemma2 series—handled CoT prompting better. This may be attributed to these models being fine-tuned to follow instructions while maintaining alignment with the final classification goal, thereby producing reasoned answers without losing track of the task. This observation aligns with the recent literature on the risks of over-explanation and hallucinations in LLM outputs when complex prompting is unnecessarily applied to simple classification tasks [24].

A key takeaway for prompt engineering is that increased prompt complexity does not always translate to improved performance. In our evaluation, the simplest zero-shot prompts already delivered strong results. Moreover, few-shot prompting—where models are shown a few labeled examples—consistently outperformed chain-of-thought prompting across models, without incurring additional reasoning overhead. Thus, for phishing URL detection, prompt strategies that minimize cognitive load on the model while leveraging pattern recognition appear to be more effective.

### *5.3. Model Size vs. Efficiency Trade-Off*

Larger models generally performed better, but are slower. For real-world deployment, one must consider the volume of URLs to scan and the acceptable latency. For example, in an email gateway that needs to come up with a verdict for a URL in, say, under 200 ms to not delay email delivery, using a 70B model might be too slow (0.4 s per URL). A compromise could be using a 7B model which can respond in 0.1 s, at the cost of a few percentage points of accuracy.

To mitigate this speed–accuracy trade-off, one promising approach is knowledge distillation. Here, a large and accurate model (e.g., LLaMA 70B) could be used to generate labeled outputs for a large corpus of URLs. These outputs would then serve as supervision to train a smaller model (e.g., 7B or 13B), effectively transferring the large model's decision patterns into a faster, deployable version.

The open nature of these models means one could attempt techniques like knowledge distillation or ensemble methods to obtain a better speed/accuracy balance. Another complementary approach is ensemble modeling—combining the outputs of multiple mid-sized models (e.g., several 7–13B models) to improve reliability and accuracy without incurring the full latency of a 70B model. Such ensembles could also vote or defer to a slower model only on uncertain inputs, thereby optimizing throughput.

Our results showed that a 27B model was very close to 70B performance, which hints that careful training is more important than sheer size beyond a point. Specifically, models like Gemma2 27B and Mistral-small 24B matched or exceeded 88–90% F1 with far less inference time and memory, suggesting that architectural efficiency and fine-tuning quality are just as critical as scale.

We further acknowledge that deploying 70B models at scale may be impractical for many organizations, especially those lacking access to high-end GPUs or distributed inference infrastructure. These models typically require over 40 GB of GPU memory even with 4-bit quantization, limiting their accessibility. To enable local inference of 70B models, we relied on pre-quantized versions available in the Ollama framework. These models are typically compressed to 4-bit representations (e.g., using GGUF format), reducing memory usage and allowing deployment on consumer GPUs. While quantization may introduce approximation errors, our evaluation showed no significant degradation in F1-score, while achieving substantial reductions in latency and memory footprint. As such, our discussion emphasizes mid-sized alternatives like the 24–27B models, which achieved nearly comparable accuracy with far more efficient memory and latency profiles. Quantization techniques, such as 4-bit GPTQ, and knowledge distillation pipelines, are especially promising for translating performance from large models into deployable formats. In scenarios with

strict latency budgets, a cascading system that first applies a fast lightweight model and defers uncertain cases to a larger model may further help balance resource constraints with detection quality.

If an organization has the computational budget, it might still opt for the largest model to maximize detection (since missing phishing can be very costly). But if the system needs to handle millions of URLs per hour, then using a handful of 13B or 7B models in parallel might be more feasible. These strategies—distillation, ensemble voting, and selective cascading—represent viable paths toward real-world deployment of LLM-based phishing detection under resource constraints.

### *5.4. Limitations and Error Analysis*

While our models achieved high overall metrics, it is important to consider what kinds of URLs they misclassified. LLMs, when used in this manner, essentially act as pattern matchers based on their training data and understanding of URL syntax. They might fail on adversarial or novel obfuscation techniques that were not present in their training. For example, an attacker could include misleading tokens like "secure" or "login-verification" in a URL to make it appear legitimate. If an LLM has learned that "secure" is a positive word, it might wrongly judge the URL as safe. Conversely, if an attacker includes terms that triggered phishing in many training samples, the LLM might flag even a legitimate site that coincidentally has a similar string. The models also might not handle internationalized domain names or homograph attacks well, depending on how URLs were represented in their training data (e.g., a Cyrillic "a" in place of Latin "a"). Robustness to such adversarial examples is a concern. Traditional ML systems deal with this by explicitly checking character encodings or using a blacklist of known malicious domains. LLMs do not have an explicit notion of such security rules unless it was in text that they have read.

The dataset we used, though balanced and well-curated, may not capture the full diversity of the web. Real phishing links might sometimes be part of a longer URL (with trackers, etc.) or come from certain domains that constantly rotate. Also, in a real email, the context (email text, sender) provides additional clues; here we only gave the URL. In practice, an LLM could combine context—for instance, an email saying "Your account is locked, click here"—plus the URL, which might be an even stronger indicator. Our study isolated the URL string to keep the task focused, but future work could involve feeding more context to the model (which could either help or distract—an interesting research question).

Although the dataset is commonly used in phishing research, its curated and static nature introduces potential biases. For instance, it may contain overrepresented phishing templates or domain styles, which could influence LLM pattern learning. Moreover, certain types of phishing—such as those involving non-English domains, heavy URL obfuscation, or dynamic redirection—are underrepresented. We recommend that future evaluations incorporate more diverse, real-world, and multilingual datasets to improve external validity.

Another important limitation is the size of the evaluation set. Our experiments were conducted on a balanced subset of 1000 URLs (500 phishing and 500 legitimate) drawn from a larger, publicly available benchmark dataset. While this subset is sufficient for meaningful benchmarking and comparative analysis across prompt strategies and model families, it may not fully reflect performance in real-world deployments. The decision to use 1000 samples was driven by practical computational constraints: with 21 models and 4 prompting strategies, a full factorial experiment required 84 independent model runs, each incurring non-trivial inference time and GPU cost. Despite the moderate scale, we observed consistent trends across models—particularly in the relative performance of

prompt types and model sizes—suggesting robustness of findings. We acknowledge this as a limitation and encourage future work to expand the empirical base.

### *5.5. Deployment Considerations*

Deploying an open LLM for phishing detection would require integrating it into a pipeline. One concern is consistency—LLMs can sometimes produce varied output phrasing. We mitigated that with careful prompt wording and parsing rules. In a production system, one would likely constrain the model's output (perhaps via a custom head or a regex) to ensure a definitive label is extracted. Another consideration is model updates: open models can be improved or fine-tuned over time. Unlike a static ML classifier, an LLM might actually improve with new releases (for example, a future "Llama 3" model might outperform current ones, and we could drop it in). Our results serve as a baseline for what current (as of 2024–2025) open models achieve. As model architectures improve (especially focusing on being lighter and faster, like the recent Mistral which packs more punch per parameter), we expect this balance to shift—smaller models will inch closer to the performance of the largest. In fact, the existence of multiple 70B open models in our test allowed us to see that not all 70B models are equal—some variants (llama3.3_70b) performed better than others (llama3_70b) in F1. This could be due to different fine-tuning or quantization. It highlights that "size" alone does not guarantee top performance; the quality of training data and fine-tuning matters too.

### *5.6. Future Improvements and Research*

Future work could explore several promising directions to extend the findings of this study. First, while this benchmarking focused on prompt-based classification accuracy and efficiency, future research could incorporate explainability techniques to better understand the behavior of LLMs in phishing detection tasks. For example, post-hoc interpretability methods such as attention visualization, input attribution techniques (e.g., SHAP, LIME), or token-level saliency maps could reveal which features of the URL strings influence model decisions. This would help elucidate whether models rely on semantically meaningful cues (e.g., suspicious subdomains or misleading lexical patterns) or exhibit biases toward superficial heuristics.

Additionally, model-generated rationales—especially under chain-of-thought prompting—could be qualitatively analyzed to identify recurring reasoning strategies or common failure modes. Such analyses may uncover the extent to which models apply valid generalizations versus overfitting to patterns seen during pre-training.

Another promising direction is the development of hybrid explainable systems that combine LLMs with symbolic rules or handcrafted features, allowing predictions to be traced back to both statistical inference and interpretable logic. Integrating an explainability layer into LLM-based phishing detection workflows would enhance transparency, improve stakeholder trust, and potentially guide the design of safer and more accountable model deployments in cybersecurity applications.

One other immediate extension of this work is to explore fine-tuning the best open models on the phishing data and comparing that with prompting. The work by Trad and Chehab suggests fine-tuning yields a large boost [11]. It would be interesting to confirm this with a model like Llama-2-70B-chat—would it go from 91% F1 to 97% F1 after fine-tuning on 10 k phishing URLs? If so, one could then compare that fine-tuned model's performance against the original (to see how much benefit the pre-training gives versus task-specific training). Another direction is adversarial robustness testing: we could take the model and deliberately mutate URLs in ways to fool it (e.g., inserting benign words, or typosquatting

domains) to see where it fails. This could guide the creation of a more robust model, perhaps via adversarial training (training the model on those tricky examples).

To enhance generalizability, future work should explore evaluations on larger-scale datasets, include examples from evolving phishing techniques, and consider multilingual or context-rich phishing attacks. To further improve reliability, future work should consider incorporating cross-validation or resampling techniques to evaluate how results vary across different subsets of phishing data. This would help mitigate sampling bias and allow more robust statistical comparisons between model–prompt combinations. Moreover, while this study employed a fixed evaluation set with deterministic model outputs, it did not include formal statistical significance testing. Future research could benefit from applying non-parametric tests—such as the Wilcoxon signed-rank or Friedman test—to rigorously compare model performance across prompt strategies, especially in studies involving cross-validation or multiple datasets. We acknowledge this as a limitation and encourage future work to expand the empirical base.

In addition, the robustness of open-source LLMs against adversarial attacks remains an open question. Future investigations should explore adversarial prompting, input perturbations, and evasion techniques to assess the security of prompt-based LLM classification systems in phishing detection. Combining LLMs with symbolic or rule-based heuristics could further improve trust and resilience in operational environments.

We also foresee integrating LLM-based URL analysis as part of a larger cybersecurity AI assistant. For example, an LLM could take an email as input and not only classify the URL but also explain why it might be phishing (which is useful for analysts). Some of the chain-of-thought outputs we saw, while not improving raw accuracy, did produce human-readable rationales. If we fine-tune or prompt a model to provide explanations in addition to decisions, we might improve user trust in the system's verdicts. However, care must be taken as LLM explanations can sometimes be convincingly wrong (so-called "hallucinations").

Another future direction is scaling to real-time and large-scale deployment. One could employ model distillation to create smaller models specialized in phishing detection, or use techniques like ONNX optimization or quantization to speed up inference. Since phishing detection is a yes/no classification, one might even convert the LLM into a more traditional classifier by training a lightweight model (like a transformer or even a logistic regression) on embeddings or outputs from the LLM. For instance, one could use the LLM to compute an embedding of the URL (maybe the hidden state after processing it) and then train a classifier on those embeddings. This hybrid approach could retain much of the LLM's knowledge while being faster at runtime.

Lastly, as attackers adapt, we should keep the model up-to-date. Open-source LLMs allow for continuous improvement: one could periodically fine-tune on recent phishing kits or URLs as they emerge (something not possible with a closed API model). This way, the detection system can evolve alongside the threat landscape.

## 6. Conclusions

This comprehensive benchmarking study demonstrates that state-of-the-art open-source LLMs, guided by carefully designed prompt engineering strategies, can effectively detect phishing URLs with performance comparable to proprietary models. Practically, organizations can confidently deploy open-source models as phishing detection filters without relying on costly API-driven closed models, benefiting from transparency, lower operational costs, and ease of adaptation to emerging threats. Future work includes periodic model updates with new phishing data to maintain robustness against evolving threats and exploration of fine-tuning to further enhance detection accuracy.

In this paper, we conducted an in-depth benchmarking of 21 open-source large language models on the task of phishing URL detection using prompt engineering. We demonstrated that with appropriate prompting—especially few-shot example-based prompts—open LLMs can identify malicious URLs with high accuracy (exceeding 90% F1 in the best cases), approaching the performance of top-tier proprietary models. We analyzed four prompting strategies and found that few-shot prompting consistently provided the best results, while chain-of-thought prompting was counterproductive for this classification task, often reducing accuracy and greatly increasing inference time. We also highlighted the trade-offs between model size and efficiency: smaller 7–13B models offer faster inference with only a modest hit in accuracy, whereas the largest 70B models give the highest accuracy at significantly higher computational cost.

Our results suggest that organizations have viable options to deploy AI-driven phishing detection without relying on black-box services. An ensemble of open models, or a suitably fine-tuned open model, can serve as a powerful phishing filter. However, careful consideration must be given to the system design—from prompt construction to managing latency and ensuring robustness against adversarial inputs. There is no one-size-fits-all solution: the choice of model and prompt may depend on the specific requirements (e.g., an enterprise email gateway might prioritize recall and accept more false positives, whereas a browser might want higher precision to avoid annoying users).

Future work includes exploring how incremental training (fine-tuning) of these models on fresh phishing data can further boost performance, and how to mitigate any failure modes observed (such as biases or blind spots in the LLM's knowledge of URLs). We also plan to investigate the use of LLMs for alert explanation—providing reasoning to users or security analysts about why a URL was flagged, which can be as important as the decision itself in practice. Additionally, expanding the scope to other related threats, like detecting phishing emails or web content with LLMs, could leverage their broader language understanding capabilities beyond just URLs.

In conclusion, our study shows that large language models—a cornerstone of modern NLP—have a strong role to play in cybersecurity. By benchmarking a wide array of open models, we provide insights into how these models can be harnessed for phishing detection. As both AI and phishing tactics evolve, continued research at their intersection will be vital. Using AI to fight AI-powered phishing (attackers are increasingly using tools like ChatGPT to craft phishing lures) creates an arms race in which staying at the cutting edge of model capability and prompt strategy is essential. We hope this work serves as a foundation and reference point for developing next-generation phishing defenses that are accurate, efficient, and transparent.

**Author Contributions:** Conceptualization, A.H.N. and W.M.; methodology, A.H.N. and W.M.; software, A.H.N.; validation, A.H.N., W.M., and A.O.; formal analysis, A.H.N., W.M., and A.O.; investigation, A.H.N., W.M., and A.O.; resources, A.H.N. and W.M.; data curation, A.H.N.; writing—original draft preparation, A.H.N.; writing—review and editing, A.H.N. and W.M.; visualization, A.H.N.; supervision, A.H.N., A.O., and Y.M.; funding acquisition, A.H.N. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research is supported by Universitas Islam Riau under research grant number 939/KONTRAK/P-K-KI/DPPM-UIR/10-2024.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** The data presented in this study are openly available at Mendeley Data https://data.mendeley.com/datasets/c2gw7fy2j4/3

**Conflicts of Interest:** The authors declare no conflicts of interest.

## References


1. IBM Security. Cost of a Data Breach Report 2024, 2024. Available online: https://www.ibm.com/reports/data-breach (accessed on 31 March 2025).
2. Sahingoz, O.K.; Buber, E.; Demir, O.; Diri, B. Machine learning based phishing detection from URLs. *Expert Syst. Appl.* **2019**, *117*, 345–357.
3. Zhao, X.; Langlois, K.; Furst, J.; McClellan, S.; Fleur, R.; An, Y.; Hu, X.; Uribe-Romo, F.; Gualdron, D.; Greenberg, J. When LLM Meets Material Science: An Investigation on MOF Synthesis Labeling. In Proceedings of the 2023 IEEE International Conference on Big Data (BigData), Sorrento, Italy, 15–18 December 2023; pp. 6320–6321. https://doi.org/10.1109/BigData59044.2023.10386438.
4. Hu, H.; Yan, J.; Zhang, X.; Jiao, Z.; Tang, B. Overview of CHIP2023 Shared Task 4: CHIP-YIER Medical Large Language Model Evaluation. In *Communications in Computer and Information Science*; Springer Nature Singapore: Singapore, 2024; pp. 127–134. https://doi.org/10.1007/978-981-97-1717-0_11.
5. Schur, A.; Groenjes, S. Comparative Analysis for Open-Source Large Language Models. In *Communications in Computer and Information Science*; Springer Nature Switzerland: Cham, Switzerland, 2024; pp. 48–54. https://doi.org/10.1007/978-3-031-49215-0_7.
6. Raiaan, M.A.K.; Mukta, M.S.H.; Fatema, K.; Fahad, N.M.; Sakib, S.; Mim, M.M.J.; Ahmad, J.; Ali, M.E.; Azam, S. A Review on Large Language Models: Architectures, Applications, Taxonomies, Open Issues and Challenges. *IEEE Access* **2024**, *12*, 26839–26874. https://doi.org/10.1109/ACCESS.2024.3365742.
7. Ahmed, T.; Piovesan, N.; De Domenico, A.; Choudhury, S. Linguistic Intelligence in Large Language Models for Telecommunications. In Proceedings of the 2024 IEEE International Conference on Communications Workshops (ICC Workshops), Denver, CO, USA, 9–13 June 2024; pp. 1237–1243. https://doi.org/10.1109/ICCWorkshops59551.2024.10615609.
8. Hidayat, F.; Nasution, A.H.; Ambia, F.; Putra, D.F.; Mulyandri. Leveraging Large Language Models for Discrepancy Value Prediction in Custody Transfer Systems: A Comparative Analysis of Probabilistic and Point Forecasting Approaches. *IEEE Access* **2025**, *13*, 65643–65658. https://doi.org/10.1109/ACCESS.2025.3560254.
9. Khalila, Z.; Nasution, A.H.; Monika, W.; Onan, A.; Murakami, Y.; Radi, Y.B.I.; Osmani, N.M. Investigating Retrieval-Augmented Generation in Quranic Studies: A Study of 13 Open-Source Large Language Models. *Int. J. Adv. Comput. Sci. Appl.* **2025**, *16*. https://doi.org/10.14569/IJACSA.2025.01602134.
10. Nasution, A.H.; Onan, A. ChatGPT Label: Comparing the Quality of Human-Generated and LLM-Generated Annotations in Low-Resource Language NLP Tasks. *IEEE Access* **2024**, *12*, 71876–71900. https://doi.org/10.1109/ACCESS.2024.3402809.
11. Trad, F.; Chehab, A. Prompt engineering or fine-tuning? a case study on phishing detection with large language models. *Mach. Learn. Knowl. Extr.* **2024**, *6*, 367–384.
12. Zhang, L.; Lin, Y.; Yang, X.; Chen, T.; Cheng, X.; Cheng, W. From sample poverty to rich feature learning: A new metric learning method for few-shot classification. *IEEE Access* **2024**, *12*, 124990–125002.
13. Abdelhamid, N.; Ayesh, A.; Thabtah, F. Phishing detection based on rough set theory. *Expert Syst. Appl.* **2014**, *41*, 5948–5959. https://doi.org/10.1016/j.eswa.2014.03.019.
14. Koide, T.; Fukushi, N.; Nakano, H.; Chiba, D. Chatspamdetector: Leveraging large language models for effective phishing email detection. *arXiv* **2024**, arXiv:2402.18093.
15. Heiding, T.; Schiele, T.; Reuter, C. Large Language Models for Phishing Email Detection: GPT-4 vs. Humans. In Proceedings of the 2023 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), Delft, The Netherlands, 3–7 July 2023; pp. 269–275. https://doi.org/10.1109/EuroSPW60778.2023.00046.
16. Trad, B.; Chehab, L. Prompting Large Language Models for Phishing URL Detection. *arXiv* **2023**, arXiv:2311.01786.
17. Hannousse, A.; Yahiouche, S. Towards benchmark datasets for machine learning based website phishing detection: An experimental study. *Eng. Appl. Artif. Intell.* **2021**, *104*, 104347.
18. Haq, Q.E.u.; Faheem, M.H.; Ahmad, I. Detecting Phishing URLs Based on a Deep Learning Approach to Prevent Cyber-Attacks. *Appl. Sci.* **2024**, *14*, 10086.
19. Heiding, F.; Schneier, B.; Vishwanath, A.; Bernstein, J.; Park, P.S. Devising and detecting phishing: Large language models vs. smaller human models. *arXiv* **2023**, arXiv:2308.12287.
20. Nicklas, F.; Ventulett, N.; Conrad, J. Enhancing Phishing Email Detection with Context-Augmented Open Large Language Models. In Proceedings of the Upper-Rhine Artificial Intelligence Symposium, Offenburg, Germany, 13th–14th November 2024; pp. 159–166.
21. Liu, R.; Geng, J.; Wu, A.J.; Sucholutsky, I.; Lombrozo, T.; Griffiths, T.L. Mind your step (by step): Chain-of-thought can reduce performance on tasks where thinking makes humans worse. *arXiv* **2024**, arXiv:2410.21333.
22. Hannousse, A.; Yahiouche, S. Web page phishing detection. *Mendeley Data* **2021**, *3*, 2021.

23. Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Xia, F.; Chi, E.; Le, Q.V.; Zhou, D.; et al. Chain-of-thought prompting elicits reasoning in large language models. *Adv. Neural Inf. Process. Syst.* **2022**, *35*, 24824–24837.
24. Wang, Y.; Liu, X.; Peng, N. The Pitfalls of Chain-of-Thought Prompting: Contamination, Misinterpretation, and Overthinking. *arXiv* **2024**, arXiv:2401.01482.