Review Article

Assessment of the Practicality of LLMs in the Field of Cybersecurity and Detection of Malicious Code

Shreyas Illindala[1], Samar K. Sabie[2]

[1]*Cheshire High School, 525 South Main Street, Cheshire, Connecticut, 06410, USA;*
[2]*Institute of Communication, Culture, Information and Technology, University of Toronto, Canada*

ABSTRACT

Large Language Models (LLMs) hold significant promises to revolutionize cybersecurity. Unlike traditional machine learning (ML) models, LLMs can be fine-tuned for specific tasks such as identifying malicious code, analyzing software vulnerabilities, and detecting phishing attacks. However, challenges remain, including the high computational cost of development, potential biases in training data, and the risk of adversarial attacks against these models. This review examines the practicality of using LLMs in cybersecurity with current technologies over the next five years. It evaluates their real-world applicability and draws on recent literature to highlight potential security threats, benefits, and implementations of LLMs in this domain. These studies provide examples where LLM-based tools have been both effective and flawed, helping establish precedents for LLM use. Additionally, this review discusses the history and rapid development of LLMs, comparing current advances to past technological growth, and explores future research directions for integrating LLMs into cybersecurity.

Keywords: cybersecurity, malicious code,Large Language Models, machine learning, Information Flow Control, Primitive Intrusion Detection

INTRODUCTION

LLMs have emerged as powerful tools in cybersecurity due to their ability to learn from vast amounts of data and adapt to evolving threats. In theory, an LLM fine-tuned on security data can analyze code or network traffic to detect malicious patterns far beyond the capability of static signature-based systems. For example, an LLM could automatically flag suspicious code snippets or phishing email content by recognizing subtle linguistic or structural cues (1). This potential to augment defensive capabilities has generated excitement about moving beyond traditional rule-based detection toward AI-driven security. At the same time, deploying LLMs for cyber defense comes with significant challenges. These models require extensive computing resources and are prone to false positives; AI models have historically tended to flag benign behavior as malicious, especially when classifying nuanced code or behaviors. Additionally, adversaries may exploit LLMs' vulnerabilities (for instance, by crafting inputs that the model fails to recognize as malicious). As a result, organizations are cautious about relying entirely

Corresponding author: Shreyas Illindala, E-mail: sksillindala@gmail.com.

Received June 15, 2025**; Accepted** August 1, 2025
https://doi.org/10.70251/HYJR2348.34177186

on LLMs for critical security tasks. This review provides an overview of how LLMs can be applied in cybersecurity and examines their strengths, limitations, and the emerging research on using LLMs to detect malicious code (1-4).

What is a malicious attack?

When applying LLMs in cybersecurity, one must consider the various forms of malicious attacks that these AI systems might face or help defend against. Distributed Denial of Service (DDoS) attacks, for example, aim to flood networks or services with traffic to disrupt their normal functioning. Another common threat is malware – malicious software such as viruses or trojans that infiltrate systems. LLM-based security systems, like any other, need effective strategies to address these threats. In modern cybersecurity operations, cyber threat intelligence (CTI) plays a crucial role in identifying, analyzing, and mitigating risks by anticipating how cybercriminals might execute attacks and developing counter-strategies. Key defensive mechanisms include intrusion detection systems (IDS), which monitor networks or systems for unauthorized access or abnormal behavior, enabling early identification of potential threats. The concept of intrusion detection dates back to a 1972 paper by James P. Anderson for the United States Air Force (USAF), which helped shape the evolution of modern intrusion detection practices (2). Traditional IDS often rely on signature-based methods – looking for known patterns of malicious activity – but these are increasingly vulnerable to evasion as attackers modify their tactics. This has led to a growing interest in AI-driven anomaly detection that can adapt to new threats (3). LLMs have garnered widespread media attention for their advanced capabilities, sparking enthusiasm that they could transform cybersecurity. There is hope that AI-based systems will reduce reliance on long-standing statistical modeling techniques that have been used to detect malicious code for years. However, the practical nature and side effects of using LLM technology in security are under scrutiny. Companies often still lean on proven statistical models to avoid the false positives that plague AI models. In other words, an LLM might incorrectly flag safe software as malicious, which is unacceptable in production environments. Moreover, there is the risk of adversarial AI – attackers may find ways to manipulate an LLM into misclassifying malicious code as benign or even extract sensitive information from the model's training data. These threats underscore that, as of today, replacing traditional security tools with LLMs outright is risky (2, 5).

What is fine-tuning, and what are the different types of fine-tuning?

Fine-tuning is a key process that allows LLMs to specialize in particular tasks or domains, distinguishing them from many other AI models. An LLM is first pre-trained on a broad corpus (for general language understanding) and then fine-tuned on task-specific data so it can perform a narrower job, like malicious code detection. There are various forms of fine-tuning, each suited to different scenarios:

- Supervised fine-tuning: Continuing to train the model on a labeled dataset for the target task (e.g., a corpus of code labeled as malicious). The model learns directly from examples with correct answers.
- Reinforcement Learning from Human Feedback (RLHF): An advanced fine-tuning approach in which model outputs are ranked or scored by humans, and reinforcement learning is used to update the model to prefer higher-rated responses. This has been used to align LLMs with human preferences and could, for instance, help an LLM learn to avoid insecure coding suggestions.
- Parameter-efficient fine-tuning: Techniques that adjust only a small portion of the model's parameters, rather than all of them, to reduce computational requirements. This category includes methods like prefix tuning and prompt tuning, among others.

In practice, fine-tuning enables developers to use LLMs as powerful engines that generalize information for specific needs. Notably, there are both open-source LLMs (where the model architecture and sometimes weights are openly available) and closed-source LLMs (proprietary models from companies like OpenAI). Open-source models allow researchers to apply custom fine-tuning and inspect the model's inner workings, whereas closed-source models limit such transparency and flexibility.

Prefix tuning and prompt tuning are two parameter-efficient fine-tuning strategies often discussed. In prefix tuning, the model's original parameters remain frozen; instead, the training optimizes a set of small task-specific vectors (the "prefix") that are prepended to the model's inputs. This approach injects task-relevant information without altering the core model weights. Prompt tuning, similarly, involves learning adjustable "soft prompts" that guide the model on a task by being added to the input, also without changing the model's main parameters. Both methods drastically reduce the number of parameters that need to be trained, making fine-tuning more

efficient. These techniques are valuable in cybersecurity applications where one might need to quickly adapt a large model to detect a new type of malware or phishing technique without retraining the entire model.

Another concept often mentioned is prompt engineering – creating or refining the textual prompts given to an LLM to elicit better performance. Prompt engineering is not a training method per se, but rather a way of coaxing the model to produce the desired output by carefully phrasing queries or instructions. It is an important practical tool, especially when one cannot fine-tune a model directly (as is the case with many closed-source APIs) (2, 6, 7).

Transformers, the underlying architecture for modern LLMs, also deserve mention. A transformer model typically consists of an encoder and decoder (or only an encoder, or only a decoder, depending on the variant). These models use mechanisms of self-attention to process input text. (It's worth noting that a statement in earlier literature compared transformers to having a "generator" and a "discriminator" component, but that terminology is more applicable to Generative Adversarial Networks. In transformers like GPT, the terms "generator" and "discriminator" are not used in the same sense; GPT is a generative model that predicts text and does not internally have a separate discriminator judging output realism as a GAN would. The confusion likely arose from naming conventions; for clarity, GPT stands for Generative Pre-trained Transformer, a model known for robust text generation.) (6, 8, 9).

A major benefit of LLMs in security contexts is that they can operate without constant human supervision. Once fine-tuned, an LLM can run continuously and autonomously flag or even mitigate threats, something that would be impractical with humans in the loop for every decision. The vision is that a general-purpose model could be fine-tuned for cybersecurity and then used with minimal direct instructions. It could leverage *few-shot prompting* (providing a few examples or reasoning steps in the prompt) techniques such as Chain-of-Thought (CoT) prompting or ReAct (Reason+Act) prompting to break down complex threat analysis tasks. Few-shot prompting gives the model guidance on *how* to approach a problem without explicitly coding a solution, thus allowing the LLM to generalize from a small number of examples or cues (7, 10).

After fine-tuning, developers can deploy LLMs in security pipelines to perform tasks like triaging alerts, analyzing suspicious code, or generating defensive strategies, with the LLM deducing the solution approach from its training and the prompt. This is powerful, but it again serves to emphasize the importance of the quality of the training data and prompts; an LLM will only be as good at detecting threats as the information and guidance it has been given.

LoRA (Low-Rank Adaptation of LLMs) and QLoRA (Quantized LoRA) are two advanced techniques for parameter-efficient fine-tuning that have significant practical implications for cybersecurity applications. LoRA inserts small trainable adapter modules within the transformer layers and freezes the rest of the model's weights. Essentially, LoRA learns a low-rank representation of the model updates required for the task. This drastically reduces the number of parameters that need to be trained for a new task, making fine-tuning large models feasible even on smaller hardware setups. After training, the LoRA adapters can be merged with the original model weights, effectively integrating the new knowledge. QLoRA (Quantized LoRA) is an extension of this idea that further improves efficiency by using a *quantized* version of the base model. In QLoRA, the pre-trained model's weights are compressed to a lower precision (such as 4-bit), which significantly lowers the memory footprint. The model is then fine-tuned with LoRA adapters on this quantized backbone. Remarkably, QLoRA achieves nearly the same performance as full fine-tuning the original model, yet it requires far less GPU memory and computing power. This is particularly significant for cybersecurity, where organizations may need to fine-tune large models on domain-specific data (like network logs or malware code) but lack the resources to run full-scale training. By using QLoRA, a security team could economically train an LLM to detect malicious code patterns or anomalous behavior without needing a supercomputer. In summary, LoRA and QLoRA make it practical to customize powerful LLMs for cybersecurity tasks even under resource constraints, bringing the benefits of fine-tuned models within reach of more organizations. To summarize fine-tuning approaches, Table 1 provides a comparison of several methods and their characteristics (11, 17).

Applications of LLMs in Cybersecurity

Thanks to their versatility, LLMs are being tested in numerous cybersecurity applications, including secure code generation, scam detection, and anomaly detection techniques. Secure code generation is an especially promising but challenging application. An LLM like GitHub's Copilot can assist developers by suggesting code, potentially helping to write more secure software. However, if an LLM generates code with vulnerabilities

Table 1. Comparison of LLM Fine-Tuning Methods and Their Characteristics

Fine-Tuning Method	Approach	Advantages and Use Cases in Cybersecurity
Full fine-tuning	Update all model weights on task-specific data.	Maximizes performance on new task, but requires huge computational resources. Useful when task-specific data is abundant and critical (e.g., thoroughly training an LLM to detect malware patterns), but often impractical for very large models.
Supervised fine-tuning	Train on labeled examples of the task output.	Straightforward and effective when high-quality labeled security data (e.g. known malicious vs. benign code) is available. The model directly learns from examples, improving accuracy on similar inputs.
RLHF (Reinforcement Learning from Human Feedback)	Use human feedback to reward or penalize model outputs, refining the model via reinforcement learning.	Aligns the model with expert preferences or ethical guidelines. In cybersecurity, experts could guide an LLM to avoid false positives or inappropriate actions by providing feedback on the model's alerts or responses.
Prefix tuning	Keep model weights fixed; learn small prefix vectors that condition the model.	Requires tuning only a few parameters, making it memory-efficient. Useful for quickly adapting an LLM to a new security task (like a new malware family) with limited computing resources.
Prompt tuning	Keep model weights fixed; learn soft prompt embeddings that steer the model.	Also very lightweight. Can be deployed to adjust a model's behavior on a new task without full retraining. For instance, tuning a "prompt" to make an LLM better at generating secure code snippets.
LoRA (Low-Rank Adaptation)	Insert trainable low-rank adapter matrices into the model layers, keeping original weights frozen.	Highly efficient in terms of trainable parameters. Allows fine-tuning large models on specific cybersecurity tasks (e.g., intrusion log analysis) using modest hardware. Adapters can be merged back, so deployment remains a single model.
QLoRA (Quantized LoRA)	Fine-tune using LoRA on a 4-bit quantized version of the model.	Ultra-efficient fine-tuning with minimal performance loss. Makes it feasible to tailor very large LLMs (billions of parameters) to security tasks on limited hardware, democratizing access to custom LLMs for cybersecurity.

or logic errors and that code is deployed unchecked, it could introduce security flaws. The goal, therefore, is fail-safe code generation – using LLMs to produce code that is correct and secure by design. In practice, any AI-generated code needs human oversight; without it, mistakes could "devastate the system," causing new insecurities (10). Researchers are exploring ways to make LLM-generated code more trustworthy. One intriguing approach is watermarking AI-generated code to identify its origin. For instance, when an LLM generates code, a hidden watermark can be embedded. If that code later appears in malware, the watermark remains detectable. This aids in accountability and forensics by tracing malicious code back to an LLM source. In fact, recent work demonstrated that by using watermarks, hundreds of LLM-generated malware samples could be successfully identified, which corresponds to a high detection success rate (11). This kind of technique helps investigators and could deter threat actors from abusing open-source LLMs to mass-produce malware (8).

LLMs are also being used to counter social engineering attacks like phishing scams. Because phishing emails often follow certain patterns or use familiar deceptive language, an LLM can be fine-tuned on a dataset of phishing examples to recognize and flag suspicious messages. In a 2023 report, Cloudflare noted that phishing remains one of the most dominant threats, with deceptive links being the most common lure; out of 13 billion emails the company processed from May 2022 to May 2023, about 250 million malicious messages were blocked for phishing content. A recent study, Detecting Scams Using Large Language Models, outlined a method

for training LLMs to understand the language used in phishing attacks, making it easier to automatically filter them out. Initial results are promising – such AI systems can catch many phishing attempts that simpler filters miss (12). Another line of research proposes using adversarial techniques against the attackers: for example, training LLMs to generate variations of phishing emails that might evade detection, and then improving filters based on those hard-to-detect examples. This creates a cat-and-mouse dynamic where LLMs help bolster defenses by anticipating attacker tactics.

Beyond detection, LLMs are being considered for more proactive defense roles. For example, an LLM-based assistant could analyze an organization's network logs in natural language, explaining potential incidents to human analysts in plain terms. LLMs could also help with threat hunting – scanning through large datasets of system events to find anomalies that might indicate a breach, then summarizing those findings. These applications take advantage of LLMs' strengths in pattern recognition and language, but again, careful fine-tuning and validation are required to ensure the AI's suggestions are accurate and actionable.

HISTORICAL APPROACHES TO SOLVING DATA SECURITY ISSUES

To appreciate what LLMs bring to the table, it's useful to review how cybersecurity threats have traditionally been handled. The classic approach to detecting intrusions or malware in a system relies on monitoring and analyzing activity for known signatures of malicious behavior. Tools that implement this are broadly known as Intrusion Detection Systems (IDS). A traditional IDS might use a database of malware signatures or network traffic patterns (like a specific sequence of bytes that indicates a known virus, or a known malicious domain name in web traffic) and flag any occurrences of those in the monitored environment. Signature-based detection is effective for known threats but struggles with new, unseen attacks. Attackers can often evade these systems by making slight modifications – for instance, changing a few bytes in a virus's code so its signature no longer matches the known pattern, or using polymorphic techniques to continually change the malware's appearance (2).

Over time, adversaries have grown adept at outsmarting signature-based defenses. In response, cybersecurity has gradually shifted toward behavior-based detection. Instead of looking only for a specific signature, behavior-based systems monitor for suspicious activities or anomalies in how software and users behave. For example, if a program that never accesses the internet suddenly starts connecting to an overseas server, a behavior-based system would flag that, even if the program's hash or signature isn't known to be malicious. This approach is more adaptive because it can catch novel threats that don't match any known signature (5).

A contemporary example of behavior-based detection is ExtraHop's Reveal(x) network detection and response system. Reveal(x) combines traditional rule-based detection with behavioral analytics to identify threats based on their operational characteristics, not just static identifiers. For instance, Reveal(x) can decrypt network traffic (where permitted) and inspect the payload and patterns of communication rather than just looking for a static indicator like an IP address or hash. In one case study, this approach proved effective at catching advanced threats such as TrickBot malware, which is known for employing evasive techniques. By focusing on how TrickBot operates (its behaviors on the network) rather than just its code signature, Reveal(x)'s rule + behavior methodology was able to detect it even when the malware altered its binary to avoid signature detection (14).

However, even behavior-based systems have limitations. They can generate false alarms if an authorized user behaves in an unusual but not actually malicious way. Traditional systems might miss this, but an AI could potentially detect subtle anomalies. This is where AI and machine learning have started to augment intrusion detection. Machine learning models can learn the normal patterns of system usage and then spot deviations that could indicate an insider threat or a malware operating under the radar. If an attacker has stolen valid credentials and is operating under an authorized user account, signature-based IDS likely won't catch anything (because the actions might not match a known attack signature). But an ML-driven system might notice that the account is now accessing resources at midnight and downloading large databases, whereas historically it never did so, which is a red flag.

In summary, historical cybersecurity defenses evolved from static, signature-based methods to more dynamic, behavior-based strategies as attackers became more sophisticated. This evolution set the stage for AI-driven solutions: LLMs and other advanced models represent the next leap, aiming to detect threats by understanding context and intent, not just patterns. In the following sections, we discuss how concepts like information flow control and adversarial attack techniques intersect with these AI-driven approaches.

INFORMATION FLOW CONTROL AND PRIMITIVE INTRUSION DETECTION

One classical concept in system security, predating modern AI, is Information Flow Control (IFC). IFC is about governing how data moves through a system to ensure confidentiality and integrity are preserved. Under IFC, every data item or variable is assigned a security level or label (for example, "public" or "private"), and the system is designed so that information can only flow in ways that don't violate security policies (typically, sensitive data shouldn't flow to less secure domains). This concept originates from multi-level security research in defense settings. In practice, IFC enforcement can prevent, say, a secret document's content from unintentionally flowing into a public output (which could catch certain types of malwares trying to exfiltrate data). Early intrusion detection mechanisms often incorporated such principles, ensuring that when an unexpected flow occurred (e.g., a high-classification process trying to send data to a low-classification network), it was flagged as a potential security breach.

Primitive intrusion detection techniques were closely tied to these strict policies and simple rule checks. For example, a rudimentary intrusion detection setup might log and alert on any login outside of business hours, or any use of an administrative command by a non-admin user. These rules are easy to implement but also easy for legitimate activities to accidentally trigger (causing false positives), and conversely, easy for sophisticated attackers to work around.

As threats have evolved, so have techniques for detecting them. Malicious activity in cybersecurity now often involves deception and obfuscation – attackers hide their true intent using various methods. Classic social engineering tricks (psychological manipulation) remain effective, but attackers also use technical obfuscation. For instance, early AI models could be tricked by simple jailbreak prompts – specially crafted inputs that bypass the model's content filters and cause it to produce disallowed output. However, a recent study highlighted that as LLMs like GPT have improved their defenses, the success rate of those same jailbreak prompts has declined. This indicates a kind of arms race: as AI models get better at blocking known malicious inputs, attackers must devise new methods (15-18).

ADVERSARIAL THREATS TO LLMS IN CYBERSECURITY

One of the emerging challenges is adversarial attacks on LLMs themselves. These are techniques where an attacker presents input to an AI model that is designed to mislead or exploit it. A common method is paraphrasing attacks – the attacker takes a malicious input (say, a piece of malware code or a banned prompt) and paraphrases or transforms it so that it looks innocuous to the AI, even though its essence is the same. For example, an attacker might use round-trip translation: take a malicious script, translate it to another language and back to English, thereby changing the wording but not the harmful intent. An LLM security filter that was looking for specific keywords might miss the transformed version. Similarly, attackers have used LLMs themselves to generate obfuscated versions of malicious content – this can be termed LLM-based paraphrasing, where a model rephrases or reformats malicious instructions to evade detection (9).

Adversarial attacks exploit weaknesses in AI by feeding manipulated input that the system wasn't explicitly trained to handle. To function effectively, LLMs in cybersecurity must be trained on vast datasets, including examples of these tricky inputs. Yet obtaining such data (especially examples of cleverly obfuscated attacks) is hard, and training on negative data (malicious inputs) raises ethical and security challenges in itself. A recent industry survey found that over 60% of AI-driven businesses acknowledged AI technologies pose significant cybersecurity concerns to themselves. This underscores that as organizations adopt AI, they must remain vigilant that the AI systems don't become new attack vectors or points of failure.

Phishing, as mentioned, is a prevalent threat that combines both technical and human deception. LLMs can assist in detecting phishing content, but they can also be used by attackers to generate extremely convincing phishing messages at scale. In fact, it has been demonstrated that OpenAI's GPT-3.5 and GPT-4 models can be guided to produce tailored phishing emails for a very low cost – on the order of fractions of a cent per email. This lowers the barrier for launching large-scale phishing campaigns. An attacker could, for instance, prompt an LLM to generate a unique phishing email for each target (to evade spam filters that might catch bulk-sent identical emails). The cost and effort per email become negligible, potentially leading to an increase in phishing volume.

Attackers may also try to jailbreak LLM-based security systems by crafting inputs that bypass content filters or exploit model quirks. For example, inserting certain obscure Unicode characters, or phrasing a request as a hypothetical scenario, might trick a model into ignoring its safety training. Researchers developed the

IntentObfuscator framework to illustrate how ambiguity can be systematically injected into malicious prompts. By hiding the malicious intent under layers of benign or confusing text, an attacker can cause an LLM to misjudge the input. Attackers have even resorted to creative approaches like role-playing scenarios, where the input convinces the model that it's just acting out a role, thus bypassing safeguards that would normally prevent it from, say, giving instructions to build malware [2, 19].

To uncover such weaknesses, security researchers use tools like GPTFUZZER, an automated "red team" tool that generates many variations of inputs to test an LLM's defenses. GPTFUZZER can systematically find prompts that cause the model to fail or produce insecure outputs, which helps developers patch those vulnerabilities (2, 20).

The range of adversarial attacks against LLMs is broad. Some attacks involve adding specific token sequences to input (sometimes called adversarial suffixes or triggers) that cause the model to malfunction. Others exploit language gaps – for instance, giving instructions in a language or dialect the model might not have robust filtering for, as non-English or code-mixed prompts have been shown to bypass some moderation systems. In one approach, researchers suggested using a Bayesian Regression Tree model to automatically generate paraphrased attacks – essentially creating numerous synthetic "mutations" of a malicious prompt that all have the same meaning but different wording. By testing an LLM with these, they can identify how much an input must be obfuscated to slip past the model's defenses. The Obfuscation Degree Metric was introduced to quantify how different an obfuscated prompt is from the original malicious prompt, giving defenders a way to measure an attack's stealthiness.

On the defensive side, several strategies are under development to harden LLMs against such attacks. One method called SmoothLLM aims to protect models from adversarial suffix tricks by injecting random noise (perturbations) into prompts during processing. By slightly altering the input (e.g., misspelling some words or encoding parts of the text) in various ways, SmoothLLM makes it harder for an attacker's specific crafted sequence to have the intended effect, since the sequence won't match exactly due to the random changes. This approach is somewhat analogous to adding random jitter to a system so that an attacker can't reliably exploit timing – here it's adding noise so an attacker can't reliably exploit a specific input pattern. Early results show that SmoothLLM can prevent some known adversarial prompts from bypassing filters (2, 21).

Another concept is to use "ensemble" detectors or zero-shot detection methods that attempt to identify malicious content without having seen examples during training. OpenAI, for instance, released a classifier to detect AI-generated text as a way to identify misuse, but it was found to be not very accurate (around 26% success) and was soon discontinued. Similarly, OpenAI and others have proposed using LLMs themselves to detect malicious usage (like a language model that flags outputs of another model), and fine-tuning models specifically as detectors. Unfortunately, follow-up research showed these solutions to be ineffective or not robust against determined attackers. It turns out that if you know a detector is in place, you can often adjust the malicious output slightly to evade it, especially if the detector isn't extremely thorough or if it tries to work in a zero-shot manner (without extensive training on what it's looking for).

New offensive techniques against LLMs emerge regularly, but so do new defensive mechanisms. For example, as tools like IntentObfuscator and GPTFUZZER reveal weaknesses, they inform researchers how to patch models or training procedures. Conversely, as defenses like SmoothLLM and advanced content filters improve security, attackers devise more sophisticated prompts or find entirely new angles of attack. This dynamic evolution will shape how LLMs are deployed in cybersecurity. It suggests that while LLMs hold tremendous promise (due to their ability to reason about context and adapt to new patterns), we must continuously update our defensive strategies to address their inherent risks (23-25).

COUNTERMEASURES AND DETECTION OF MALICIOUS LLM USE

Even as LLMs become integral to cybersecurity defense, we must also guard against malicious use of LLMs by adversaries. Attackers can use the same models for harmful purposes – for example, generating polymorphic malware code or automating phishing content as discussed. A growing area of research focuses on countermeasures to detect and attribute such AI-assisted malicious activities.

One promising countermeasure is the use of watermarks and provenance tracking in AI outputs. Earlier, we mentioned watermarking AI-generated code. More generally, researchers have proposed embedding hidden signals in content produced by LLMs so that it can later be identified. Recently, MCGMark was introduced as an online watermark specifically for code generated by LLMs. In tests, MCGMark was able to detect its

watermark in about 353 instances of AI-generated code with a success rate of approximately 97.8%, without significantly affecting the code's functionality (2). Such high reliability is encouraging: if every piece of code or text an LLM produces carries an invisible marker, then even if criminals use LLMs to produce malicious scripts, those could potentially be recognized after the fact by security tools scanning for the watermark. This could help attribute attacks (e.g., confirming that a strain of malware was machine-generated, which might indicate a larger automated campaign) (26, 27).

However, attribution, figuring out who (or which model/user) generated a malicious item, remains a hard problem. As one study pointed out, even if we know an LLM was used to create malware, identifying the specific malicious user who employed the LLM is still nearly impossible with current techniques. Attackers can operate anonymously or through stolen accounts, and the output doesn't inherently tie back to a specific person. This makes malicious code generation via LLMs a low-risk, high-reward activity for attackers: they can unleash AI-generated attacks at scale, and even if defenders recognize the content as AI-made, the attacker's identity remains obscured.

Major AI providers and the research community have not ignored these issues. OpenAI and others have proposed various solutions to detect AI-generated malicious content before it causes harm. For instance, zero-shot detectors are models that try to identify malicious or AI-generated content without needing extensive examples (they might use heuristics or general features of AI text). Another idea has been to fine-tune dedicated language model detectors – essentially training an AI to recognize outputs from another AI. Unfortunately, many of these solutions have proven either too weak or too brittle. As mentioned, OpenAI's own attempt at an AI-written text classifier was so inaccurate (only 26% accuracy, with a lot of false positives on human-written text) that it was withdrawn. In cybersecurity, some early "AI detectors" that try to spot AI-written code or messages have similarly struggled; attackers can slightly modify the AI output (e.g., by rephrasing or adding some human-written lines) to throw off the detector.

Where does this leave us? A key takeaway is that defensive efforts must be multi-pronged. There is likely no single tool that will robustly detect all malicious AI outputs. Instead, a combination of approaches like watermarks, anomaly detection, behavior profiling, and continuous human expert oversight will be needed. Organizations such as OpenAI are researching improved watermarking and detection algorithms, while others are focusing on monitoring how AI tools are used (for example, rate-limiting how quickly one account can generate code or content, to prevent someone from using a single API key to mass-produce malware).

Looking ahead, the community is also considering regulatory or ethical steps: perhaps requiring AI models to have provenance tracking or setting up agreements between AI providers to share information on abuse patterns. While technical solutions are still catching up, simply acknowledging the risk is an important first step. We now see conferences and security teams dedicated to LLM security; a field that encompasses both using LLMs for defense and protecting against malicious uses of LLMs.

In summary, the countermeasures for malicious LLM use are in their infancy. Techniques like MCGMark provide a glimpse of what's possible for tracing AI-generated attacks, and the failure of simple detectors has taught us that this is a complex challenge. The arms race will continue: as defenders innovate, attackers will adapt, and vice versa (26, 27).

CONCLUSION

LLMs may one day become a fundamental part of cybersecurity defenses. They offer capabilities – from understanding natural language threats to adapting quickly to new patterns – that traditional tools lack. However, at present there are still too many faults, insecurities, potential threats, and ethical implications for fully trusting LLMs with critical security tasks. Much of the current literature on LLMs in security is exploratory or theoretical, examining models that implement countermeasures to known issues like jailbreak prompting and code obfuscation. These studies highlight that while LLMs can be extremely useful, they also introduce new vulnerabilities.

Given the state of the technology today, placing the full burden of cybersecurity on still-maturing LLM technology is not advisable. In practical terms, organizations should use LLM-based tools in moderation and in conjunction with traditional methods. For instance, an LLM could be used to assist analysts by summarizing alerts, but final decisions might remain with humans or rule-based systems to avoid blind trust in AI. Likewise, deploying LLMs in contained environments (for example, monitoring specific subsystems rather than an entire enterprise network) can ensure that if the model makes a mistake, the impact is limited and observable. This

incremental approach allows us to gain the benefits of LLMs, such as improved threat detection and automated analysis, while mitigating the risk.

To mitigate the rise of misuse of LLMs (both by insiders and adversaries), preventive measures are essential. These include robust user access policies (to prevent unauthorized use of AI tools), monitoring for abnormal usage of AI systems, and perhaps most importantly, education and training. Security teams should be aware of what LLMs can and cannot do, and be prepared for scenarios where AI might err. They should also stay informed about the latest adversarial attack techniques on AI, since this knowledge will enable them to harden their defenses accordingly.

Future research should focus on maximizing the practical applications of LLMs while directly addressing the inherent risks that come with their deployment. This means not only improving LLM accuracy on cybersecurity tasks, but also enhancing their robustness against adversarial manipulation, reducing biases (so they don't overlook certain attacks or unfairly flag benign behavior (22)), and ensuring transparency in their decision-making. Techniques like better interpretability (to understand why an LLM flagged something as malicious) will increase trust and usability in operational settings.

By adopting a proactive approach and integrating LLMs carefully, continuously testing their limits, and iteratively improving both the models and the surrounding safeguards, the cybersecurity community can better prepare for a future where LLMs play a central role in safeguarding the digital environment. In essence, LLMs should be seen as powerful new allies that come with quirks; if we handle them wisely, aligning their strengths with our security needs and compensating for their weaknesses, they could significantly enhance our ability to defend against the ever-evolving landscape of cyber threats.

REFERENCES

1. Hofstetter M, Riedl R, Gees T, Koumpis A, Schaberreiter T. Applications of AI in Cybersecurity. Proceedings of the 2020 Second International Conference on Transdisciplinary AI (TransAI), Irvine, CA, USA, 2020; pp 138–141. DOI: 10.1109/TransAI49837.2020.00031. (accessed on 2025-07-29).
2. Sewak M, Emani V, Naresh A. CRUSH: Cybersecurity Research Using Universal LLMs and Semantic Hypernetworks. *CEUR Workshop Proceedings*. 2023; Vol. 3532. (accessed on 2025-07-29).
3. Uday Kiran A, Neha S, Manasa A, Anusha N. Artificial Intelligence for Cyber Security: A Systematic Mapping of Literature. *International Journal of Research in Engineering and Science*. 2022; 10 (6): 10061010. https://www.ijres.org/papers/Volume-10/Issue-6/100610061010.pdf. (accessed on 2025-07-29).
4. Caviglione L, Comito C, Coppolillo E, Gallo D, *et al*. Dawn of LLM4Cyber: Current Solutions, Challenges, and New Perspectives in Harnessing LLMs for Cybersecurity. *CEUR Workshop Proceedings*. 2024; 3762: 513–513. https://ceur-ws.org/Vol-3762/513.pdf. (accessed on 2025-07-29).
5. Cyber Triage Team. Intro to Import Hashing for DFIR. Cyber Triage Blog, 13 October 2017. https://www.cybertriage.com/blog/intro-to-import-hashing-for-dfir/. (accessed on 2025-07-29).
6. Mandiant Threat Intelligence Team. Tracking Malware with Import Hashing. Google Cloud Blog, 2020. https://cloud.google.com/blog/topics/threat-intelligence/tracking-malware-with-import-hashing. (accessed on 2025-07-29).
7. Mohammed Hassanin, Nour Moustafa. A Comprehensive Overview of Large Language Models (LLMs) for Cyber Defenses: Opportunities and Directions. arXiv 2024, arXiv:2405.14487. (accessed on 2025-07-29).
8. Jie Zhang, Haoyu Bu, Hui Wen, Yongji Liu, *et al*. When LLMs Meet Cybersecurity: A Systematic Literature Review. arXiv 2024, arXiv:2405.03644. (accessed on 2025-07-29). https://doi.org/10.1186/s42400-025-00361-w
9. Constantinos Patsakis, Fran Casino, Nikolaos Lykousas. Assessing LLMs in Malicious Code Deobfuscation of Real-world Malware Campaigns. arXiv 2024, arXiv:2404.19715. (accessed on 2025-07-29). https://doi.org/10.1016/j.eswa.2024.124912
10. Fangzhou Wu, Ning Zhang, Somesh Jha, Patrick McDaniel, Chaowei Xiao. A New Era in LLM Security: Exploring Security Concerns in Real-World LLM-based Systems. arXiv 2024, arXiv:2402.18649. (accessed on 2025-07-29).
11. Jean Marie Tshimula, D'Jeff K. Nkashama, Jean Tshibangu Muabila, René Manassé Galekwa, *et al*. Psychological Profiling in Cybersecurity: A Look at LLMs and Psycholinguistic Features. arXiv 2024, arXiv:2406.18783. (accessed on 2025-07-29). https://doi.org/10.1007/978-981-96-1483-7_31
12. Trad F, Chehab A. Prompt Engineering or Fine-Tuning? A Case Study on Phishing Detection with Large Language Models. *Machine Learning and Knowledge Extraction*. 2024; 6 (1): 367–384. https://ouci.dntb.gov.ua/en/works/lRnVmXGl/. (accessed on 2025-07-29). https://doi.org/10.3390/make6010018
13. Prompting Guide. Prompt Engineering Guide: Introduction. Promptingguide.ai. https://www.promptingguide.ai/introduction. (accessed on 2025-07-29).
14. Shi C, Yang H, Cai D, Zhang Z, *et al*. A Thorough Examination of Decoding Methods in the Era of

LLMs. arXiv 2024, arXiv:2402.06925. https://arxiv.org/abs/2402.06925. (accessed on 2025-07-29).
15. Zhiyuan Yu, Xiaogeng Liu, Shunning Liang, Zach Cameron, Chaowei Xiao, Ning Zhang. Don't Listen To Me: Understanding and Exploring Jailbreak Prompts of Large Language Models. arXiv 2024, arXiv:2404.11338. (accessed on 2025-07-29).
16. Do Anything Now: Characterizing and Evaluating In-the-Wild Jailbreak Prompts on Large Language Models. arXiv 2023, arXiv:2307.15043. https://arxiv.org/abs/2307.15043. (accessed on 2025-07-29).
17. Dettmers T, Pagnoni A, Holtzman A, Zettlemoyer L. QLoRA: Efficient Finetuning of Quantized LLMs. arXiv 2023, arXiv:2305.14314. https://arxiv.org/abs/2305.14314. (accessed on 2025-07-29).
18. Barbos A, Chebrolu A, Landen C, Shah S, *et al*. Advancing Cybersecurity: A Comprehensive Review of AI-Driven Detection Techniques. *Journal of Big Data*. 2024; 11: 1–24. https://journalofbigdata.springeropen.com/articles/10.1186/s40537-023-00833-7. (accessed on 2025-07-29). https://doi.org/10.1186/s40537-024-00957-y
19. Shang S, Zhao X, Yao Z, Yao Y, *et al*. Can LLMs Deeply Detect Complex Malicious Queries? A Framework for Jailbreaking via Obfuscating Intent. arXiv 2024, arXiv:2405.03654. https://arxiv.org/abs/2405.03654. (accessed on 2025-07-29).
20. Yu J, Lin X, Yu Z, Xing X. GPTFUZZER: Red Teaming Large Language Models with Auto-Generated Jailbreak Prompts. arXiv 2024, arXiv:2309.10253. https://arxiv.org/abs/2309.10253. (accessed on 2025-07-29).
21. Robey A, Wong E, Hassani H, Pappas GJ. SmoothLLM: Defending Large Language Models Against Jailbreaking Attacks. arXiv 2024, arXiv:2310.03684. https://arxiv.org/abs/2310.03684. (accessed on 2025-07-29).
22. Jared Moore. Are Large Language Models Consistent over Value-laden Questions?. arXiv 2024, arXiv:2212.01681. (accessed on 2025-07-29).
23. Bo Qiao, Liqun Li, Xu Zhang, Shilin He, *et al*. TaskWeaver: A Code-First Agent Framework. arXiv 2024, arXiv:2311.17541. (accessed on 2025-07-29).
24. Jingqing Ruan, Yihong Chen, Bin Zhang, Zhiwei Xu, *et al*. TPTU: Large Language Model-based AI Agents for Task Planning and Tool Usage. arXiv 2024, arXiv:2308.03427. (accessed on 2025-07-29).
25. Xinyue Shen, Zeyuan Chen, Michael Backes, Yun Shen, Yang Zhang. "Do Anything Now": Characterizing and Evaluating In-The-Wild Jailbreak Prompts on Large Language Models. arXiv 2024, arXiv:2308.03825. (accessed on 2025-07-29).
26. Barbos A, Chebrolu A, Landen C, Shah S, *et al*. Advancing Cybersecurity: A Comprehensive Review of AI-Driven Detection Techniques. *Journal of Big Data*. 2024; 11: 1–24. https://journalofbigdata.springeropen.com/articles/10.1186/s40537-023-00833-7. (accessed on 2025-07-29). https://doi.org/10.1186/s40537-024-00957-y
27. Ning K, Chen J, Zhong Q, Zhang T, *et al*. MCGMark: An Encodable and Robust Online Watermark for LLM-Generated Malicious Code. arXiv 2024, arXiv:2408.01354. https://arxiv.org/abs/2408.01354. (accessed on 2025-07-29).