Language Models in Cybersecurity: A Comparative Approach to Task-Driven Model Assessment

Holger Schmidt[1,†], Klaus Kaiser[1,*,†] and Daniel Spiekermann[1,†]

[1]Dortmund University of Applied Sciences and Arts, Germany

Abstract

Large language models (LLMs) have demonstrated impressive general-purpose capabilities across a wide range of computational tasks. However, their substantial resource demands and integration constraints raise critical concerns for deployment in security-sensitive scenarios. In response, *small language models (SLMs)* and *tiny language models (TLMs)* have gained attention as lightweight, adaptable alternatives, especially when operational context and task requirements are well understood. This paper provides a *task-driven approach to language model (LM) assessment*, emphasizing that the largest LM is not always the optimal choice. We perform a systematic analysis of representative tasks from *cybersecurity*, i.e., especially in the fields of *secure software development* and *digital forensics*, and extract key technical and operational characteristics. By mapping these characteristics profiles to properties of different LM classes, we identify *practical scenarios* where SLMs or TLMs are not only sufficient but preferable.

Keywords

Language Model, Cybersecurity, Secure Software Development Lifecycle, Model Assessment

1. Introduction

The rapid proliferation of artificial intelligence and natural language processing has revolutionized software and system development, with *large language models (LLMs)* becoming essential tools for various tasks such as logical reasoning, summarization, code generation, and automated decision support [1]. Although LLMs are increasingly used in *cybersecurity* for secure software development [2] and digital forensics [3], supporting tasks such as secure coding and reasoning over forensic artifacts, their strong generalization capabilities and performance in diverse scenarios make them inherently more aligned with the role of *generalists*.

However, this general-purpose strength results in different challenges. LLMs are resource-intensive, with substantial demands on memory, compute, and energy consumption, creating significant limitations to integration in constrained environments such as edge devices, embedded systems, or security-critical infrastructures. Additionally, concerns about data leakage, lack of transparency, update latency, and limited control raise critical questions regarding their suitability for regulated or mission-critical domains [4].

These challenges led to the focus on *small language models (SLMs)* [5] or even *tiny language models (TLMs)* [6]. While TLMs and SLMs cannot match the general-purpose performance of LLMs, they often offer improved characteristics in terms of efficiency and deployment flexibility. Technically, SLMs achieve their efficiency through reduced layer depth, narrower hidden dimensions, and careful optimization of attention mechanisms. Although this reduced capacity limits their generalization range, SLMs and TLMs can be easier fine-tuned on domain-specific tasks - sometimes even outperforming LLMs [7]. Furthermore, the lower complexity of SLMs facilitates better runtime control, faster adaptation cycles, and cost reduction [8].

SPAIML'25: International Workshop on Security and Privacy-Preserving AI/ML, October 26, 2025, Bologna, Italy

*Corresponding author.

†These authors contributed equally.

klaus.kaiser@fh-dortmund.de (K. Kaiser)

0000-0002-9370-0086 (H. Schmidt); 0009-0005-9793-2975 (K. Kaiser); 0000-0003-4762-6062 (D. Spiekermann)

© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

published 2026-01-28

CEUR-WS.org/Vol-4154/paper5.pdf

The main contributions of this paper are:

- An analysis based on recent research and practices examining *characteristics* and *potential support of LMs* for selected *cybersecurity tasks*. The latter cover all phases of a state-of-the-art *secure software development lifecycle (SSDL)*, including in particular secure software development and digital forensics.
- A *task-driven assessment* and *technical comparison* of LLMs, SLMs, and TLMs based on their *suitability* for addressing the analyzed cybersecurity tasks' characteristics, with a specific focus on *resource-aware* and *practical deployment*.

The remainder is structured as follows. Section 2 lists relevant work and research in the field of LMs and related topics. Section 3 outlines seven representative cybersecurity tasks, envisages potential LM support, and presents tasks' characteristics. Section 4 defines critical properties related to LMs and evaluates how well the cybersecurity tasks' characteristics are satisfied by different LMs. Section 5 maps the tasks to LMs by comparing the tasks' characteristics with the LM properties to identify the most suitable LM. We conclude this paper in Section 6 and give an outlook to our future research.

2. Related Work

In the context of cybersecurity and SSDL, LM research primarily focuses on code-centric tasks such as secure code generation and vulnerability detection but also on digital forensics. Accordingly, we discuss a selection of corresponding works. In the field of secure software development, using LLMs for secure code generation is explored [9, 10], a systematic literature review on code security is conducted [2], and SecureBERT [11], an example of a domain-specific LM, is developed. Moreover, LLMs supporting automated vulnerability detection [12, 13] are developed. In the field of digital forensics, the application of the ChatGPT LLM is explored [14, 15], a systematic literature review is conducted [3], and ForensicLLM [16], an example of a domain-specific LLM, is developed. In contrast to the aforementioned works, our approach is broader in that we not only consider code-centric cybersecurity tasks, but also other tasks such as threat modeling and analysis. As explained above, although there are domain-specific LMs in the field of cybersecurity, most recent research focuses on LLMs.

As far as we know, the task-driven LM assessment approach presented in this paper is currently the only one of its kind. Nevertheless, we consider benchmarks that evaluate LMs in the context of cybersecurity as related work. In [17], LLMs are evaluated regarding their ability to refute security and privacy misconceptions. SALLM [18], LLMSecEval [19] and *CyberSecEval*[1] are example benchmarks for LLMs to evaluate secure coding abilities. Similarly to the general LM works in cybersecurity discussed above and in contrast to our work, the benchmark works focus on LLMs and code-centric tasks.

3. Cybersecurity Tasks and their Characteristics

The suitability of an LM for a given scenario cannot be assessed solely based on its raw capabilities. Instead, a thorough examination of multiple critical requirements that reflect functional, technical, and operational constraints specific to the intended application context is necessary. As we focus on cybersecurity, especially on the topics of secure software development and digital forensics, we analyze in Section 3.1 representative cybersecurity tasks to establish a foundation for assessing the suitability of LMs in Section 3.2.

3.1. Cybersecurity Tasks

There are several *best practices* in the area of secure software development such as *Microsoft Security Development Lifecycle (SDL)*[2], *OWASP Software Assurance Maturity Model (SAMM)*[3], *NIST Secure Software*

[1]https://meta-llama.github.io/PurpleLlama/CyberSecEval
[2]https://www.microsoft.com/en-us/securityengineering/sdl
[3]https://owaspsamm.org

Development Framework (SSDF)[4], and also *standards* such as *IEC 62443-4-1*[5]. Based on an analysis and comparison of the aforementioned best practices and standards using *SAMMY*[6], we derive a base set of *business functions* for a state-of-the-art SSDL. For each business function, common *security practices* as advocated by the best practices and standards are selected. These security practices are implemented by specific *security tasks*. For instance, *anomaly detection* is a task typically performed as part of the security practice of *incident detection and response*, which is integral to the business function of *operations*. In the following, we present a selection of tasks that are *representative* in the sense that they cover both, early (requirements, design) and late phases (implementation, verification, operation) of the SSDL. Moreover, this selection includes tasks that are currently supported by LMs (as outlined in Section 2) as well as those lacking such support. Each task is described in a concise, profile-like format and is accompanied by a discussion of potential support through LMs as well as a presentation of the task's requirements (potentially constraining LM suitability).

3.1.1. Threat Modeling & Analysis

Threat modeling and analysis [20] allows to identify weaknesses at different early stages of SSDL, prior to their integration into the system through implementation or deployment. Threat modeling and analysis typically follows a systematic methodology such as STRIDE [21] and PASTA [22]. The methods include system modeling techniques (e.g., based on data flow diagrams) to establish the modeling context and creativity approaches (e.g., brainstorming, serious gaming) to explore system models with the aim of discovering weaknesses.

Potential LM support: LMs can support threat modeling and analysis by automatically generating system models based on, e.g., given artifacts or observations of a human developer. Moreover, LMs can act as companions to human developers, guiding creativity approaches to identify weaknesses in systems.

Requirements: Input artifacts and outputs, e.g., system and threat models, can contain internal information to be kept confidential and personal data. Since threat models are an essential foundation for software and system development, they need to be particularly reliable. Threat models need to be updated from time to time, e.g., whenever there are system changes.

3.1.2. Secure Coding

Secure coding is the practice of writing source code with the intention of minimizing or eliminating vulnerabilities. This task is typically driven by basic tenets such as Saltzer and Schroeder's design principles [23] as well as the use of secure coding guidelines, such as Oracle's guidelines for Java[7] and checklists as provided by OWASP cheat sheet series[8].

Potential LM support: LMs can enhance secure coding practices by automatically generating code snippets based on provided specifications or by refining code according to a developer's input. Additionally, LMs can serve as assistants to developers, providing guidance on best practices and identifying potential vulnerabilities in code.

Requirements: For use while programming, source code has to be generated immediately. Source code is specific to programming languages, frameworks, and libraries. It is often seen as intellectual property and can also contain personal data. Since source code plays a central role in software and

[4]https://csrc.nist.gov/Projects/ssdf
[5]https://webstore.iec.ch/en/publication/33615
[6]https://sammy.codific.com
[7]https://www.oracle.com/technetwork/java/seccodeguide-139067.html
[8]https://cheatsheetseries.owasp.org

system development, it must be particularly reliable. In particular, latest secure coding guidelines must be followed, and known vulnerabilities should be avoided. The secure coding task must be continuously applied throughout the entire SSDL.

3.1.3. Secure Code Review

Modern code review (MCR) [24] is a developer-centric process that aims to identify *source code defects*. MCR consists of two phases: planning and setup, and the actual review. The author prepares a review package and notifies the chosen reviewers. These analyze the code, interact with the author, and decide on acceptance, rejection, or rework. MCR is integrated with systems like Git[9] and supported by tools such as Gerrit[10], focusing on quick, small code changes. Regarding cybersecurity, MCR is crucial for vulnerability detection [25], using secure coding guidelines and checklists, as seen in OWASP's code review guide[11] which refers to this process as *secure code review.*

Possible LM support: LMs can support secure code review, either impersonating the reviewer role completely, or serving as a companion to a human reviewer. LMs can support code checking, on the one hand, by automatically detecting potential weaknesses, the maliciousness of which must then be manually confirmed or refuted, and, on the other hand by methodically accompanying a reviewer during manual code checking.

Requirements: Essentially, the requirements are similar to those of the previously discussed task of secure coding, with the difference that secure code reviews are not conducted continuously, but only when review packages must be processed.

3.1.4. Static Application Security Testing

Static application security testing (SAST) [26] involves analyzing application source code for vulnerabilities without executing the code. SAST, as a white-box approach, integrates with secure coding and secure code review practices, and is typically supported by tools that involve a comprehensive examination of the internal code structure. These tools automate the detection of vulnerabilities by applying a set of predefined rules and patterns, which are continuously updated to reflect the latest security threats.

Potential LM support: LMs might overcome traditional rule-based SAST approaches and allow detecting complex source code anomalies. LMs can offer natural language explanations and recommendations for remediation, aiding developers in understanding and fixing identified vulnerabilities.

Requirements: Since we focus here on those SAST tools that require compilable code, we consider the use of SAST rather during secure code review than during secure coding. However, the requirements are similar to those of the previously discussed task of secure coding except the following: Usage does not require any specific timing constraints and occurs only intermittently. Because of the compilable code, input size is rather large. SAST results in terms of identified vulnerabilities have to be kept confidential until security measures are in place.

3.1.5. Penetration Testing

Penetration testing [27] comprises simulating real-world attacks on an application, network, or system to identify and exploit vulnerabilities in a controlled environment. Typically mimicking the perspective and techniques of potential attackers, penetration testing makes use of several different tools, including,

[9]https://git-scm.com
[10]https://www.gerritcodereview.com
[11]https://owasp.org/www-project-code-review-guide

for example, SAST tools. Penetration testing follows a methodology, e.g., *Penetration Testing Execution Standard (PTES)*[12] and is generally employed in late SSDL phases such as verification and operation.

Potential LM support: LMs can assist human penetration testers by generating realistic attack vectors, developing exploits and providing insights on complex security environments.

Requirements: In penetration testing, there is typically no particular time pressure. The artifacts processed may include both personal data and internal information. Penetration testing reports have to be kept confidential until risks have been reduced. Similar to secure coding, the latest known best practices, standards, and specific vulnerabilities must be taken into account. Penetration testing is one of the last lines of defense, so reliability is important. Similar to anomaly detection, the input data is typically extensive.

3.1.6. Incident Response

Incident response refers to the structured process of detecting, analyzing, mitigating, and recovering from security incidents such as intrusions, data breaches, or malware outbreaks [28]. It is a time-critical and information-intensive task that often requires automated support for log analysis, correlation of indicators, threat classification, and response decision-making [29]. The goal is to minimize impact, contain threats, and ensure rapid recovery while maintaining auditability and compliance.

Potential LM support: LMs can improve the incident response process by facilitating the automated analysis of large data-sets, quickly identifying patterns and anomalies. They can assist in threat classification by evaluating and interpreting indicators in real-time, leading to a faster understanding of incidents. Additionally, LMs can provide decision support by deriving insights from historical data and suggesting targeted response actions to optimize threat containment and recovery efforts.

Requirements: Incident response tasks are performed if suspicious behaviour in IT infrastructures emerge. In this case, the investigation of personal or company-internal information comes into play, leading to privacy-related restrictions. Depending on the criticality of the incident, the investigation process has to be accelerated in order to retrieve results about the attack timely.

3.1.7. Anomaly Detection

Anomaly detection involves identifying deviations from expected patterns in data, which may indicate misconfigurations, intrusions, system failures, or novel attack behavior [30]. It is commonly used in network monitoring [31], system behavior analysis, and threat hunting. Depending on the deployment context, it can run continuously (e.g., in real-time monitoring) or batch-wise (e.g., in offline forensics).

Potential LM support: LMs can significantly aid anomaly detection by analyzing vast streams of data to identify deviations from established patterns in real-time. By learning from historical data, they can more accurately differentiate between normal fluctuations and genuine anomalies, reducing false positives. Additionally, LMs can provide contextual insights into detected anomalies, helping to prioritize response efforts and refine detection algorithms over time.

Requirements: The process of anomaly detection is typically faced with large-scale data sources like network traffic, system logs, financial information and health monitoring data. Most of these information are privacy-related, volatile and dynamic. Depending on this, LMs must operate under strict latency constraints, restrict external access and support continuous update cycles to ensure the ongoing validity of the extracted results.

[12] http://www.pentest-standard.org

Table 1
Characteristics of representative cybersecurity tasks. The characteristics are derived in Section 3.2. Whether a tasks requires characteristic or not is derived from Section 3.1.

	Cybersecurity tasks	*Real-time usage*	*Intellectual property*	*Personal data*	*Up-to-dateness*	*Tech constraints*	*Trustworthi-ness*	*Continuous usage frequency*	*Large input size*
Requirements & Design	*Threat modeling & analysis*	✘	✓	✓	✘	✘	✓	✘	✘
Implementation	*Secure coding*	✓	✓	✓	✓	✓	✓	✓	✘
	Secure code review	✓	✓	✓	✓	✓	✓	✘	✘
	Static application security testing	✘	✓	✓	✓	✓	✓	✘	✓
Verification	*Penetration testing*	✘	✓	✓	✓	✓	✓	✘	✓
Operation	*Incident response*	✘	✘	✓	✓	✓	✘	✘	✓
	Anomaly detection	✓	✘	✓	✓	✓	✘	✓	✓

3.2. Characteristics

Since the requirements of cybersecurity tasks as presented in the previous Section 3.1 overlap, we have identified and compiled these common requirements and defined corresponding *characteristics*. E.g., the requirement that personal data is typically processed applies to all tasks, as outlined in the requirements paragraphs of the task descriptions in the previous section. Therefore, we have defined *personal data* as a characteristic. Below, all characteristics defined following this approach are listed and explained.

Real-time usage: Defines the necessity for immediate responses, demanding delivery with minimal latency, often under strict timing constraints.

Intellectual property: For a task involving proprietary knowledge, source code, or domain-specific algorithms, the protection of intellectual property must be ensured.

Personal data: For a task that entails user-specific or personally identifiable information, which are governed by legal and ethical constraints, the protection of personal data is essential.

Up-to-dateness: Emphasizes that a task requires access to the latest information, such as threat intelligence, dynamic content, or time-sensitive rules, highlighting the importance of incorporating or adapting to current data.

Tech constraints: Describes task dependencies on infrastructure, programming languages, frameworks, or libraries, where strong vendor lock-in or limited deployment portability can hinder adoption in certain environments.

Trustworthiness: Highlights the trust needed for a task regarding the robustness, predictability, and verifiability of its outputs, crucial in contexts where systems must handle edge conditions, resist adversarial inputs, and maintain consistency.

Continuous usage frequency: Describes if a task is typically performed continuously, i.e., very often.

Large input size: Describes if a task requires a large amount of input data in prompt.

In Table 1, we align the characteristics with the tasks. This way, *characteristics profiles* for the selected tasks emerge, forming constraints that are particularly critical to the assessment of LM suitability. In Table 1, a hook (✓) means the characteristic in the corresponding column is relevant for the task in the corresponding row, while cross (✗) means the characteristic is not relevant for the task. Due to the way the characteristics are defined - directly derived from the task requirements - the justifications for marking crosses or hooks become immediately clear.

In the next sections, we discuss to what extent different LMs can address the characteristics and ultimately which LMs are suitable for which task and which are not.

Table 2
Performance of different Qwen1.5 models. *Class:* class of LM, *B*: size of the model in billions, *Score:* average score on the Open LLM Leaderboard [34], *GB:* memory utilized on the GPU during test, *s:* time needed to perform the complete test. Results are obtained from [32, 33].

Model	Class	B	Score (%)	GB	s
Qwen1.5-1.8B	TLM	1.8	9.12	3.83	2.62
Qwen1.5-7B	SLM	7	15.22	11.28	5.94
Qwen1.5-14B	SLM	14	20.22	12.77	12.86
Qwen1.5-110B	LLM	111	29.56	65.20	102.23

4. Task-driven LM Assessment

We present definitions for LLM, SLM and TLM in Section 4.1 and we consider in Section 4.2 LM properties such as *size*, *deployment* and *usage* to derive LM profiles in Section 4.3.

4.1. Definitions

As stated before, we want to consider not only LLMs but also SLMs and TLMs. Unfortunately, there is no clear definition for these classes. For our case, we differentiate LMs based on the hardware requirements, which are needed to operate them. An identifier for this is the number of parameters - which defines the amount of VRAM (graphics card memory) and operations needed for inference. Therefore we define:

LLMs: Models being so large, that specialized hardware is needed (like NVIDIA H100 graphic cards).

SLMs: Models being smaller than LLMs such that they can be run on consumer hardware (like NVIDIA RTX 4070 graphics card).

TLMs: Models being so small, that no dedicated hardware is needed (without dedicated graphics card).

To get a glimpse on the differences of the model sizes, we take a look on the Qwen1.5[13] model, which was developed by the Alibaba group and is provided in different sizes ranging from 0.5B up to 110B[14] parameters. In Table 2, the performance of some of these models with their corresponding classification are given [32, 33].

4.2. Differentiation of LMs

Many LLMs are only available in *cloud*, by an external provider. If the model is publicly available, it can also be deployed locally on machines with a suitable *GPU*. Alternatively, they can also be deployed on a machine only equipped with a *CPU*, resulting in slower inference time making LLMs unusable and SLMs hardly usable. We differentiate LMs concerning deployment, which directly affects data protection (cloud vs. local) and speed (cloud vs. GPU vs. CPU).

Next to the size of the models and the deployment, it is useful to differ how an LM is *used* to solve the task. In general an LM creates a prediction based on a given prompt. The simplest way is to directly use the resulting prediction in the following process. In this case, one can differ between the usage of a *pre-trained* model, i.e., an LM without further improvement, or a *fine-tuned* model, i.e., a model which is tuned on a specific task with an additional training phase. A different way is *retrieval-augmented generation (RAG)*: The model only serves as an interface between task and solution database. The model reformulates the task into a query, which can then be evaluated externally. Afterwards, the result is given back to the model to generate the final answer [35]. A pre-trained model can be used directly, without further improvements, a fine-tuned model can be more accurate in solving tasks and in RAG new information can be directly added.

[13]https://qwenlm.github.io/blog/qwen1.5
[14]https://qwenlm.github.io/blog/qwen1.5-110b

Table 3
LM profiles based on LM properties. Whether an LM fulfills a property or not is derived in Section 4.3.

		Real-time usage	*Intellectual property*	*Personal data*	*Up-to-dateness*			*Tech constraints*			*Trustworthiness*			*Continuous usage frequency*	*Large input size*
Class	**Hardware**				*pre-trained*	*fine-tuned*	*RAG*	*pre-trained*	*fine-tuned*	*RAG*	*pre-trained*	*fine-tuned*	*RAG*		
LLM	*Cloud*	✗	✗	✗	✗	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓
	GPU	✗	✓	✓	✗	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓
SLM	*Cloud*	✗	✗	✗	✗	✓	✓	✗	✓	✓	✗	✗	✗	✗	✓
	GPU	✗	✓	✓	✗	✓	✓	✗	✓	✓	✗	✗	✗	✓	✓
	CPU	✗	✓	✓	✗	✓	✓	✗	✓	✓	✗	✗	✗	✗	✓
TLM	*GPU*	✓	✓	✓	✗	✓	✓	✗	✓	✓	✗	✗	✗	✓	✗
	CPU	✗	✓	✓	✗	✓	✓	✗	✓	✓	✗	✗	✗	✗	✗

4.3. LM Assessment based on Characteristics

In Section 3, we derived characteristics relevant for different cybersecurity tasks. Next, we check to what extent the different classes of LMs fulfill these characteristics. For this, we consider the following groups of LMs:

- *LLMs* deployed in *cloud* or *GPU* and used *pre-trained*, *fine-tuned* or in *RAG*
- *SLMs* deployed in *cloud*, *GPU* or *CPU* and used *pre-trained*, *fine-tuned* or in *RAG*
- *TLMs* deployed in *GPU* or *CPU* and used *pre-trained*, *fine-tuned* or in *RAG*

We summarize our results in Table 3, where a hook (✓) means the given LM class is suitable to fulfill this characteristic, while cross (✗) means that the given LM class is less preferable for this characteristic.

Note that for most characteristics, there is no strict separation whether an LM fulfills it or not. Thus, we give an impression based on our experience from current practice and the state of the art. Depending on a specific task and concrete LM, the assessment could be completely different.

It is important to note that we do not compare LMs in terms of accuracy for a cybersecurity task, as such comparison are intended to be conducted only after selecting a concrete model.

Real-time usage: Generally, a large model is slower than a small model given the same hardware. For real-time usage, an LLM in the cloud can only be used in a limited way - since delay due to the data transfer is added. TLMs can be used on proper local hardware in a quite fast way. Therefore, only GPU-deployed TLM is marked with a hook, while all others are marked with a cross.

Intellectual property and personal data: To ensure the protection of intellectual property and personal data, the model should ideally be deployed locally, e.g., within the application's domain. Therefore, cloud-deployed LMs are marked with a cross, while all others are marked with a hook.

Up-to-dateness: In RAG, new data can always be added and are directly available. Pre-trained and fine-tuned models can only retrieve data given at training stage - or when these are submitted in the prompt (limited by the context length). Therefore, a pre-trained model is limited to its first training, while the fine-tuning process can be restarted with new data containing the missing information. Therefore, pre-trained LMs are marked with a cross, while all others are marked with a hook.

Tech constraints: LLMs are generalists. Thus, they can access a wealth of knowledge of various technologies by design. SLMs and TLMs are smaller, therefore limited in their knowledge. With fine-tuning and RAG all LM classes can be enhanced in their technology knowledge. Therefore, pre-trained SLMs and TLMs are marked with a cross, while all others are marked with a hook.

Trustworthiness: LMs can achieve a good performance on different tasks, but they remain neural networks and thus statistical models - suffering from false predictions or hallucinations [36]. Even more, neural networks are seen as black boxes, since it is not clear how the model gets to its result. If trustworthiness is needed, the LM should only be used in a setting where the output can be controlled. Therefore, we mark LLMs, SLMs and TLMs with a cross.

Continuous usage frequency: The usage frequency directly affects the costs of the usage of a model, while an LM is in general slow and expensive to operate, this model might not be suitable in a setting where a continuous frequency is needed. On the other hand, smaller models can be used more frequently, due to their reduced hardware consumption. Therefore, only GPU-deployed SLM and TLM are marked with a hook, while all others are marked with a cross.

Large input size: In general: all classes of LMs could be built to handle large input sizes. On the other hand, a large input directly affects the needed resources and inference time. Consequently, since TLMs are used in low resource and fast inference settings, they are less suitable for large input sizes. Therefore, TLMs are marked with a cross, while all others are marked with a hook.

5. Comparison of Characteristics and LM Profiles

We now compare the characteristics profile of each cybersecurity task with the profile of each LM. We compute a score S ranging from 0 to 8, where 0 equals to "the model is not suitable for this task" and 8 equals to "the model is suitable for this task" using the formula

$$S(LM) = \sum_{i=1}^{8} f(c_i, LM).$$

Here, c_i represents one out of eight task characteristic as presented in Section 3.2 and LM represents an LM profile based on LM class, hardware and usage (if applicable) as presented in Section 4.3. We calculate f as follows:

$$f(c_i, LM) = \begin{cases} 0 & \text{if } c_i \text{ required and } c_i \text{ not fulfilled by } LM \\ 1 & \text{otherwise} \end{cases}$$

E.g., since the characteristic personal data is required by the task secure coding, we calculate $f(Personal\ data, (SLM, Cloud)) = 0$ because according to Table 3 the characteristic personal data is not supported by a cloud-deployed SLM. Following this approach, the resulting scores for conformity of LMs and tasks are summarized in Table 4.

Note that these results give only an indication on which combination might be promising and which not. Since the characteristics of LMs and tasks are generalized, a different result might be obtained if a concrete LM is investigated and compared with respect to a specific task. Note furthermore, that there might be some characteristics which are show-stoppers, e.g., if data protection is an unavoidable restriction, an LM that does not fulfill this property must be excluded directly.

From Table 4, we can obtain some key findings:

- TLMs on GPU and fine-tuned or RAG seem to be promising for all tasks.
- Cloud models are less favorable due to their restrictions on data protection and real-time usage.
- For some cases, the choice of models seems to be less important, i.e., threat modeling & analysis and incident response.

Overall, depending on the task, we can observe that SLMs and TLMs are promising approaches if they reach the required accuracy.

Table 4
Comparison of LM profiles and characteristics profiles of cybersecurity tasks. A large value (up to 8) results into a good match, while a small value (down to 0) corresponds to a less good match.

			Threat modeling & analysis	Secure coding	Secure code review	Static application security testing	Penetration testing	Incident response	Anomaly detection
LLM	Cloud	pre-trained	5	2	3	4	4	6	4
		fine-tuned	5	3	4	5	5	7	5
		RAG	5	3	4	5	5	7	5
	GPU	pre-trained	7	4	5	6	6	7	5
		fine-tuned	7	5	6	7	7	8	6
		RAG	7	5	6	7	7	8	6
SLM	Cloud	pre-trained	5	1	2	3	3	5	3
		fine-tuned	5	3	4	5	5	7	5
		RAG	5	3	4	5	5	7	5
	GPU	pre-trained	7	4	4	5	5	6	5
		fine-tuned	7	6	6	7	7	8	7
		RAG	7	6	6	7	7	8	7
	CPU	pre-trained	7	3	4	5	5	6	4
		fine-tuned	7	5	6	7	7	8	6
		RAG	7	5	6	7	7	8	6
TLM	GPU	pre-trained	7	5	5	4	4	5	5
		fine-tuned	7	7	7	6	6	7	7
		RAG	7	7	7	6	6	7	7
	CPU	pre-trained	7	3	4	4	4	5	3
		fine-tuned	7	5	6	6	6	7	5
		RAG	7	5	6	6	6	7	5

6. Conclusions and Future Work

This paper highlights that larger LMs are not always the optimal choice. Through a task-driven assessment approach, we demonstrate how SLMs and TLMs can provide adequate solutions for certain practices in the fields of secure software development and digital forensics. By analyzing representative cybersecurity tasks, we identify task characteristics and check their fulfillment by LMs. Our technical comparison shows where smaller LMs are more preferable than larger ones, offering resource-aware deployment without sacrificing performance.

This work lays the foundation for strategically developing LMs for cybersecurity tasks. In the future, we plan to implement and study especially SLMs and TLMs for distinct cybersecurity tasks selected based on our approach. Moreover, we intend to explore additional LM properties and expand the range of cybersecurity tasks to provide a more comprehensive understanding of the applicability of LMs. Furthermore, we plan to investigate specific LMs in more detail and evaluate their characteristics in the context of selected tasks - including, in particular, an assessment of their accuracy for each task.

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] H. Naveed, A. U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Akhtar, N. Barnes, A. Mian, A comprehensive overview of large language models (2024). doi:10.48550/arxiv.2307.06435. arXiv:2307.06435.

[2] E. Basic, A. Giaretta, Large language models and code security: A systematic literature review (2025). doi:10.48550/arxiv.2412.15004. arXiv:2412.15004.

[3] A. Wickramasekara, F. Breitinger, M. Scanlon, Exploring the potential of large language models for improving digital forensic investigation efficiency, Forensic Science International: Digital Investigation 52 (2025) 301859. doi:10.1016/j.fsidi.2024.301859.

[4] B. C. Das, M. H. Amini, Y. Wu, Security and privacy challenges of large language models: A survey, ACM Computing Surveys 57 (2025) 1–39.

[5] T. Schick, H. Schütze, It's not just size that matters: Small language models are also few-shot learners, in: K. Toutanova, A. Rumshisky, L. Zettlemoyer, D. Hakkani-Tur, I. Beltagy, S. Bethard, R. Cotterell, T. Chakraborty, Y. Zhou (Eds.), Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Association for Computational Linguistics, Online, 2021, pp. 2339–2352. doi:10.18653/v1/2021.naacl-main.185.

[6] I. Lamaakal, Y. Maleh, K. El Makkaoui, I. Ouahbi, P. Pławiak, O. Alfarraj, M. Almousa, A. A. Abd El-Latif, Tiny language models for automation and control: Overview, potential applications, and future research directions, Sensors 25 (2025). doi:10.3390/s25051318.

[7] F. Wang, Z. Zhang, X. Zhang, Z. Wu, T. Mo, Q. Lu, W. Wang, R. Li, J. Xu, X. Tang, et al., A comprehensive survey of small language models in the era of large language models: Techniques, enhancements, applications, collaboration with LLMs, and trustworthiness (2024). doi:10.48550/arXiv.2411.03350.

[8] C. Irugalbandara, A. Mahendra, R. Daynauth, T. K. Arachchige, J. Dantanarayana, K. Flautner, L. Tang, Y. Kang, J. Mars, Scaling down to scale up: A cost-benefit analysis of replacing OpenAI's LLM with open source SLMs in production, in: Proceedings of the International Symposium on Performance Analysis of Systems and Software, ISPASS, IEEE, 2024, pp. 280–291.

[9] J. Wang, L. Cao, X. Luo, Z. Zhou, J. Xie, A. Jatowt, Y. Cai, Enhancing large language models for secure code generation: A dataset-driven study on vulnerability mitigation (2023). doi:10.48550/arxiv.2310.16263. arXiv:2310.16263.

[10] J. Li, A. Sangalay, C. Cheng, Y. Tian, J. Yang, Fine tuning large language model for secure code generation, in: Proceedings of the First International Conference on AI Foundation Models and Software Engineering, FORGE, ACM, New York, NY, USA, 2024, pp. 86–90. doi:10.1145/3650105.3652299.

[11] E. Aghaei, X. Niu, W. Shadid, E. Al-Shaer, SecureBERT: A domain-specific language model for cybersecurity, in: Proceedings of the International Conference on Security and Privacy in Communication Systems, SecureComm, Springer, 2023, pp. 39–56.

[12] S. Z. Ridoy, M. S. H. Shaon, A. Cuzzocrea, M. S. Akter, Enstack: An ensemble stacking framework of large language models for enhanced vulnerability detection in source code, in: International Conference on Big Data, BigData, IEEE, 2024, pp. 6356–6364.

[13] M. F. Sultan, T. Karim, M. S. H. Shaon, M. Wardat, M. S. Akter, Enhanced LLM-based framework for predicting null pointer dereference in source code (2024). doi:10.48550/arxiv.2412.00216. arXiv:2412.00216.

[14] M. Scanlon, F. Breitinger, C. Hargreaves, J.-N. Hilgert, J. Sheppard, ChatGPT for digital forensic investigation: The good, the bad, and the unknown, Forensic Science International: Digital Investigation 46 (2023) 301609. doi:10.1016/j.fsidi.2023.301609.

[15] H. Henseler, H. van Beek, ChatGPT as a copilot for investigating digital evidence, in: Proceedings of the Third International Workshop on Artificial Intelligence and Intelligent Assistance for Legal Professionals in the Digital Workplace (LegalAIIA 2023), volume 3423 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023, pp. 58–69.

[16] B. Sharma, J. Ghawaly, K. McCleary, A. M. Webb, I. Baggili, ForensicLLM: A local large language model for digital forensics, in: Proceedings of the Digital Forensics Research Conference Europe, DFRWS EU, 2025.

[17] Y. Chen, A. Arunasalam, Z. B. Celik, Can large language models provide security & privacy advice? measuring the ability of llms to refute misconceptions, in: Proceedings of the 39th Annual

Computer Security Applications Conference, ACSAC, ACM, New York, NY, USA, 2023, pp. 366–378. doi:10.1145/3627106.3627196.

[18] M. L. Siddiq, J. C. S. Santos, S. Devareddy, A. Muller, SALLM: Security assessment of generated code, in: Proceedings of the 39th International Conference on Automated Software Engineering Workshops, ASEW, Sacramento, CA, USA, 2024. doi:10.1145/3691621.3694934.

[19] C. Tony, M. Mutas, N. E. D. Ferreyra, R. Scandariato, LLMSecEval: A dataset of natural language prompts for security evaluations (2023). doi:10.48550/arxiv.2303.09384. arXiv:2303.09384.

[20] I. Tarandach, M. J. Coles, Threat Modeling - A Practical Guide for Development Teams, O'Reilly, 2021.

[21] A. Shostack, Threat Modeling: Designing for Security, Wiley, 2014.

[22] T. UcedaVélez, M. Morana, Risk Centric Threat Modeling: Process for Attack Simulation and Threat Analysis, Wiley, 2015.

[23] J. H. Saltzer, M. D. Schroeder, The protection of information in computer systems, Proceedings of the IEEE 63 (1975) 1278–1308.

[24] A. Bacchelli, C. Bird, Expectations, outcomes, and challenges of modern code review, in: Proceedings of the 35th International Conference on Software Engineering, ICSE, 2013, pp. 712–721. doi:10.1109/ICSE.2013.6606617.

[25] L. Braz, C. Aeberhard, G. Çalikli, A. Bacchelli, Less is more: supporting developers in vulnerability detection during code review, in: Proceedings of the 44th International Conference on Software Engineering, ICSE, ACM, New York, NY, USA, 2022, pp. 1317–1329. doi:10.1145/3510003.3511560.

[26] B. Chess, J. West, Secure Programming with Static Analysis, O'Reilly, 2007.

[27] M. Hickey, J. Arcuri, Hands on Hacking: Become an Expert at Next Gen Penetration Testing and Purple Teaming, Wiley, 2020.

[28] D. Schlette, M. Caselli, G. Pernul, A comparative study on cyber threat intelligence: The security incident response perspective, IEEE Communications Surveys & Tutorials 23 (2021) 2525–2556.

[29] B. Schneier, The future of incident response, IEEE Security & Privacy 12 (2014) 96–96. doi:10.1109/MSP.2014.102.

[30] M. Thottan, C. Ji, Anomaly detection in IP networks, IEEE Transactions on Signal Processing 51 (2003) 2191–2204.

[31] D. Spiekermann, Positional packet capture for anomaly detection in multitenant virtual networks, International Journal of Network Management 35 (2025) e2326.

[32] R. P. Ilyas Moutawwakil, Optimum-benchmark: A framework for benchmarking the performance of transformers models with different hardwares, backends and optimizations., ????

[33] R. P. Ilyas Moutawwakil, Llm-perf leaderboard, https://huggingface.co/spaces/optimum/llm-perf-leaderboard, 2023.

[34] C. Fourrier, N. Habib, A. Lozovskaya, K. Szafer, T. Wolf, Open llm leaderboard v2, https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard, 2024.

[35] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, et al., Retrieval-augmented generation for knowledge-intensive NLP tasks, Advances in neural information processing systems 33 (2020) 9459–9474.

[36] S. Banerjee, A. Agarwal, S. Singla, LLMs will always hallucinate, and we need to live with this (2024). doi:10.48550/arXiv.2409.05746. arXiv:2409.05746.