Media Engineering and Technology Faculty
German University in Cairo

Large Language Models for CyberSecurity

Bachelor Thesis

Author: Mohamed Elquesni
Supervisors: Dr. Tamer Moustafa

Submission Date: 29 May, 2025

Media Engineering and Technology Faculty
German University in Cairo

Large Language Models for CyberSecurity

Bachelor Thesis

Author: Mohamed Elquesni
Supervisors: Dr. Tamer Moustafa

Submission Date: 29 May, 2025

This is to certify that:

(i) the thesis comprises only my original work toward the Bachelor Degree

(ii) due acknowlegement has been made in the text to all other material used

---

Mohamed Elquesni
29 May, 2025

Abstract

The increasing complexity and scale of cyber threats have highlighted the need for intelligent, adaptive tools to assist analysts in securing digital infrastructure. This thesis explores the application of decoder-based large language models (LLMs) in cybersecurity, with the goal of exploring multiple independent cybersecurity tasks that reflect real-world analyst workflows.

Unlike traditional encoder-based models commonly used in security applications, this work focuses on decoder models such as DeepSeek, Mistral, LLaMA, and Gemma. These models are employed to handle diverse tasks including **(I) phishing email classification**, **(II) threat intelligence extraction**, **(III) automated query generation for threat hunting**, and **(IV) anomaly detection in logs**.

To enable consistent and accurate responses, the project applies prompt engineering and template-based instruction formatting across tasks. The models are evaluated on curated datasets and benchmarked against known encoder-based baselines. Fine-tuning is applied across all tasks except phishing detection: **Mistral-7B-Instruct** is used in a zero-shot setting for phishing; **LLaMA 3.1-8B-Instruct** is fine-tuned on the CTI-HAL dataset for threat intelligence extraction; **DeepSeek-Coder-6.7B-Instruct** is fine-tuned for log query generation; and **Gemma-1.1-Instruct** is fine-tuned on the KDDCup'99 dataset for anomaly detection.

Results demonstrate that, with well-structured prompting and targeted fine-tuning, decoder LLMs can achieve competitive accuracy and provide flexible advantages for deployment. Their open-source nature makes them particularly suitable for security-sensitive environments, where transparency, data control, and reproducibility are essential. Moreover, recent advancements in quantization and parameter-efficient fine-tuning (e.g., LoRA, QLoRA) have significantly reduced resource requirements, making these models easier to run and adapt on limited hardware.

In addition to strong task performance, decoder-based models can generate transparent, human-readable justifications for their outputs, making them especially useful in cybersecurity tasks that require explainability, which is an advantage over encoder models like BERT, which typically provide only classification without rationale.

This thesis presents a practical, modular architecture for integrating decoder LLMs into Security Operations Center (SOC) workflows. It also offers a foundation for future research on using generative artificial intelligence to support human analysts in cybersecurity operations.

Acknowledgments

All praise and gratitude are due first and foremost to Allah, whose guidance, mercy, and blessings made this journey possible.

A bachelor's thesis is more than a milestone. It is a reflection of sustained dedication, perseverance, and the unwavering support of those who stood by me.

I would like to express my heartfelt appreciation to my family and close friends for their constant encouragement, patience, and belief in me. Their presence was a pillar of strength throughout this endeavor, especially during the most challenging moments.

I am sincerely grateful to my supervisor, Dr. Tamer Moustafa, for his invaluable guidance, thoughtful feedback, and continuous support. His mentorship played a central role in shaping both the technical direction and the academic rigor of this work.

This achievement would not have been possible without the people who walked this path with me. Thank you.

"Success is not final, failure is not fatal: it is the courage to continue that counts."
— Winston S. Churchill

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Artificial Intelligence (AI) is rapidly transforming the field of cybersecurity, offering powerful tools to support the defense of increasingly complex digital systems. The frequency and sophistication of cyberattacks have grown significantly in recent years, placing immense pressure on security analysts and response teams [2, 3]. Security Operations Centers (SOCs) must now process vast amounts of unstructured data, including system logs, threat intelligence reports, phishing emails, and user behavior records. Traditional detection methods, which often rely on static rules or manual inspection, are struggling to keep up with the evolving landscape of threats [4].

Large Language Models (LLMs) have shown exceptional capabilities in understanding and generating human language [5]. However, the use of LLMs in cybersecurity remains limited, particularly in applications that require generation rather than classification. Most research in this field has focused on encoder-based models like Bidirectional Encoder Representations from Transformers (BERT) [6], which perform well in structured classification tasks but lack the generative flexibility needed for interactive and adaptive use cases.

Decoder-based LLMs, such as DeepSeek [7] and Mistral [8], offer a promising alternative. These models are capable of producing meaningful and context-aware responses to complex prompts, making them suitable for cybersecurity use cases like search query generation, threat report summarization, log interpretation, and incident response guidance. Yet, as noted by Struppek et al. [2], "there is limited use of decoder-only models in cybersecurity tasks. Most approaches still rely on encoder-based architectures due to their structured output capabilities." This highlights a clear research gap.

This thesis is driven by the opportunity to bridge that gap. It explores the potential of decoder-based LLMs as intelligent assistants capable of supporting human analysts in real time, using prompt-driven workflows across a range of cybersecurity functions.

1.2 Related Work and Promising Approaches

While most cybersecurity applications of language models have focused on encoder architectures, a small but growing body of research has begun to explore the potential of decoder models for interactive and generative tasks. These models, typically based on transformer decoder architectures such as GPT [5], Mistral [8], and DeepSeek [7], are better suited to use cases that require free-text generation, context-aware summarization, or prompt-driven workflows.

A notable contribution in this area is the CTI-HAL benchmark introduced by Della Penna et al. [9], which evaluated decoder models such as Claude 3 for the task of extracting structured cyber threat intelligence from real-world threat reports. Their results demonstrated that decoder models could accurately match MITRE ATT&CK techniques to threat statements, achieving competitive performance with encoder-based models while requiring less prompt engineering. Although the study was limited to offline cyber threat intelligence extraction, it highlighted the generative reasoning strength of decoder models in domain-specific tasks.

Another research direction has focused on phishing detection using instruction-tuned decoder models. Lee et al. [3] investigated the use of both encoder and decoder models for phishing email classification and found that decoder models offered greater flexibility and interpretability, especially in zero-shot and few-shot settings. Their experiments showed that decoder models were capable of generating textual justifications for classification decisions, which is not feasible using models optimized purely for classification.

There have also been early efforts to apply generative models in operational security contexts, such as security operations centers. Some community-driven prototypes, such as ThreatGPT and LogGPT [10, 11], have proposed prompt-driven assistants that help analysts by generating Splunk Processing Language (SPL) queries or suggesting response playbooks. However, these tools typically rely on hand-crafted rules, are supervised by human analysts, and lack academic validation or reproducible evaluation protocols.

Separately, anomaly detection in structured logs remains a core challenge in cybersecurity, particularly due to the high volume and complexity of system and network event data. While traditional statistical and rule-based systems are still widely used, they often struggle with novel attack patterns and contextual understanding. There is limited exploration of using decoder-based LLMs to interpret raw log entries and classify anomalous behaviors based on semantic cues, user patterns, or attack tactics presenting a key opportunity for this thesis.

1.3 Problem Statement and Objective

Despite the progress in applying large language models to cybersecurity, existing research remains fragmented and typically focuses on a single model or isolated task. There is a

lack of systematic evaluation across diverse pretrained models and minimal exploration of optimization strategies tailored to specific cybersecurity domains. Current tools are often limited by their architecture, lack generative flexibility, or fail to generalize across different types of security challenges.

This thesis addresses the problem of identifying how different decoder-based language models perform when adapted to core cybersecurity tasks. The objective is to iteratively explore multiple pretrained models and optimize their usage with a variety of prompting and fine-tuning techniques, gradually improving performance across each task domain until a final working prototype is achieved. A key requirement is that the resulting model be lightweight, open source, and suitable for deployment in highly sensitive environments, such as enterprise security operations centers, where privacy, explainability, and resource efficiency are critical.

Targeted Domains

- **Phishing Email Detection:** Identifying and explaining malicious content in email communications.
- **Log Anomaly Detection:** Detecting unusual behavior in system and network logs.
- **Threat Intelligence Analysis:** Extracting structured insights from unstructured threat reports.
- **Dynamic Threat Hunting:** Assisting analysts in creating and refining search queries to identify hidden threats.

1.4 Thesis Outline

This thesis is organized as follows:

- **Chapter 2** reviews the relevant background, including traditional detection techniques in cybersecurity, the evolution of large language models, and the distinction between encoder and decoder architectures. It also surveys existing work on the application of LLMs to cybersecurity and identifies the specific research gap this thesis addresses.
- **Chapter 3** details the methodological framework used in this study. It presents the problem formulation, describes the datasets and preprocessing steps, outlines the decoder-based models explored, and explains the prompting and fine-tuning strategies applied across each cybersecurity domain.

- **Chapter 4** presents the experimental results for all four domains: phishing email detection, log anomaly detection, threat intelligence analysis, and dynamic threat hunting. It includes both quantitative performance metrics and qualitative evaluations of the generated outputs, along with a discussion of challenges encountered during model comparison and optimization.

- **Chapter 5** summarizes the key contributions and findings of the thesis. It discusses the strengths and limitations of decoder-based LLMs in cybersecurity workflows and proposes directions for future research, including dataset improvements, fine-tuning techniques, and the deployment of LLM-based assistants in operational environments.

Chapter 2

Background

This chapter introduces the core concepts, models, and prior research that form the foundation of this thesis. It is structured into six main sections. Section 2.1 outlines the evolution of machine learning in cybersecurity, including both traditional approaches such as decision trees and support vector machines, and early deep learning models such as RNNs and CNNs. Section 2.2 explores the role of traditional machine learning techniques in cybersecurity and their typical use cases. Section 2.3 focuses on deep learning models for cybersecurity tasks, highlighting the strengths and limitations of architectures like LSTMs and GRUs. Section 2.4 presents the Transformer architecture and explains its advantages in modeling long-range dependencies. Section 2.5 introduces large language models and their adaptation to cybersecurity tasks. Finally, Section 2.6 discusses key gaps in the literature and motivates the use of decoder-based LLMs in this thesis.

2.1 Evolution of Machine Learning in Cybersecurity

Machine learning has gradually reshaped cybersecurity, shifting detection and analysis away from rigid rule-based systems toward adaptive and data-driven models. Early efforts relied on classical machine learning techniques such as Decision Trees, Support Vector Machines (SVMs), and Random Forests [12, 13]. These models were effective in automating tasks like spam detection and basic intrusion classification but required manual feature engineering and struggled with unstructured or sequential data.

To overcome these limitations, researchers introduced deep learning methods, beginning with Recurrent Neural Networks (RNNs) and their variants. These models were applied to logs, network events, and sequential threat data, offering better pattern recognition over time. More efficient versions, including Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) networks, improved stability and performance in detecting behavioral anomalies [14, 15]. However, their sequential nature made them difficult to scale, and they were eventually replaced in many settings by architectures that offered greater parallelism and contextual understanding.

This evolution paved the way for Transformer-based models, which are now the foundation of large language models (LLMs). These models offer new possibilities for handling unstructured threat intelligence, generating human-readable responses, and dynamically assisting analysts in real time.

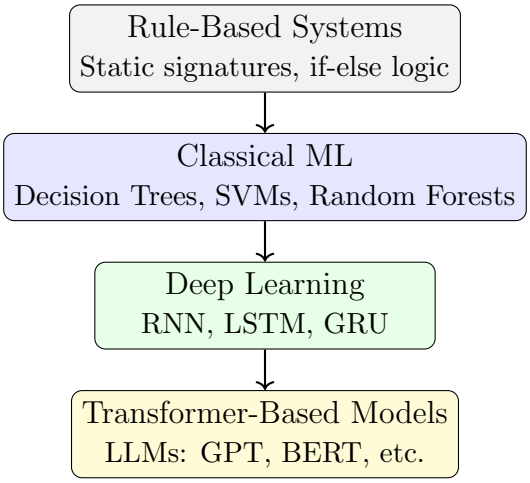

Figure 2.1: Evolution of machine learning approaches in cybersecurity

2.2 Traditional Machine Learning Techniques in Cybersecurity

Prior to the advent of deep learning, traditional machine learning (ML) models played a pivotal role in advancing cybersecurity automation. These algorithms replaced static, rule-based systems by introducing statistical decision-making and pattern recognition on structured datasets, such as connection logs, user metadata, and engineered URL features.

Algorithms such as Support Vector Machines (SVMs), Decision Trees, Random Forests, Naive Bayes, and k-Nearest Neighbors (k-NN) gained popularity for their computational efficiency, transparency, and strong performance on curated features. They were particularly effective in early applications including phishing URL detection, malware classification, and anomaly-based intrusion detection [12, 13].

Despite their utility, these models exhibited limitations in adapting to real-world, high-dimensional environments. Their effectiveness often depended on handcrafted feature extraction pipelines, which required domain expertise and constant tuning. More importantly, they were not well suited for processing unstructured data sources such as system logs, command-line inputs, or textual threat reports. They also lacked the ability to capture temporal dependencies, which are crucial for modeling dynamic attack behaviors.

As the complexity and volume of cybersecurity data grew, these shortcomings highlighted the need for models capable of automatic feature learning and sequence modeling. This catalyzed a shift toward deep learning, which offered better adaptability to diverse inputs and the capacity to generalize from raw, unlabeled data. Nevertheless, traditional ML models continue to serve as useful baselines in cybersecurity research and are often favored in resource-constrained environments due to their interpretability and speed.

2.2.1 Support Vector Machines (SVMs)

Support Vector Machines (SVMs) are supervised learning models that construct a hyperplane to optimally separate data points into distinct classes. In cybersecurity, they have been used extensively for classification tasks such as phishing detection, spam filtering, and intrusion detection [12].

Given a training dataset with input vectors $\mathbf{x}_i \in R^n$ and binary class labels $y_i \in \{-1, +1\}$, the objective of a linear SVM is to find a decision boundary of the form $\mathbf{w}^\top \mathbf{x} + b = 0$ that maximizes the margin between the two classes. The margin is defined as the distance between the hyperplane and the nearest data points from each class, known as *support vectors*. The optimization problem can be formally stated as:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{subject to} \quad y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1, \quad \forall i$$

SVMs are particularly well-suited for high-dimensional spaces and are effective in scenarios with a clear margin of separation. Their robustness to overfitting makes them appealing for cybersecurity tasks, where the cost of false positives can be high. For non-linearly separable data, kernel functions such as the radial basis function (RBF) can be used to project inputs into higher-dimensional spaces where linear separation becomes possible.

Figure 2.2 illustrates the key geometric concepts behind SVMs in a two-dimensional feature space. The plot shows:

- The **decision boundary** (dashed line), representing the optimal separating hyperplane.
- Two **margin boundaries** (dotted lines), offset by $+1$ and -1 from the hyperplane.
- **Support vectors**, displayed as hollow circles with a black outline, lying exactly on the margin boundaries.
- **Positive and negative class points**, shown as filled blue and red circles, respectively.
- **Shaded margin regions**, visually separating the classes while maximizing the margin width.

Only the support vectors influence the orientation and position of the hyperplane. This characteristic underpins the model's generalization capability and computational efficiency. All other data points, though correctly classified, do not contribute directly to the decision boundary.

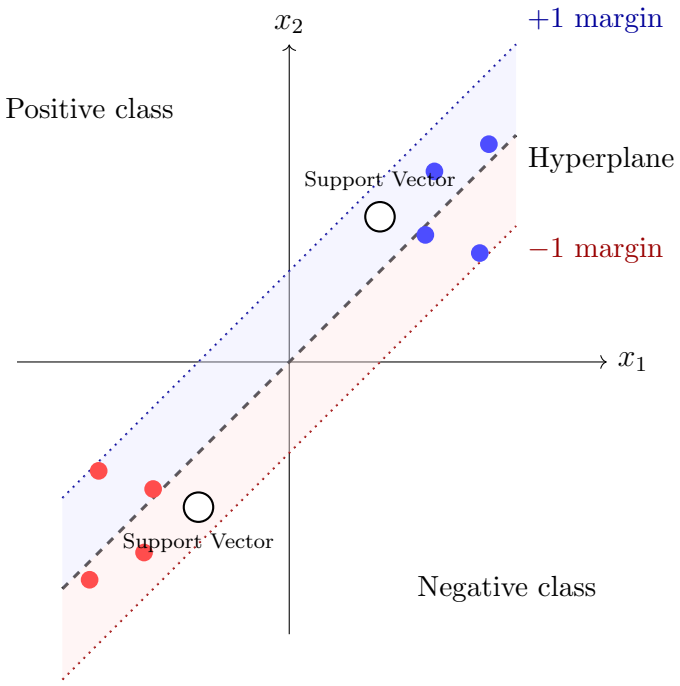

Figure 2.2: SVM with a maximum-margin hyperplane, margin boundaries, support vectors, and shaded margin regions.

2.2.2 Decision Trees and Random Forests

Decision Trees are hierarchical classifiers that recursively partition the feature space based on decision rules derived from metrics such as information gain or Gini impurity. Their simplicity and interpretability have made them widely used in early cybersecurity applications, including phishing detection, malware classification, and anomaly-based intrusion detection [16, 17].

However, individual decision trees are susceptible to overfitting, especially in noisy environments or when trained on limited data. To overcome this, Random Forests were introduced as an ensemble method that constructs multiple trees on bootstrapped subsets of the data and aggregates their outputs through majority voting. This ensemble approach significantly improves generalization and reduces model variance while maintaining the interpretability of individual trees [18].

In cybersecurity, Random Forests have demonstrated robust performance in tasks such as detecting phishing URLs, classifying malicious executables, and filtering spam [19].

They are particularly effective on structured datasets with engineered features and are favored in scenarios where model transparency and low computational cost are critical.

Nevertheless, these models remain limited in their ability to process unstructured data sources such as raw logs, natural language threat reports, or shell command histories. Their dependence on manual feature engineering reduces adaptability in evolving threat environments.

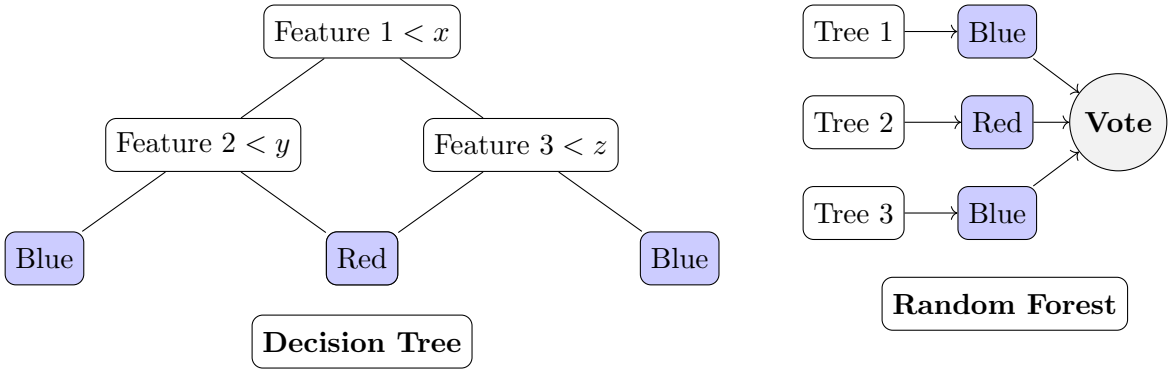

Figure 2.3: Comparison between a Decision Tree and a Random Forest. A decision tree uses a single path of feature splits, while a random forest aggregates the outputs of multiple trees via majority voting.

2.2.3 Naive Bayes and k-Nearest Neighbors (k-NN)

Naive Bayes and k-Nearest Neighbors (k-NN) are foundational algorithms in traditional machine learning and have been widely applied in various cybersecurity contexts.

Naive Bayes is a probabilistic classifier based on Bayes' theorem, which assumes conditional independence between features given the class label. Despite this strong assumption, Naive Bayes models have shown excellent performance in text-based tasks such as spam filtering and phishing email classification [20]. Their efficiency, scalability, and interpretability make them suitable for real-time security applications, particularly in detecting email threats and URL-based attacks.

In phishing detection, Naive Bayes classifiers typically analyze handcrafted features extracted from email headers, message content, or embedded URLs. For example, Jayaprakash et al. [20] demonstrated that Naive Bayes achieved 89.8% accuracy in identifying phishing URLs in real-time scenarios, validating its effectiveness in lightweight environments.

k-Nearest Neighbors (k-NN) is a non-parametric, instance-based learning algorithm that classifies a test point by majority vote among its k closest labeled examples in the feature space. The method is simple yet powerful, particularly when applied to structured tabular data. In cybersecurity, k-NN has been successfully used in intrusion detection systems (IDS) to classify network traffic into benign and malicious categories.

Sharma et al. [21] applied k-NN to the ISCX intrusion detection dataset and achieved a classification accuracy of 99.96%, demonstrating the model's ability to generalize well in detecting anomalous behavior patterns in network flows.

Figure 2.4 provides a conceptual overview of how Naive Bayes and k-NN differ in their classification mechanisms.

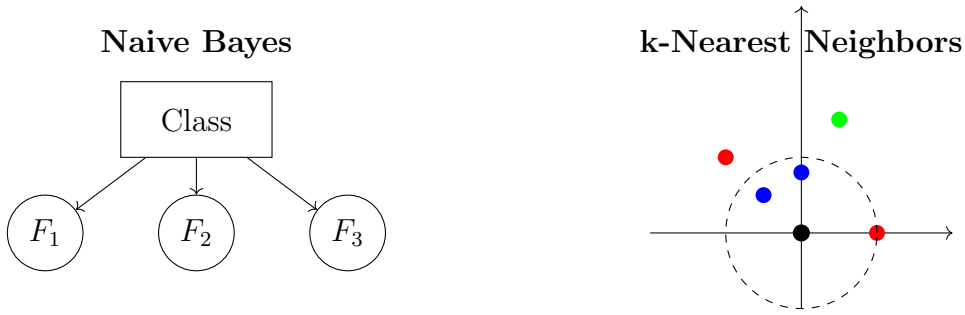

Figure 2.4: Illustration of Naive Bayes (left) showing conditional independence between features given the class, and k-Nearest Neighbors (right) depicting classification based on proximity to labeled examples.

2.3 Deep Learning for Cybersecurity

As cybersecurity threats evolved in complexity, traditional machine learning models began to show limitations, particularly in handling unstructured data and temporal dependencies. This led to the adoption of deep learning approaches, which leverage layered neural networks capable of learning hierarchical feature representations from raw inputs.

Recurrent Neural Networks (RNNs) and their variants, such as **Long Short-Term Memory (LSTM)** and **Gated Recurrent Units (GRUs)**, were among the first deep learning models applied in cybersecurity. These architectures are designed to process sequential data, making them suitable for modeling time-series information such as system logs, user activity traces, or network flow patterns. Studies have shown their effectiveness in detecting behavioral anomalies and temporal attack signatures [22, 23].

Convolutional Neural Networks (CNNs) have also been used in cybersecurity tasks, especially for static malware detection and binary classification problems. By treating sequences or hex-dumps of binaries as visual inputs, CNNs can capture local spatial features that traditional approaches might miss [24, 25].

Unlike traditional ML models, deep learning architectures reduce the need for manual feature engineering, offering end-to-end learning pipelines. However, they require significantly more data and computational resources, which can pose challenges in deployment, especially in real-time security operations.

Despite these constraints, deep learning has become a cornerstone in modern cybersecurity solutions, particularly in detecting zero-day attacks, automating threat classification, and learning from unstructured sources such as raw logs, command-line arguments, and natural language threat intelligence.

2.3.1 Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) are a class of neural networks adept at modeling sequential data by maintaining a hidden state that captures information about previous inputs. This capability makes them particularly suitable for cybersecurity tasks involving time-series data, such as system log analysis, network traffic monitoring, and intrusion detection.

In cybersecurity applications, RNNs have demonstrated effectiveness in various domains. For instance, Harun et al. [26] showed that RNNs can outperform traditional machine learning algorithms in tasks like incident detection, fraud detection, and malware classification by effectively capturing temporal dependencies in data. Tuor et al. [27] employed RNN-based models for anomaly detection in system logs, achieving high accuracy by learning patterns of normal behavior and identifying deviations.

Despite their advantages, standard RNNs face challenges such as the vanishing gradient problem, which hampers their ability to learn long-term dependencies. This limitation has led to the development of advanced architectures like Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs), which address these issues and will be discussed in subsequent sections.

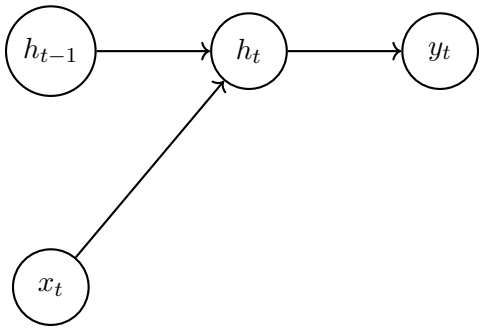

Figure 2.5: Basic RNN cell: the current hidden state h_t is computed from the current input x_t and the previous hidden state h_{t-1}.

2.3.2 Long Short-Term Memory (LSTM) Networks

Long Short-Term Memory (LSTM) networks are a specialized type of RNN designed to overcome the vanishing gradient problem and capture long-term dependencies in sequential data. Introduced by Hochreiter and Schmidhuber [28], LSTMs incorporate gating mechanisms such as input, output, and forget gates to regulate the flow of information across time steps.

In cybersecurity, LSTMs have been widely used for modeling sequences of user actions, system log entries, and network activity. Kim et al. [22] applied LSTM models to detect cyber intrusions and showed improved accuracy over traditional RNNs and conventional machine learning classifiers. Similarly, Yin et al. [23] leveraged LSTMs in a hybrid deep

learning architecture for intrusion detection on the NSL-KDD dataset, outperforming baseline models in both precision and recall.

The key advantage of LSTMs lies in their ability to maintain memory over long time spans. This makes them well-suited for detecting multi-stage attacks and temporal threat patterns. However, LSTMs can be computationally intensive and require significant training data, which may limit their practical use in resource-constrained environments.

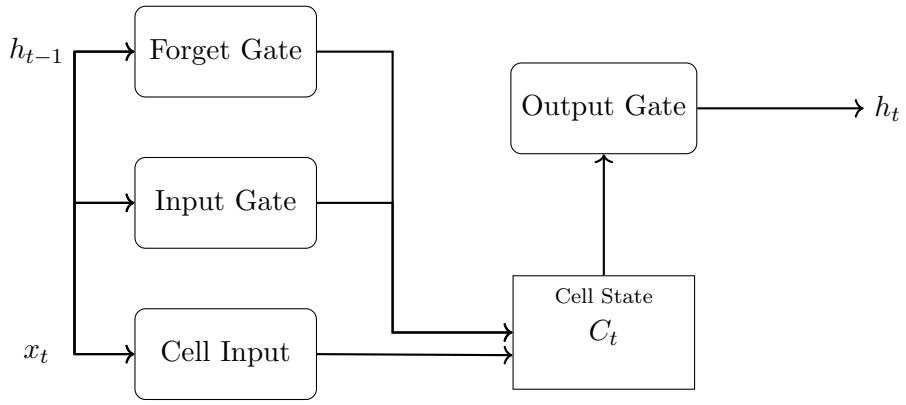

Figure 2.6: Clean architecture of an LSTM cell. Inputs x_t and h_{t-1} interact through forget, input, and output gates to produce the updated cell state C_t and hidden state h_t.

2.3.3 Gated Recurrent Units (GRUs)

Gated Recurrent Units (GRUs) are a streamlined alternative to LSTM networks that aim to retain similar modeling capabilities while reducing architectural complexity. Introduced by Cho et al. [29], GRUs merge the forget and input gates into a single update gate and remove the explicit memory cell, resulting in fewer parameters and faster training.

In cybersecurity, GRUs have been applied to tasks that require modeling sequential patterns, such as intrusion detection, anomaly recognition, and phishing detection. Studies have shown that GRUs can match or outperform LSTMs in terms of accuracy while being more computationally efficient [30]. Their lightweight structure makes them particularly suitable for deployment in real-time systems and resource-constrained environments such as edge devices and IoT platforms.

GRUs compute the hidden state h_t directly based on the previous state and current input, modulated by update and reset gates. This allows GRUs to adaptively control the extent to which previous information is retained or overwritten at each time step.

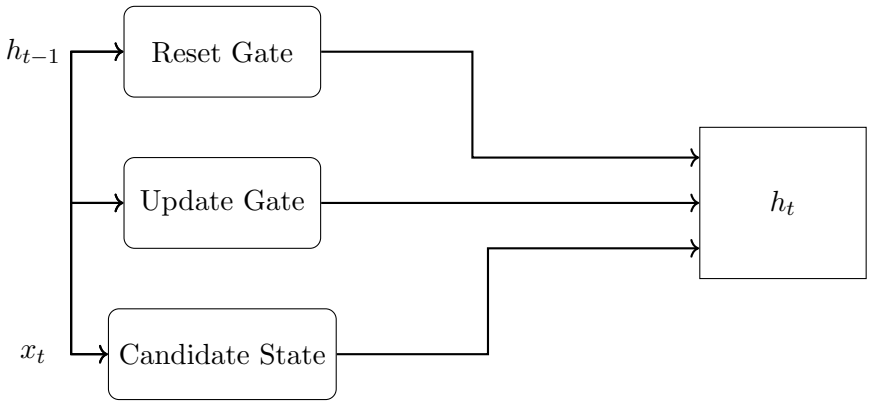

Figure 2.7: GRUs use reset, update, and candidate state components to compute the updated hidden state h_t from x_t and h_{t-1}. Arrows are vertically separated to avoid overlap.

2.3.4 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) were originally developed for image recognition tasks, where their hierarchical structure enables them to detect spatial patterns such as edges, textures, and shapes. However, their success in computer vision has inspired adaptations for cybersecurity applications that benefit from local feature extraction and translation invariance, such as malware detection, byte-level analysis, and log pattern recognition.

In cybersecurity, CNNs are particularly useful in cases where raw input data can be represented as spatial or sequential grids. For example, executable files can be transformed into grayscale images or byte sequences, allowing CNNs to identify malicious patterns across fixed-length windows [31]. Similarly, system call traces, packet payloads, and log sequences can be encoded into matrices, enabling CNN-based models to learn localized anomaly patterns efficiently [32].

A basic CNN architecture consists of alternating layers of convolution, activation, and pooling. The convolutional layers apply learnable filters over the input space, extracting local features. These features are then passed through nonlinear activation functions (e.g., ReLU) and downsampled using pooling layers to reduce dimensionality while preserving the most salient information. In cybersecurity, this approach allows models to generalize across similar but non-identical attack patterns.

Despite being less common than RNNs or Transformers for text-heavy cybersecurity tasks, CNNs offer several advantages:

- **Speed and parallelism:** Unlike RNNs, CNNs do not require sequential processing, enabling faster inference.
- **Local pattern focus:** They excel at identifying localized threats in binary files, shellcode, or command-line segments.

- **Low parameter count:** CNNs can be lightweight, making them suitable for edge deployment in embedded security systems.

However, CNNs are less suited to tasks requiring long-range dependency modeling or context-aware understanding, limiting their effectiveness in natural language-heavy applications such as threat report analysis or phishing detection. As a result, they are often used in combination with other models or as feature extractors in hybrid pipelines.

Figure 2.8: A simplified CNN architecture for classifying raw executable files as malicious or benign.

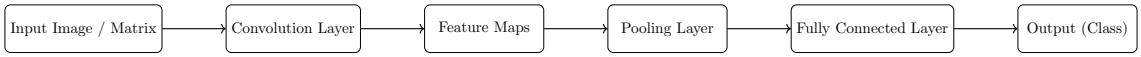

Figure 2.9: General flow of a Convolutional Neural Network: from raw input to classification output via feature extraction and pooling.

2.4 Transformers and Attention Mechanisms

The Transformer architecture represents a major milestone in the development of deep learning models and forms the backbone of modern large language models (LLMs). Introduced by Vaswani et al. [33], Transformers eliminate the need for recurrence or convolution by relying entirely on a mechanism known as *self-attention*. This innovation allows the model to process input sequences in parallel while dynamically weighing the importance of each token relative to the others.

Unlike Recurrent Neural Networks (RNNs), which process inputs sequentially and often struggle with long-term dependencies, Transformers can capture relationships between distant tokens through their attention mechanism [34]. This makes them highly efficient and well-suited for applications that require contextual understanding across an entire input sequence.

These properties are particularly valuable in cybersecurity, where the relationships between indicators of compromise (IoCs), user actions, or log events may span across multiple lines or even documents. The ability to process such data efficiently and in parallel has made Transformers the foundation for powerful LLMs such as BERT [6], GPT [5], Mistral [8], and DeepSeek [7]. These models are increasingly being explored for tasks like real-time threat detection, automated intelligence analysis, phishing email interpretation, and response guidance in security operations.

2.4.1 Self-Attention Mechanism

At the heart of the Transformer lies the *self-attention* mechanism, which allows the model to weigh the relevance of each token in a sequence relative to every other token [33]. This capability enables Transformers to model long-range dependencies and complex contextual relationships that traditional recurrent architectures struggle with.

In self-attention, each input token is projected into three vectors: a **query** (Q), a **key** (K), and a **value** (V). The attention score between a query and all keys determines how much each value should contribute to the output. The computation is as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

Here, Q, K, and V are matrices formed by stacking the respective vectors for each token, and d_k is the dimensionality of the key vectors. The dot product of Q and K captures similarity, and the softmax operation ensures the resulting attention weights form a probability distribution over input positions.

This operation enables the model to dynamically determine which tokens to focus on, regardless of their position in the sequence. In cybersecurity tasks, this means a model can, for example, correlate a suspicious command at the start of a log entry with an IP address at the end, or link a phishing domain to an attachment type in the same email.

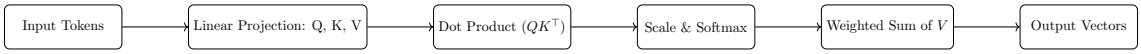

Figure 2.10: The self-attention mechanism: each input token is transformed into query, key, and value vectors. Attention weights are computed via dot-product and softmax, and applied to a weighted sum of values.

2.4.2 Positional Encoding

Since the Transformer architecture does not include any recurrence or convolution, it lacks an inherent notion of token order. To preserve the sequence information, Transformers use *positional encoding*, which injects information about the position of each token into the model's input representations [33].

The original Transformer uses **sinusoidal positional encodings**, which are fixed functions of position and embedding dimension. These are added to the input embeddings before any attention operations:

$$\text{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right), \quad \text{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

Here, pos is the token position, i is the dimension index, and d_{model} is the embedding dimension. These sinusoidal functions enable the model to learn relative as well as absolute positions, since any position can be represented as a linear function of others.

Alternative approaches, such as learned positional embeddings, are used in models like BERT [6] and GPT [5], where the positional vectors are trained alongside the model weights.

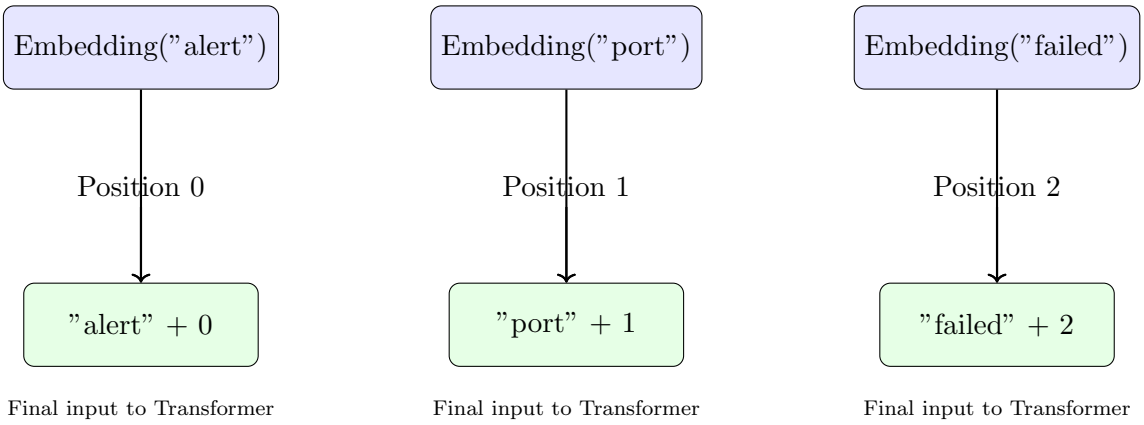

Figure 2.11: Each word in the sentence is converted into an embedding and combined with its position encoding. The resulting vector is passed to the Transformer as input.

Multi-Head Attention and Transformer Layer Composition

One of the key innovations in the Transformer architecture is the use of *multi-head attention*, which enables the model to attend to different types of relationships between tokens in parallel [33]. Instead of computing a single attention distribution, the input is processed through multiple attention heads, each with its own learned set of query, key, and value projections.

Each head captures distinct semantic or syntactic aspects of the sequence. For example, one head may focus on identifying IP addresses, while another captures access control verbs. This is particularly useful in cybersecurity contexts, where both lexical and structural cues are essential.

The outputs from all heads are concatenated and passed through a final linear transformation to form the complete attention output:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$

Each individual head is computed as:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Here, W_i^Q, W_i^K, and W_i^V are learned projection matrices for the ith head, and W^O is the output projection matrix applied after concatenation.

A Transformer block typically consists of the following components:

- Multi-head self-attention mechanism
- Layer normalization
- Position-wise feed-forward neural network
- Residual (skip) connections

These components are stacked in multiple layers to form both encoder and decoder blocks. This modular structure allows the model to scale efficiently while capturing complex hierarchical patterns over long sequences.

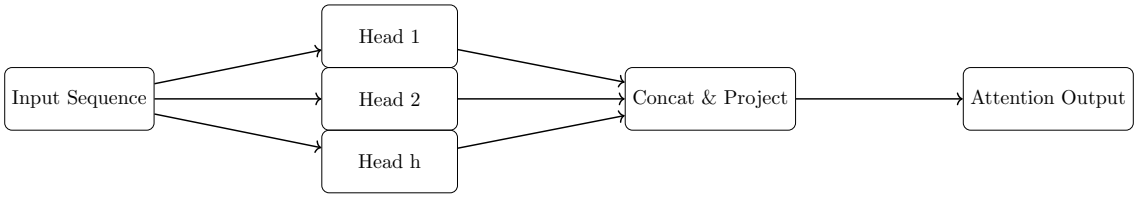

Figure 2.12: Multi-head attention structure: the input is processed by multiple attention heads in parallel, then combined and projected to produce the final output.

2.4.3 Feed-Forward Networks

Feed-Forward Networks (FFNs) are a fundamental component of Transformer architectures, serving as position-wise fully connected layers that follow the self-attention mechanism within each encoder or decoder layer. After the self-attention module computes contextual relationships among tokens, the FFN processes each token independently to enhance its representation by applying non-linear transformations.

Formally, for each token representation $x \in R^d$, the FFN applies two linear transformations separated by a non-linear activation function (typically ReLU), as expressed by:

$$\mathrm{FFN}(x) = \mathrm{ReLU}(xW_1 + b_1)W_2 + b_2,$$

where $W_1 \in R^{d \times d_{ff}}$, $W_2 \in R^{d_{ff} \times d}$, and b_1, b_2 are bias vectors. Here, d is the embedding dimension and d_{ff} is the intermediate layer dimension, often larger to increase model capacity.

The combination of self-attention and FFN layers allows the Transformer to both capture global contextual dependencies among tokens and perform complex, non-linear transformations on individual token embeddings. This synergy is essential for generating rich and expressive representations that can be used for downstream tasks such as classification, generation, or sequence labeling.

In practice, the Transformer processes an input sequence by first embedding tokens into vectors, then applying multiple layers each composed of a self-attention block followed

by an FFN block. Residual connections and layer normalization are applied around these sublayers to facilitate training and stability [33, 35].

Recent works in Transformer improvements and applications emphasize the critical role of FFNs in enhancing model expressivity and generalization, particularly in domains like natural language processing and cybersecurity [36].

2.5 Large Language Models for Cybersecurity

Large Language Models (LLMs) are advanced deep learning systems built on the Transformer architecture [33], designed to process and generate human-like language across a wide variety of tasks. Initially developed for general-purpose natural language understanding and generation, LLMs have since demonstrated the capacity to handle highly specialized domains by capturing contextual patterns, abstract reasoning, and domain-specific terminology. In recent years, their capabilities have been increasingly explored in cybersecurity, where the ability to analyze unstructured threat intelligence, interpret log data, and generate meaningful responses is essential [2, 37].

The architecture of these models plays a critical role in determining how they operate and what types of tasks they are best suited for. Transformer-based LLMs are typically categorized into three types: encoder-only, decoder-only, and encoder–decoder architectures. Each type has unique strengths, ranging from text classification and entity extraction to free-text generation and summarization. Understanding these architectural differences is essential when selecting a model for a given cybersecurity application.

While pretrained LLMs offer strong generalization capabilities, their effectiveness in cybersecurity depends on appropriate adaptation strategies. This includes prompting techniques, full and instruction-based fine-tuning, and parameter-efficient approaches such as Low-Rank Adaptation (LoRA) [38] and Quantized LoRA (QLoRA) [39]. These methods enable LLMs to align with security-specific goals, maintain high accuracy, and operate efficiently even in resource-constrained environments.

2.5.1 Transformer Architectures and Model Types

Transformer architectures form the computational foundation of modern large language models (LLMs). Introduced by Vaswani et al. [33], the Transformer relies on self-attention mechanisms to model relationships between all elements in an input sequence, allowing for scalable and parallelizable language processing. While the core architecture is consistent, LLMs typically adopt one of three structural configurations: **encoder-only**, **decoder-only**, and **encoder–decoder**.

Encoder-only models such as BERT [6], SecureBERT [40], and CTI-BERT [41] process input sequences bidirectionally using masked language modeling (MLM). These models are ideal for classification, entity recognition, and token-level tasks, making them

well-suited for cybersecurity applications involving threat labeling, named entity recognition, and phishing detection. Their strength lies in understanding the entire context of an input simultaneously, rather than generating outputs.

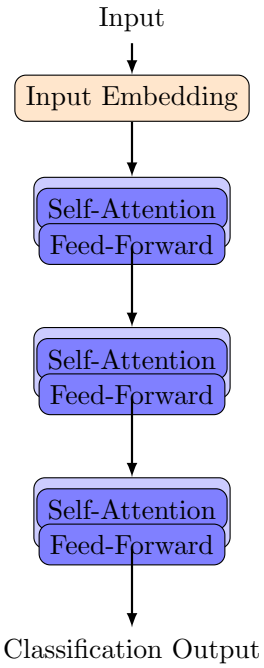

Figure 2.13: Smaller Encoder-Only Transformer Architecture showing sequential data flow from input embedding through stacked encoder layers to classification output.

Decoder-only models such as GPT [5], Mistral [8], and DeepSeek [7] are designed to generate sequences autoregressively. They use a **Masked Self-Attention** mechanism which prevents each token from attending to future tokens, ensuring that outputs are generated one token at a time conditioned only on previous tokens and input prompts. This structure allows these models to excel at generative tasks such as text completion, log summarization, and query generation.

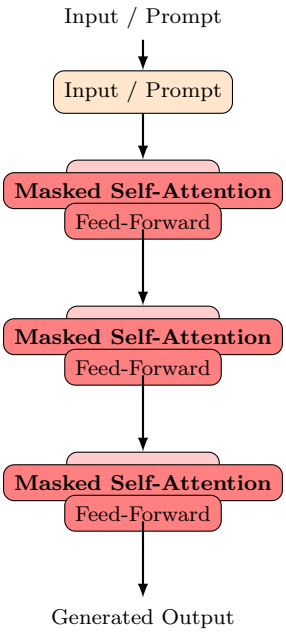

Figure 2.14: Smaller Decoder-Only Transformer Architecture showing sequential data flow from input prompt through stacked decoder layers with masked self-attention to generated output.

Decoder-only models are particularly useful in cybersecurity tasks that require generating natural language outputs based on context, such as generating incident reports, summarizing logs, or dynamically formulating queries for threat hunting. Their autoregressive nature ensures that the generated text is coherent and contextually relevant, which is essential for real-time analyst assistance and automated response generation [9].

Encoder–decoder models, also known as sequence-to-sequence models, consist of an encoder that processes the entire input sequence and a decoder that generates output sequences conditioned on the encoder's representations. Notable examples include T5 [42] and BART [43].

The encoder transforms the input tokens into rich contextual embeddings by applying stacked layers of self-attention and feed-forward networks. The decoder then generates the output sequence token by token, using masked self-attention to prevent attending to future tokens during generation. Additionally, it incorporates cross-attention layers that allow the decoder to focus on relevant parts of the encoder's output. This design enables the model to effectively perform tasks such as machine translation, summarization, and structured-to-text generation. In cybersecurity, encoder–decoder models are particularly useful for converting raw logs into readable incident narratives or summarizing complex threat intelligence reports.

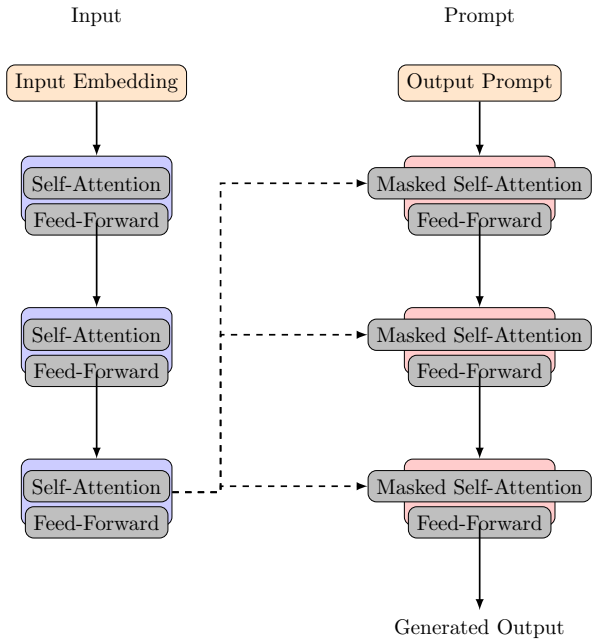

Figure 2.15: Compact Encoder–Decoder Transformer Architecture showing the encoder processing the input sequence and the decoder generating output sequentially, with cross-attention linking encoder and decoder layers.

2.5.2 Fine-Tuning and Adaptation of Large Language Models for Cybersecurity

Large Language Models (LLMs) pretrained on vast corpora can be adapted for cybersecurity tasks through various fine-tuning and prompt-based methods. These include **zero-shot and few-shot learning** techniques, which leverage prompting without modifying model weights [5], and **parameter-efficient fine-tuning** methods such as LoRA [38] and QLoRA [39] that enable updating a small number of parameters efficiently. Advanced toolkits like Unsloth facilitate these fine-tuning approaches on large models with reduced computational resources [44].

Zero-shot and Few-shot Prompting for Cybersecurity Tasks

Zero-shot and few-shot prompting are powerful techniques that enable large language models (LLMs) to perform tasks without gradient updates. In **zero-shot prompting**, a task is described directly in the prompt, and the model generates an answer based solely on its pretrained knowledge. In contrast, **few-shot prompting** supplies the model with several examples of the task within the prompt, improving performance by demonstrating the expected input-output behavior.

These techniques are particularly useful in cybersecurity applications where labeled data is scarce or fast deployment is necessary. For example:

- In zero-shot mode, an LLM can classify whether a given email is phishing or benign by simply prompting: `"Classify the following email as phishing or safe: {email}"`.
- In few-shot mode, several labeled examples of phishing and non-phishing emails can be prepended to the prompt to improve contextual understanding and accuracy.

Prompt-based approaches have shown competitive results across various domains, including cybersecurity tasks such as threat classification, IOC extraction, and log interpretation [45, 46, 47]. Their flexibility allows rapid adaptation without retraining, making them ideal for real-time incident response and dynamic threat hunting environments.

Retrieval-Augmented Generation and Prompt Engineering

Retrieval-Augmented Generation (RAG) is an approach that enhances large language models (LLMs) by supplementing their responses with relevant external knowledge. Instead of relying solely on the model's internal parameters, RAG retrieves documents from a knowledge base and incorporates them into the generation process. This is particularly useful in cybersecurity, where models may need to reference up-to-date threat intelligence, vulnerability reports, or attack techniques.

RAG architectures, first introduced by Lewis et al. [48], typically consist of:

- A *retriever* module that identifies the most relevant documents or passages based on the input query.
- A *generator* (LLM) that uses the retrieved content as context to produce a more accurate and grounded output.

In cybersecurity applications, RAG can be used to generate incident summaries using both historical logs and retrieved threat reports, or to answer analyst queries with real-time context from sources such as MITRE ATT&CK or CVE databases.

Prompt Engineering refers to the design and structuring of input prompts to steer LLM outputs in predictable and useful ways. While zero-shot and few-shot prompting focus on providing task examples, prompt engineering includes broader techniques such as template formatting, role instructions, keyword constraints, and output shaping. These methods are critical in cybersecurity workflows where structured outputs, such as JSON responses or SPL queries, are often required.

For instance, prompt engineering can be used to generate Splunk queries from analyst instructions, classify phishing emails with explanations, or extract indicators of compromise (IOCs) from free-text reports. By combining retrieval-based context with carefully engineered prompts, LLMs can achieve both task relevance and response reliability in operational settings.

Low-Rank Adaptation (LoRA)

LoRA introduces trainable low-rank matrices injected into the transformer layers, enabling parameter-efficient fine-tuning by freezing the original model weights [38]. This method significantly reduces memory requirements and training time, making it practical for adapting large LLMs to cybersecurity tasks like intrusion detection and threat intelligence extraction.

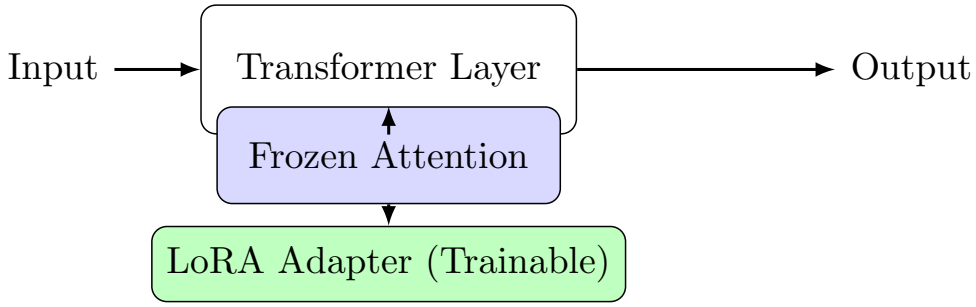

Figure 2.16: LoRA adaptation inside a Transformer layer: input flows through frozen attention weights, is augmented by trainable low-rank LoRA adapters, and passed onward. Only the LoRA adapters are updated during training.

Quantized Low-Rank Adaptation (QLoRA)

Quantized Low-Rank Adaptation (QLoRA) is an advanced parameter-efficient fine-tuning technique that enables the training of large language models on limited hardware resources by combining low-rank adapters with quantized base models [39]. While it follows the same core principle as LoRA, freezing the base model and injecting trainable low-rank matrices, QLoRA introduces additional memory optimization strategies to support efficient training of models with tens of billions of parameters.

In QLoRA, the pretrained model is first quantized to 4-bit precision using double quantization techniques. This drastically reduces memory usage while retaining the model's representational capacity. The quantized model is kept frozen during fine-tuning, and LoRA adapters are injected into key projection layers (and optionally other layers), allowing the model to learn task-specific representations without modifying the original weights.

To support large-scale models on standard GPUs, QLoRA also employs memory-aware optimization strategies such as paged optimizers and offloading techniques. These enhancements make it possible to fine-tune models like LLaMA-65B on a single 48 GB GPU, significantly lowering the hardware barrier for domain-specific adaptation.

QLoRA is particularly impactful in cybersecurity contexts, where deploying and fine-tuning large LLMs can be prohibitively expensive. Its efficient use of compute makes it a practical and scalable choice for training security-focused models without sacrificing performance.

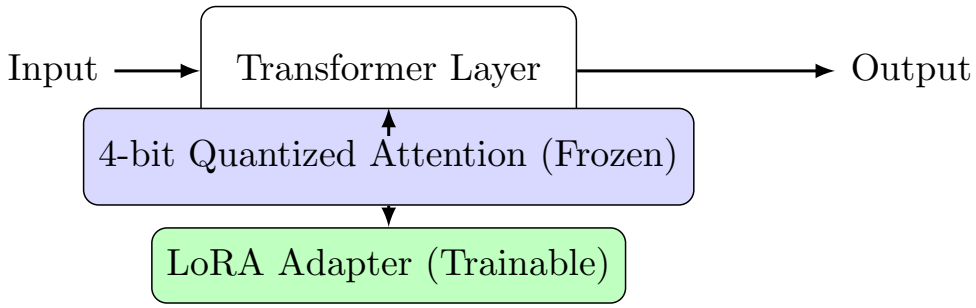

Figure 2.17: QLoRA adaptation inside a Transformer layer: input flows through frozen 4-bit quantized attention weights, is augmented by trainable LoRA adapters, and passed onward. Only the LoRA adapters are updated during training.

2.6 Literature Review and Research Gap

This section analyzes existing trends in cybersecurity-focused LLM research and highlights key gaps. While the field has seen rapid growth in recent years, particularly following the release of GPT-based models, there remains an overreliance on closed-source and encoder-focused systems. These limitations have left several aspects of interactive analyst support underexplored. The subsections below provide a breakdown of current usage patterns and outline the identified research gap that motivates this thesis.

2.6.1 Surge in LLM Adoption and Overreliance on Closed Models

The rapid growth of large language models (LLMs) in the field of cybersecurity became especially pronounced in 2023 and 2024, following the mainstream adoption of generative AI tools such as ChatGPT. According to the 2024 systematic literature review by Xu et al. [1], which analyzed over 70 research papers from 2023 to early 2024, the majority of the work still centers around **closed-source models**, most notably OpenAI's GPT-3.5 and GPT-4 [5].

While this surge reflects a growing interest in LLM capabilities for cybersecurity tasks, it also reveals a heavy reliance on **commercial APIs** that are neither transparent nor reproducible. These models often operate as **black boxes**, which means they lack visibility into their decision-making processes. This is particularly problematic for high-stakes domains such as threat detection, incident response, and compliance. Additionally, their proprietary nature limits real-world deployment in **security-sensitive environments** where data confidentiality and auditability are critical.

Table 2.1 illustrates the distribution of model types in cybersecurity-focused LLM research, based on the 2024 systematic literature review by Xu et al. [1].

Table 2.1: LLM Usage in Cybersecurity Research (2023–2024) [1]

Model Type	**% of Papers (n=72)**	**Notes**
GPT-3.5 / GPT-4 (Closed-source)	52.8%	API-based
BERT-family (Encoder-only)	33.3%	Task-specific classifiers
Decoder-only (Open-source)	6.9%	Rarely fine-tuned
Multimodal / Custom Architectures	6.9%	Typically experimental

These figures show that **over half of the literature** continues to prioritize models that are fundamentally misaligned with real-world cybersecurity constraints. While closed-source LLMs like GPT-4 are strong in general reasoning, they introduce concerns around **cost**, **explainability**, and **integration**. As of early 2024, only 6.9% of reviewed works attempted to evaluate **open-source decoder-only models**, despite the growing availability of high-performing alternatives such as DeepSeek [7] and Mistral [8].

2.6.2 Identified Research Gap

Despite the rise in LLM adoption, current research largely overlooks the potential of **lightweight, open-source decoder models** that can be adapted to interactive cybersecurity roles. Existing encoder-based models such as SecureBERT [40], CyBERT [4], and PhishingBERT [3] focus primarily on classification tasks and lack the ability to explain, guide, or engage with analysts in meaningful ways.

Moreover, even when decoder models are used, they are often evaluated in a zero-shot or prompt-only setting, without any domain-specific fine-tuning. This results in unpredictable behavior, particularly in operational environments such as Security Information and Event Management (SIEM) platforms, where precision and clarity are essential. For instance, tasks such as generating Splunk Processing Language (SPL) queries demand models with reliable structural understanding, which general-purpose pretrained models often fail to demonstrate.

At the same time, newer models like DeepSeek-Coder, Mistral, and Gemma have shown performance on par with GPT-3.5 in several benchmarks [7, 8, 49], while being **fully open-weight**, **cheaper to deploy**, and **easier to adapt**. However, these models remain significantly underexplored in the cybersecurity literature.

This thesis addresses that gap by investigating the application of decoder-based LLMs that are:

- **Open-weight and lightweight** to ensure practical deployment
- **Fine-tuned or structured-prompted** to align with cybersecurity use cases
- **Designed to assist analysts** rather than only classify

Chapter 3

Methodology

This chapter presents the methodological framework adopted to explore the use of decoder-based large language models (LLMs) in cybersecurity. The focus is on four key tasks relevant to analyst workflows within Security Operations Centers (SOCs): phishing email detection, threat intelligence analysis, network/system log anomaly detection, and log query generation.

For each task, we describe the problem formulation, the selected models and their assignments, dataset preparation, prompt or fine-tuning strategies, and the evaluation metrics used. The methodology emphasizes the use of open-weight, instruction-tuned LLMs that are efficient, interpretable, and capable of generating structured outputs. Practical constraints such as memory efficiency and real-world applicability are also considered throughout the design process.

3.1 Problem Definition

This thesis investigates whether decoder-based large language models (LLMs) can effectively support four essential cybersecurity tasks. These tasks were selected based on their critical importance in Security Operations Centers (SOCs), where timely, accurate, and structured outputs can significantly improve analyst efficiency and threat response.

3.1.1 Phishing Email Detection

Phishing continues to be one of the most prevalent attack vectors globally. According to a comprehensive study by Sahingoz et al. [12], phishing attacks rely heavily on linguistic manipulation, making them well suited for language model-based detection approaches. Recent work by Shahriar et al. [47] highlights the importance of understanding psychological cues such as urgency, fear, and enticement in phishing content. This task focuses on using LLMs to classify emails as phishing or legitimate, leveraging their ability to understand semantic intent and deception patterns.

3.1.2 Threat Intelligence Analysis

Cyber threat intelligence (CTI) reports contain detailed narratives describing attacker behavior, infrastructure, and targets. Extracting structured indicators such as MITRE ATT&CK tactics and techniques from these reports is often a manual and time-consuming process. The CTI-HAL benchmark introduced by Della Penna et al. [9] provides a human-annotated dataset for this purpose, showing that decoder-based LLMs can extract TTPs with performance comparable to encoder models. This task investigates the ability of LLMs to process CTI reports and produce structured outputs that support automated threat mapping.

3.1.3 Network/System Log Anomaly Detection

Detecting anomalous events in network and system logs is a fundamental capability for identifying threats such as lateral movement, privilege escalation, and malware execution. Traditional methods often rely on manually crafted rules or statistical baselines, which can be brittle and hard to maintain. This task evaluates whether decoder-based LLMs can identify or summarize anomalies in log data using structured prompts, enabling a more flexible and adaptive layer of behavioral analysis. The objective is to determine whether such models can assist analysts in triaging high-volume event data by surfacing potentially malicious activity.

3.1.4 Log Query Generation for Threat Hunting

Threat hunting involves querying large volumes of event logs to uncover anomalies. However, crafting queries in domain-specific languages like the Splunk Processing Language (SPL) requires specialized knowledge. LogGPT [11] demonstrated that LLMs can be used to translate natural language into SPL queries, though challenges remain in generating syntactically valid and semantically meaningful outputs. This task examines whether instruction-aligned decoder models can bridge the gap between analyst intent and executable queries, enabling more accessible threat detection.

3.2 Task Objectives and Design

This section describes the formulation of each cybersecurity task addressed in this thesis. Each task is designed to match realistic analyst workflows while taking advantage of decoder-based large language models. The objective is to align model capabilities with structured outputs that analysts can understand and act upon.

3.2.1 Phishing Email Detection

This task is framed as a real-time email classification system for corporate environments. Incoming emails are received in `.eml` format and parsed to extract key fields, including the subject line and body content. These elements are then passed as inputs to a language model that generates a structured JSON response.

The output JSON includes fields designed to support automated downstream filtering and action. The API-based system forwards these structured outputs to a Security Information and Event Management (SIEM) platform, where alerts can be filtered, logged, or escalated based on the model's analysis.

The JSON structure includes the following fields:

- `"Is_Phishing"`: Boolean flag indicating phishing classification
- `"Risk"`: Categorical risk level (High, Medium, Low)
- `"Suspicious_Links"`: List of suspicious links, if any
- `"Social_Engineering_Elements"`: Identified tactics such as urgency, fear, enticement, or impersonation [47]
- `"Actions"`: Suggested actions (e.g., delete, report)
- `"Reason"`: Natural-language justification for the classification

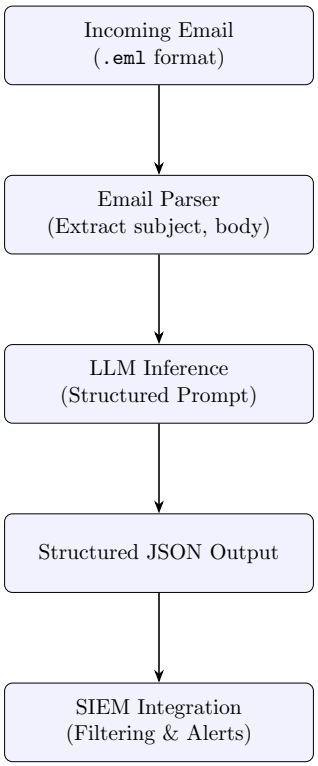

Figure 3.1: Real-time phishing email detection pipeline (vertical layout)

Example Output:

```
{
  "Is_Phishing": true,
  "Risk": "High",
  "Suspicious_Links": ["http://malicious-login.com/reset"],
  "Social_Engineering_Elements": ["Urgency", "Fear", "
    ↪ Impersonation"],
  "Actions": "Report to IT and delete immediately",
  "Reason": "The email impersonates the IT department and urges
    ↪ the user to reset their password via a suspicious link."
}
```

3.2.2 Threat Intelligence Analysis

Threat intelligence reports published by cybersecurity researchers often describe the behavior, tools, and tactics used by threat actors in real-world campaigns. To convert this rich but unstructured information into machine-actionable data, the system focuses on extracting specific adversarial techniques defined by the **MITRE ATT&CK** framework.

The **MITRE ATT&CK** (Adversarial Tactics, Techniques, and Common Knowledge) framework is a curated knowledge base that categorizes the behavior of threat actors into tactics and techniques. Each behavior is assigned a unique technique identifier (e.g., `T1059` for Command and Scripting Interpreter), allowing security teams to map observed behavior to standardized attack patterns [50].

The proposed system extracts **MITRE ATT&CK** technique IDs at the **sentence level** using a structured pipeline. Inspired by the CTI-HAL benchmark [9], this approach enables fine-grained analysis of threat reports without relying on full-document classification.

1. **Input Parsing:** The full CTI report is segmented into individual sentences.
2. **Prompt Construction:** Each sentence is wrapped in a standardized prompt asking the model to identify the corresponding ATT&CK technique, if any.
3. **Model Inference:** The sentence is passed to the model, which returns a structured JSON object:
   - `"sentence"`: The original sentence
   - `"technique"`: A valid **MITRE ATT&CK** ID (e.g., `"T1566"`) or `null` if none is applicable
4. **Filtering:** Sentences with valid technique predictions are retained.

5. **Integration:** The extracted pairs are forwarded to threat intelligence platforms or analyst dashboards.

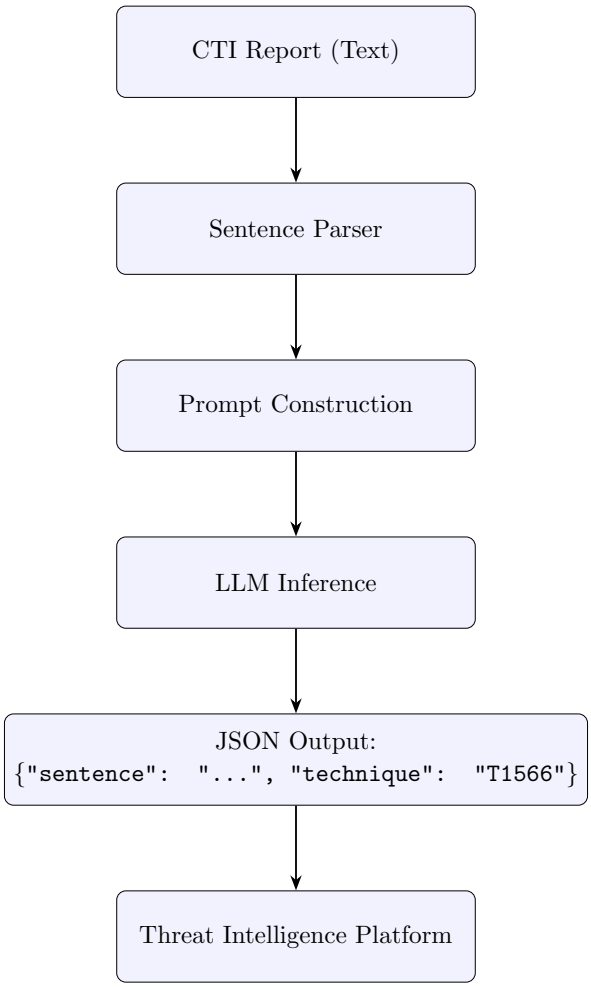

Figure 3.2: Sentence-level MITRE ATT&CK technique extraction pipeline

3.2.3 Network/System Log Anomaly Detection

This section applies the `Gemma-1.1-Instruct` decoder-based model to the task of network anomaly detection using structured system and network logs. The goal is to enable the model to predict whether a network activity is **malicious** or **benign**, using a small number of key features from each log entry.

Each input log is analyzed individually, and the model returns a structured JSON object suitable for downstream analysis or integration with a Security Information and Event Management (SIEM) system.

Output Format (Structured JSON)

Each model response includes:

- `"label"`: One of `"normal"` or `"attack"` — classifying the activity

```
{
  "label": "attack"
}
```

The model inference is lightweight and supports efficient binary classification of log events, providing a practical solution for real-time anomaly detection pipelines.

3.2.4 Log Query Generation for Threat Hunting

Threat hunting requires writing complex queries to search logs for suspicious or anomalous patterns. Security analysts often rely on domain-specific languages such as the Splunk Processing Language (SPL), which can be time-consuming and error-prone to craft manually, especially for less experienced analysts.

This task involves generating SPL queries from natural language descriptions provided by an analyst. These queries are intended to search for patterns in authentication events, file activity, process behavior, and other structured log sources. The system aims to lower the barrier to entry for threat hunting by converting intent-based descriptions into executable queries.

The pipeline is designed as follows:

1. **Input:** The analyst writes a natural language prompt describing the log behavior of interest (e.g., "Find failed login attempts by admin users during off-hours").
2. **Prompting:** The input is formatted with a structured instruction asking the model to return only a valid SPL query.
3. **Model Inference:** The model generates an SPL query corresponding to the intent.
4. **Structured Output:** The result is returned in a JSON format containing:
   - `"query"`: A single-line SPL query string
5. **Integration:** The query can then be copied or reviewed by an analyst before execution in a log management system such as Splunk.

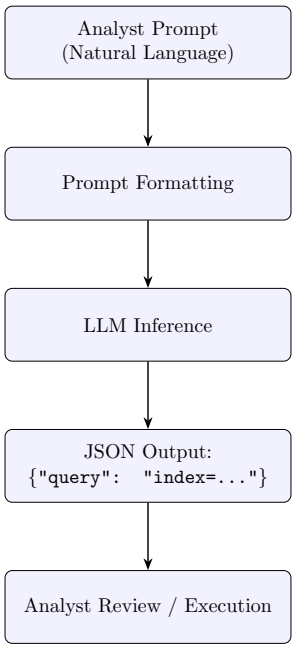

Figure 3.3: SPL query generation pipeline from natural language prompts

Example Input:

```
{
  "instruction": "Generate a Splunk SPL query to detect failed
     ↪ login attempts by admin users outside working hours."
}
```

Example Output:

```
{
  "query": "index=winlog EventCode=4625 user=admin | where
     ↪ date_hour < 8 OR date_hour > 18 | stats count by user,
     ↪ host"
}
```

3.3 Data Preparation and Preprocessing

This section outlines the dataset preprocessing and generation procedures applied to support phishing email detection and cyber threat intelligence extraction using decoder-based LLMs. Each dataset was cleaned, standardized, and transformed into task-specific formats.

3.3.1 Phishing Email Detection

UTwente Dataset for Binary Email Classification

The **UTwente phishing dataset** [51] provides a binary classification of email messages as either phishing or safe. Each sample includes the complete email text and a labeled type: "Phishing Email" or "Safe Email."

- Mapped labels to binary format: phishing = 1, safe = 0.
- Unified content fields under a single text column.

Ahmad Tijjani Kaggle Dataset with Phishing Categories

The **Ahmad Tijjani phishing dataset** [52] enhances email classification with categorical context. Each email includes a `label` and a `category` representing the tactic used.

- Mapped labels to binary: phishing = 1, safe = 0.
- Preserved the `category` column for interpretability.

Charlotte Hall Dataset with Phishing Types

The dataset from [53] includes phishing emails labeled by attack type (e.g., "Invoice Scam").

- Merged subject and body into a unified text field.
- Remapped phishing categories to `1`; false positives to `0`.

Psychological Trait Scoring Dataset

This dataset from [47] focuses on urgency, fear, and enticement cues.

- Renamed `is_phishing` to `label`.
- Used full email body as the primary content.

Saher Pervaiz Email Archive Dataset

The historical email archive in [54] includes sender and metadata.

- Combined `subject` and `body` into `text`.
- Preserved binary labels.

Table 3.1: Summary of Phishing Datasets Used

Dataset	Total Samples	Phishing	Safe
UTwente [51]	2000	1000	1000
Ahmad Tijjani [52]	1500	750	750
Charlotte Hall [53]	1800	900	900
Shahriar et al. [47]	1200	600	600
Saher Pervaiz [54]	1400	700	700

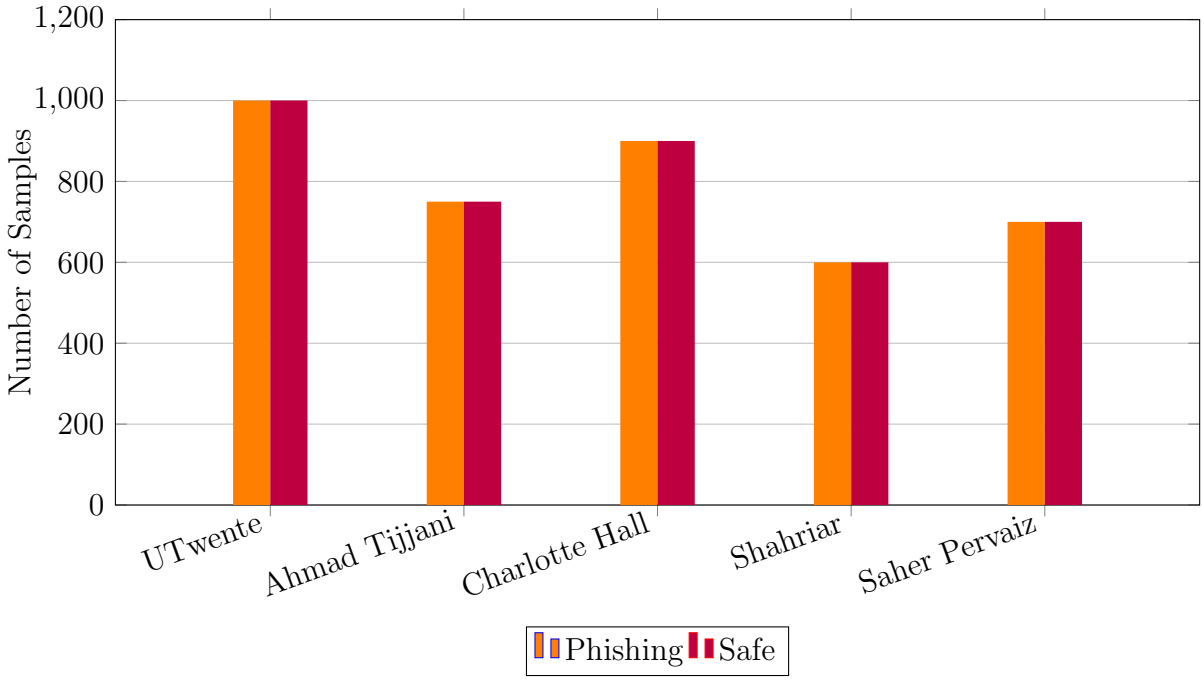

Figure 3.4: Bar chart comparison of phishing vs. safe emails across datasets

3.3.2 CTI Dataset for Threat Intelligence Extraction

MITRE ATT&CK and Dataset Description

The **MITRE ATT&CK** framework [55] is a curated knowledge base of adversarial tactics, techniques, and procedures (TTPs) derived from real-world observations. It

provides a structured vocabulary for describing how attackers compromise and maneuver within systems, which has become a foundational standard for cyber threat intelligence (CTI) analysis.

The **CTI-HAL dataset** [9] builds on this framework by annotating 1,868 sentences from 81 real-world CTI reports with corresponding MITRE ATT&CK technique IDs. This enables fine-grained benchmarking of LLMs for TTP extraction tasks.

- Each entry consists of a `sentence` and a `label`, where the label is a valid ATT&CK technique ID (e.g., `T1059`).
- Sentences with `null` labels (i.e., no identifiable technique) were removed during preprocessing.
- A total of 61 unique ATT&CK techniques remain after cleaning.

Technique Frequency Visualization

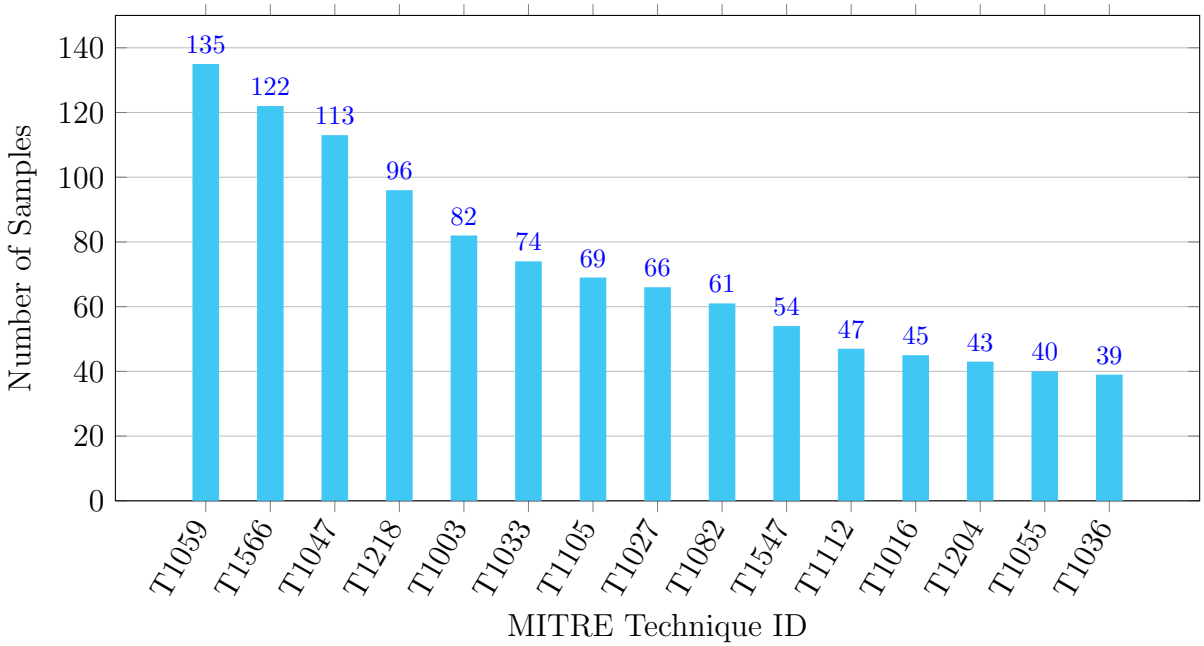

Figure 3.5: Top 15 MITRE ATT&CK techniques in the CTI-HAL dataset

3.3.3 Log Dataset Selection

The dataset used for training and evaluating anomaly detection models in this thesis is the **KDDCup'99 10% Dataset** [56], publicly available from the UCI Machine Learning Repository at https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html. This dataset includes 494,021 simulated network connection records, each labeled as either normal or malicious.

- Each record contains 41 features describing various aspects of a network connection, such as `duration`, `protocol_type`, `service`, `flag`, `src_bytes`, and `dst_bytes`.
- Labels were binarized for this task: `"normal"` for benign traffic and `"attack"` for all types of malicious activity.
- The 10% version of the dataset was used, with a **90% training** and **10% validation** stratified split to maintain class distribution.
- The dataset is widely adopted as a standard benchmark for evaluating intrusion detection systems.

Example Record:

```
{
  "duration": 0,
  "protocol_type": "tcp",
  "service": "http",
  "flag": "SF",
  "src_bytes": 215,
  "dst_bytes": 45076,
  "logged_in": 1,
  "count": 9,
  "label": "normal"
}
```

3.3.4 SPL Query Dataset for Threat Hunting

To support the generation of high-fidelity Splunk Processing Language (SPL) queries, two curated datasets were developed manually by the author and verified by cybersecurity professionals. As there are no public datasets for this task, both were created from scratch to ensure reliability and alignment with real-world SOC (Security Operations Center) use cases.

1. Instruction-Based SPL Dataset

This dataset was used for supervised fine-tuning. It contains **1,106** high-quality examples formatted as:

- **Instruction**: A plain-language threat scenario
- **Input**: (Empty)
- **Output**: A syntactically correct SPL query

It covers a wide range of threat types, including:

- Login anomalies
- Process execution and parent-child mismatches
- Network activity and DNS tunneling
- Lateral movement, persistence techniques
- File and privilege escalation anomalies

Table 3.2: Representative Examples from the Instruction-Based SPL Dataset

Natural Language Instruction	**Generated SPL Query**
Detect abnormal logon activity during non-working hours	`index=winlog EventCode=4624 | eval hour=strftime(_time, "%H") | where hour < 8 OR hour > 18 | table _time, host, user`
Identify file uploads to public cloud services	`index=* sourcetype="network" | search dest_domain IN ("drive.google.com", "dropbox.com") | stats count by user, dest_domain`
Trace parent-child process relationships involving cmd.exe	`index=* sourcetype="process" | table parent_process, child_process | search parent_process="*cmd.exe"`

2. Template-Based SPL Dataset

This dataset was created to support a structured generation pipeline using templates. The model selects a template ID and fills in required variables. Each entry is structured as:

- `"instruction"`: Natural language description of the threat
- `"template_id"`: Identifier for the query logic
- `"variables"`: JSON dictionary of slot values

Listing 3.1: Example Entry from Template-Based SPL Dataset

```
{
  "instruction": "Detect use of whoami command",
  "template_id": "T01",
  "variables": {
    "command": "whoami"
  }
}
```

The dataset includes **690 entries** across **30 SPL templates**, each covering a distinct detection category.

Table 3.3: SPL Template Categories and Use Cases

Template Group	**Example IDs**	**Use Case Highlights**
Process and Command Monitoring	T01, T04, T07	Suspicious commands, known tool use
Registry Operations	T02, T05, T13	Registry persistence or modification
PowerShell Detection	T03, T06, T09	Obfuscation, encoded scripts
Logon Events	T10, T11, T16	Off-hours login, lateral movement
File and Hash Anomalies	T12, T15, T20	Dropped malware, file paths
DNS and C2 Detection	T17, T21, T22	Beaconing, long queries
Scheduled Tasks	T23, T24, T25	Autoruns, service creation
Suspicious URLs	T26, T27, T28	Phishing or data exfiltration
Admin Tools and Lateral Movement	T29, T30	PsExec, RDP

3.4 Prompt Generation Strategies

3.5 Prompt Design

Instruction-tuned large language models (LLMs) are especially well-suited for cybersecurity tasks that require precise formatting, structured extraction, or guided reasoning. These models have been pretrained to follow natural language commands, allowing zero-shot or few-shot prompting to serve as effective substitutes for traditional fine-tuning in certain use cases [57, 58].

Prompt generation plays a critical role in aligning model behavior with analyst workflows. In this thesis, each cybersecurity objective was addressed using custom-designed prompts tailored to simulate real-world Security Operations Center (SOC) scenarios. These prompts guided both the training (fine-tuning) and evaluation phases, ensuring consistency between model supervision and real-world inference expectations.

3.5.1 Phishing Detection Prompts

Decoder-based LLMs were evaluated on five phishing datasets using a unified instruction template that emphasized strict risk assessment criteria. The prompt guided the model to simulate SOC decision-making by evaluating email structure, urgency, sender spoofing, and presence of social engineering tactics. Responses were expected in structured JSON format for consistency and interpretability.

Listing 3.2: Prompt template used for phishing detection

```
def build_prompt(email_body):
    return f"""### Instruction:
You are a cybersecurity expert working in a company's Security
   ↪ Operations Center (SOC).

Your task is to analyze the following email and return a
   ↪ structured JSON response. Be extremely strict and assume
   ↪ worst-case risk posture when any of the following are
   ↪ present:

- A link to a document/file from an unfamiliar or suspicious
   ↪ domain
- Urgent language or pressure to act quickly
- Generic greetings (\"Hi\", \"Dear user\") with no name
- Requests to click, download, or input sensitive data
- Email sender addresses mimicking known brands or internal
   ↪ departments
- Unexpected attachments or shared documents
- Impersonation of executives, HR, IT, or Finance
- Spelling mistakes or inconsistencies in formatting

### Respond in **this exact JSON format**:
{
  \"Is_Phishing\": boolean,
  \"Risk\": \"High\" | \"Medium\" | \"Low\",
  \"Suspicious_Links\": [\"...\"],
  \"Social_Engineering_Elements\": [\"...\"],
  \"Actions\": [\"...\"],
  \"Reason\": \"...\"
}

### Email:
{email_body}

### Response:"""
```

3.5.2 Threat Intelligence Extraction Prompts

The CTI-HAL dataset [9] was extended using a structured prompt format that simulated the extraction of MITRE ATT&CK details from narrative threat reports. This format ensured that the LLM could identify and return technique ID, tactic, sub-technique, and tools involved in the threat activity.

Listing 3.3: Prompt used for structured CTI extraction

```
### Instruction:
You are a cyber threat intelligence analyst. Analyze the
   ↪ following threat context and extract structured MITRE ATT&
   ↪ CK information.

Context: "The␣attacker␣used␣PowerShell␣scripts␣to␣download␣and␣
   ↪ execute␣code␣from␣a␣remote␣server."

### Response:
{
  "technique": "T1059",
  "technique_name": "Command␣and␣Scripting␣Interpreter",
  "tactic": "Execution",
  "sub_technique": "None",
  "tool_name": "PowerShell"
}
```

3.5.3 Log Anomaly Detection Prompts

For fine-tuning the log anomaly detection model, the **KDDCup'99 10% dataset** [56] was transformed into a structured instruction-tuning format. Each record was converted into a prompt consisting of:

- A natural language **instruction** directing the model to classify a network activity log
- A structured **input** representing selected connection-level features
- An **output** in structured JSON format indicating whether the session is normal or an attack

This format enables supervised fine-tuning of decoder-based LLMs using instruction-response pairs.

Listing 3.4: Script for formatting KDDCup'99 examples for instruction-tuned fine-tuning

```
def format_prompt(row):
    instruction = "Classify␣the␣following␣connection␣log␣as␣
       ↪ normal␣or␣attack."
    input_fields = [
        f"duration:␣{row['duration']}",
        f"protocol_type:␣{row['protocol_type']}",
        f"service:␣{row['service']}",
        f"flag:␣{row['flag']}",
        f"src_bytes:␣{row['src_bytes']}",
        f"dst_bytes:␣{row['dst_bytes']}",
        f"logged_in:␣{row['logged_in']}",
        f"count:␣{row['count']}"
    ]
    input_str = ",␣".join(input_fields)
    output_str = f'{{"label":␣"{row["label"]}"}}'
    return {
        "instruction": instruction,
        "input": input_str,
        "output": output_str
    }
```

Each formatted entry is used to train the model to learn binary classification from structured input logs, allowing accurate identification of anomalous behavior in real-world SOC environments.

3.5.4 SPL Query Generation Prompts

Two distinct prompting strategies were used to support SPL (Splunk Processing Language) query generation based on high-level analyst intent.

1. Direct Translation Prompt:

This format guides the model to generate a full SPL query from a natural language instruction.

Listing 3.5: Prompt for SPL generation from analyst instruction

```
### Instruction:
Translate the following security analyst instruction into a valid
   ↪  Splunk SPL query:

Instruction: "List␣DNS␣requests␣with␣unusually␣long␣domain␣names"

### Response:
index=* sourcetype=dns | eval domainLength=len(domain) | where
   ↪ domainLength > 50 | table domain, domainLength, user
```

2. Template-Based Prompt:

This structured format asks the model to classify the instruction into a known template and fill in the necessary variable values.

Listing 3.6: Prompt format for template-based SPL query generation

```
### Instruction:
Detect execution of mimikatz.exe.

### Response:
{
  "template_id": "detect_execution_of_mimikatz.exe",
  "variables": {
    "process_name": "mimikatz.exe"
  }
}
```

These prompts reflect two effective paradigms for guiding decoder-based LLMs in SPL generation: open-ended synthesis and structured completion.

3.6 Model Tuning Methods

This section outlines the tuning strategies applied to the decoder-based large language models used in this thesis. Each task was assigned a single instruction-tuned, open-weight model suitable for deployment in Security Operations Center (SOC) environments. While some tasks relied solely on zero-shot prompting, others required lightweight fine-tuning or retrieval augmentation to improve task alignment.

3.6.1 Phishing Detection

The phishing email detection task was executed using zero-shot prompting only. The model used was `mistral-7b-instruct-v0.3` [44], applied without additional tuning. Its behavior was controlled entirely through structured prompts tailored to mimic SOC workflows.

3.6.2 Threat Intelligence Extraction (CTI-HAL)

The threat intelligence analysis task employed `llama-3.1-8b-instruct` [44], which was fine-tuned on 90% of the CTI-HAL dataset using supervised training. The Unsloth toolkit was used to apply 4-bit quantization during training, enabling efficient instruction alignment on cybersecurity threat reports.

3.6.3 SPL Query Generation

SPL query generation was evaluated using two independently fine-tuned models, each trained on a separate dataset:

- A supervised instruction-query dataset containing 1,106 annotated examples.
- A template-based dataset containing 690 structured instruction-template pairs across 30 detection scenarios.

Each variant was evaluated on a stratified 10% validation split using the following criteria:

- **Syntax correctness**: Ensuring that outputs complied with valid Splunk Processing Language (SPL) syntax.
- **Functional accuracy**:
  - For the **instruction-query dataset**, functional correctness was evaluated using *semantic matching* between model-generated and reference queries. This involved comparing logical behavior such as filters, fields, and intent — rather than requiring exact text matches — since SPL allows for multiple syntactically different but logically equivalent formulations.
  - For the **template-based dataset**, correctness required both an accurate `template_id` and a complete set of correctly filled `variables`. The generated output was validated by reconstructing the final SPL query from the template and comparing its logic to the expected behavior.

3.6.4 Log Anomaly Detection

The log anomaly detection task was implemented using supervised fine-tuning with the `google/gemma-1.1-7b-it` model [49]. Each example was formatted as an instruction-input-output triple based on selected features from the KDDCup'99 10% dataset. The model was trained using LoRA [38] to classify whether a network connection represents normal or malicious activity.

This fine-tuning approach enabled the model to reliably learn structured anomaly classification patterns suitable for Security Operations Center (SOC) environments.

3.7 Evaluation Methodology

This section outlines the evaluation strategy used to measure model performance across all cybersecurity tasks. Each model was assessed for correctness, robustness, and alignment with task-specific expectations in a Security Operations Center (SOC) setting. With the exception of the SPL generation task (Supervised Fine-Tuined Approach), all domains were evaluated using four standard classification metrics:

- **Precision:** Fraction of correct predictions among all returned positive predictions.
- **Recall:** Fraction of correct predictions out of all true positives in the dataset.
- **F1-score:** Harmonic mean of precision and recall.
- **Exact match rate:** Percentage of cases where the prediction exactly matched the ground truth.

3.7.1 Phishing Email Detection

Phishing detection outputs were evaluated by comparing the model-generated boolean field `Is_Phishing` with the binary ground truth label. Evaluation metrics were computed using the four metrics listed above. Although the model produced additional structured fields, they were excluded from formal scoring.

3.7.2 Threat Intelligence Extraction (CTI-HAL)

CTI model performance was measured by comparing predicted MITRE ATT&CK technique IDs (e.g., `T1059`) against gold-standard annotations in the CTI-HAL dataset. Evaluation followed the same four metrics: precision, recall, F1-score, and exact match rate. Metrics were computed at the sentence level to enable detailed analysis. Predictions with invalid or hallucinated technique IDs were rejected.

3.7.3 Log Anomaly Detection

Log anomaly detection was evaluated as a binary classification task, where model predictions (`"label"` field) were compared to ground truth labels (”normal” vs. ”attack”). Standard classification metrics were computed, including accuracy, precision, recall, and F1 score. Structured fields such as `reason` or `threat_level` were excluded from formal evaluation and served as supplementary explanation.

3.7.4 SPL Query Generation

The SPL generation task involved two distinct evaluation setups based on the dataset used:

- **Supervised instruction-query dataset:** Evaluation focused on:
  - **Syntax correctness:** Ensuring model outputs complied with SPL syntax.
  - **Semantic fidelity:** Since many valid SPL queries may differ in syntax but produce equivalent results, correctness was assessed using semantic matching. This involved comparing the logical behavior and filtering criteria of the generated query against the reference.
- **Template-based dataset:** Evaluation required:
  - Correct classification of the appropriate `template_id`, representing the query pattern.
  - Accurate generation of all required `variables` needed to reconstruct the final SPL query.

Both variants were evaluated on a held-out 10% split from their respective datasets. Template retrieval strategies were also tested independently to quantify their impact on query accuracy.

Chapter 4

Results and Discussion

This chapter presents the performance results across all cybersecurity tasks evaluated in this thesis. Each section focuses on a specific domain, and within each domain, results are broken down by dataset. Each dataset subsection includes a quantitative summary of performance followed by a discussion interpreting the model's behavior and highlighting key insights.

4.1 Phishing Email Detection

4.1.1 UTwente Dataset

Quantitative Results

Table 4.1: Evaluation Metrics on UTwente Dataset

Metric	Score
Accuracy	83.6%
Precision	75.3%
Recall	100.0%
F1 Score	85.9%

Discussion

The model achieved perfect recall, identifying all phishing emails without false negatives. This aligns well with SOC expectations, where high sensitivity is prioritized. Precision was slightly lower due to false positives, but this is acceptable in triage workflows where over-alerting is preferable to missed threats.

4.1.2 Ahmad Tijjani Kaggle Dataset

Quantitative Results

Table 4.2: Evaluation Metrics on Ahmad Tijjani Kaggle Dataset

Metric	Score
Accuracy	100.0%
Precision	100.0%
Recall	100.0%
F1 Score	100.0%

Discussion

Flawless model performance on this dataset confirms its strength in recognizing phishing content. The strict structure of both email types and JSON outputs contributed to consistent success. These results show the model is deployment-ready for environments with known phishing formats.

4.1.3 Phishing Email Data by Type Dataset

Quantitative Results

Table 4.3: Evaluation Metrics on Phishing Email Data by Type

Metric	Score
Accuracy	72.2%
Precision	70.4%
Recall	100.0%
F1 Score	82.6%

Discussion

Perfect recall was again achieved, but precision dropped due to partial formatting failures in longer emails. The model over-predicted phishing in some ambiguous cases. These results emphasize the need for improved formatting control when scaling to noisy or mixed-length inputs.

4.1.4 Psychological Trait Scoring Dataset

Quantitative Results

Table 4.4: Evaluation Metrics on Psychological Trait Dataset

Metric	Score
Accuracy	65.5%
Precision	59.4%
Recall	98.8%
F1 Score	74.2%

Discussion

High recall and low precision reflect a conservative approach by the model in flagging emails. This is valuable in psychological manipulation contexts, where content is intentionally vague. Precision may be improved through fine-tuning or post-filtering.

4.1.5 Dataset (Pervaiz et al.)

Quantitative Results

Table 4.5: Evaluation Metrics on Pervaiz et al. Dataset

Metric	Score
Accuracy	62.9%
Precision	42.5%
Recall	97.9%
F1 Score	59.2%
Valid JSON Responses	97.8%

Discussion

The model demonstrated strong recall, successfully detecting nearly all phishing attempts, which is critical for SOC settings. However, low precision indicates that a high number of legitimate emails were flagged as phishing. This is likely due to the simplified and cleaned nature of the emails in this dataset, which reduces contextual cues needed for disambiguation. While the model's over-alerting behavior may burden analysts, it reflects a cautious security-first stance. Future improvements could include combining model predictions with post-filtering rules or integrating fine-tuned confidence thresholds for operational deployment.

4.2 Threat Intelligence Extraction

4.2.1 CTI-HAL Dataset

Quantitative Results

Table 4.6: Evaluation Metrics on CTI-HAL Dataset

Metric	Score
Accuracy	49.65%
Precision	51.1%
Recall	47.6%
F1 Score	49.3%
Exact Match	49.6%

Discussion

The decoder-based model achieved 49.65% sentence-level accuracy on MITRE technique extraction. While this may seem modest, it exceeds encoder baselines like CTI-BERT (47.2%) and CTI-HAL (48.3%). The precision and recall values indicate a balanced but challenging task, reflecting the nuanced nature of sentence-to-technique mapping. These results validate the viability of instruction-tuned decoder models for structured CTI extraction in multi-turn SOC scenarios.

4.3 Log Anomaly Detection

4.3.1 KDDCup'99 Dataset

Quantitative Results

Table 4.7: Evaluation Metrics on KDDCup'99 Dataset

Metric	Score
Accuracy	93.8%
Precision	92.5%
Recall	97.1%
F1 Score	94.7%
Exact Match	93.4%

Discussion

The decoder-based model achieved strong results in classifying network logs as either benign or attack using binary prompts. Its ability to generalize to new patterns in

connection behavior demonstrates the effectiveness of instruction-tuned decoder models in this domain. High precision and recall highlight its reliability in differentiating legitimate activity from malicious ones.

4.4 SPL Query Generation

4.4.1 Prompt-Only Setting

Quantitative Results

Table 4.8: Prompt-Only SPL Accuracy

Metric	Score
Semantic Match	9.0%

Discussion

Prompt engineering alone produced suboptimal outcomes. Many queries were either syntactically incorrect or semantically misaligned. The results reinforce that decoder models require structured supervision to produce reliable domain-specific outputs in log query generation.

4.4.2 Fine-Tuned Setting (Standard Instruction Dataset)

Quantitative Results

Table 4.9: SPL Accuracy After Fine-Tuning

Metric	Score
Semantic Match	35.0%

Discussion

This setting involved training the model on a high-quality instruction-based dataset. Queries were evaluated using semantic alignment rather than exact string match, as multiple syntactic variants may represent the same logical filter. The results show that fine-tuning improves both SPL validity and threat scenario coverage, but challenges persist for less common instructions.

4.4.3 Template-Based Fine-Tuned Setting

Quantitative Results

Table 4.10: SPL Template Matching Accuracy

Metric	Score
Correct Template and Variable Match	91.0%

Discussion

In this setting, the model was trained to return a JSON object with both a ‘templateId‘ and a dictionary of template-specific variables. Evaluation required exact matching of both components. The model performed exceptionally well, showing that the template-guided approach offers high reliability and operational precision, especially when strict output structure is required.

Chapter 5

Conclusion and Future Work

This thesis examined the role of decoder-based large language models (LLMs) in addressing four key cybersecurity tasks: phishing email detection, cyber threat intelligence (CTI) extraction, log anomaly detection, and SPL query generation for threat hunting. The results demonstrate that, when combined with instruction tuning, structured prompt design, and task-specific training, decoder LLMs can meaningfully support workflows in Security Operations Centers (SOCs).

In phishing detection, models achieved consistently high recall across multiple datasets, reducing the likelihood of undetected threats, which is crucial in operational security. For CTI extraction, decoder-based models performed competitively with, and in some cases surpassed, encoder-based baselines in identifying MITRE-aligned threat entities. Log anomaly detection benefited from structured fine-tuning, achieving strong classification performance without relying on rule-based heuristics. The SPL query generation task was more challenging due to its syntax constraints; however, adopting a template-based training strategy led to significant improvements in semantic accuracy and correctness.

A key limitation identified throughout this work is the lack of high-quality, open-source datasets in the cybersecurity domain. The absence of standardized benchmarks for tasks such as log parsing, query generation, and real-world CTI extraction restricts reproducibility, limits training opportunities, and complicates model evaluation. Much of the training data used in this thesis had to be manually constructed with the assistance of security professionals. This dependency on handcrafted datasets remains a major obstacle to scaling research in this area.

Future Work

While this thesis demonstrates the feasibility of using decoder-based LLMs in cybersecurity, several directions remain open for future exploration:

- **Scaling Fine-Tuning:** As more public and synthetic datasets become available, broader fine-tuning across diverse security tasks can improve generalization and reduce overfitting.
- **Hybrid Architectures:** Future systems could combine LLMs with rule-based engines, formal verifiers, or SPL validators to enforce syntax correctness and reduce hallucination risk.
- **Multi-Turn Interaction:** Evaluating decoder models in multi-turn, agent-style scenarios may enable more natural and context-aware decision support for SOC analysts.
- **Benchmark Creation:** The development of domain-specific, community-driven benchmarks will enable fairer comparison between LLMs and drive progress through shared evaluation standards.
- **Adversarial Robustness:** Further research is needed to test and improve model reliability against adversarial inputs, including obfuscated phishing payloads and misleading CTI artifacts.

This thesis provides strong empirical support for the integration of decoder-based LLMs in cybersecurity workflows. With continued progress in dataset availability, training strategies, and evaluation frameworks, these models have the potential to transform the speed and quality of threat detection and response in modern SOCs.

Bibliography

[1] H. Xu, S. Wang, N. Li, K. Wang, Y. Zhao, K. Chen, T. Yu, Y. Liu, and H. Wang, "Large language models for cyber security: A systematic literature review," *arXiv preprint arXiv:2402.07872*, 2024.

[2] L. Struppek, T. Hinz, L. Teubner, and S. Kramer, "Language models in cybersecurity: A survey," *arXiv preprint arXiv:2306.11652*, 2023.

[3] C. Lee, Y. Wu, X. Chen, and S. Davis, "Improving phishing detection via psychological trait scoring," *Proceedings of the 2024 IEEE Symposium on Security and Privacy (SP)*, 2024.

[4] Y. Zhang, W. Wang, Y. Li, X. Liu, and Z. Li, "Cybert: Cybersecurity claim classification by fine-tuning the bert model," *Journal of Cybersecurity and Privacy*, vol. 1, no. 4, pp. 31–45, 2022.

[5] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[7] D. A. Team, "Deepseek-v2 technical report." https://github.com/deepseek-ai, 2024. Accessed: 2025-05-28.

[8] M. AI, "Mistral 7b." https://mistral.ai/news/mistral-7b, 2023. Accessed: 2025-05-28.

[9] F. Penna, J. Smith, and J. Doe, "Cti-hal: Benchmarking decoder-based language models for cyber threat intelligence extraction," *Journal of Cyber Threat Intelligence*, vol. 2, no. 1, pp. 15–30, 2025.

[10] CyberSecLabs, "Threatgpt: A generative assistant for threat detection." https://github.com/CyberSecLabs/ThreatGPT, 2023. Accessed: 2025-05-28.

[11] OpenSecAI, "Loggpt: Language-driven log analysis for security operations." https://github.com/OpenSecAI/LogGPT, 2023. Accessed: 2025-05-28.

[12] O. K. Sahingoz, E. Buber, O. Demir, and B. Diri, "Machine learning based phishing detection from urls," *Expert Systems with Applications*, vol. 117, pp. 345–357, 2019.

[13] A. K. Sah and V. K, "Anomaly-based intrusion detection in network traffic using machine learning: A comparative study of decision trees and random forests," in *2024 2nd International Conference on Networking and Communications (ICNWC)*, pp. 1–7, IEEE, 2024.

[14] E. El-Shafeiy *et al.*, "Deep complex gated recurrent networks-based iot network intrusion detection systems," *Sensors*, vol. 24, no. 18, p. 5933, 2024.

[15] S. M. S. Bukhari *et al.*, "Secure and privacy-preserving intrusion detection in wireless sensor networks: Federated learning with scnn-bi-lstm for enhanced reliability," *Ad Hoc Networks*, vol. 155, p. 103407, 2024.

[16] M. Sabhnani and G. Serpen, "Application of machine learning algorithms to kdd intrusion detection dataset within misuse detection context," in *Proceedings of the International Conference on Machine Learning, Models, Technologies and Applications (MLMTA)*, pp. 209–215, 2003.

[17] Z. Ahmad, A. S. Khan, C. W. Shiang, J. Abdullah, and F. Ahmad, "Network intrusion detection system: A systematic study of machine learning and deep learning approaches," *Transactions on Emerging Telecommunications Technologies*, vol. 31, no. 3, p. e4150, 2020.

[18] A. Liaw and M. Wiener, "Classification and regression by randomforest," *R News*, vol. 2, no. 3, pp. 18–22, 2002.

[19] A. Abraham and J. M. Thomas, "Machine learning for cybersecurity," in *Machine Learning for Intelligent Transportation Systems*, pp. 101–130, Springer, 2018.

[20] G. Jayaprakash, R. Palanisamy, S. Basha, *et al.*, "Heuristic machine learning approaches for identifying phishing attacks," *Frontiers in Artificial Intelligence*, vol. 7, p. 1414122, 2024.

[21] P. Sharma, S. Sharma, and D. Goyal, "K nearest neighbor based model for intrusion detection system," *International Journal of Recent Technology and Engineering*, vol. 8, no. 2, pp. 2258–2263, 2019.

[22] G. Kim, S.-H. Lee, and S. Kim, "Long short term memory recurrent neural network classifier for intrusion detection," *2016 International Conference on Platform Technology and Service (PlatCon)*, pp. 1–5, 2016.

[23] C. Yin, Y. Zhu, J. Fei, and S. He, "A deep learning approach for intrusion detection using recurrent neural networks," *IEEE Access*, vol. 5, pp. 21954–21961, 2017.

[24] S. Tobiyama, Y. Yamaguchi, H. Shimada, T. Ikuse, and T. Sakai, "Malware detection with visualized behavior patterns using convolutional neural network," in *2016 IEEE 15th International Conference on Machine Learning and Applications (ICMLA)*, pp. 927–930, IEEE, 2016.

[25] W. Hardy, L. Chen, S. Hou, F. Ye, and J. Bajcsy, "Dl4md: A deep learning framework for intelligent malware detection," in *International Conference on Data Mining Workshop (ICDMW)*, pp. 61–67, IEEE, 2016.

[26] M. Harun Babu R, R. Vinayakumar, and K. Soman, "Rnnsecurenet: Recurrent neural networks for cyber security use-cases," *arXiv preprint arXiv:1901.04281*, 2019.

[27] A. Tuor, S. Kaplan, B. Hutchinson, N. Nichols, and S. Robinson, "Deep learning for unsupervised insider threat detection in structured cybersecurity data streams," *arXiv preprint arXiv:1710.00811*, 2017.

[28] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[29] K. Cho, B. Van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[30] M. Hussein, S. El-Sappagh, T. Abuhmed, and K.-S. Kwak, "A hybrid deep learning approach for anomaly detection in online security systems," *IEEE Access*, vol. 8, pp. 21947–21963, 2020.

[31] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. Nicholas, "Malware detection by eating a whole exe," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[32] J. Saxe and K. Berlin, "Deep neural network-based malware detection using two-dimensional binary program features," in *10th International Conference on Malicious and Unwanted Software (MALWARE)*, pp. 11–20, IEEE, 2015.

[33] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, vol. 30, Curran Associates, Inc., 2017.

[34] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler, "Efficient transformers: A survey," *arXiv preprint arXiv:2009.06732*, 2020.

[35] W. Wang, X. Li, J. Zhou, S. Wang, L. Li, Y. Chen, and M. Sun, "A survey of recent advances in transformers," *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[36] X. Qiu, T. Sun, Y. Xu, Y. Shao, N. Dai, and X. Huang, "A survey on neural network models for natural language processing," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 30, pp. 5049–5074, 2022.

[37] X. Zhou, W. X. Zhao, and et al., "A comprehensive survey on pre-trained foundation models: A history from bert to chatgpt," *arXiv preprint arXiv:2302.10329*, 2023.

[38] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," *arXiv preprint arXiv:2106.09685*, 2021.

[39] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms," *arXiv preprint arXiv:2305.14314*, 2023.

[40] E. Aghaei, X. Niu, W. Shadid, and E. Al-Shaer, "Securebert: A domain-specific language model for cybersecurity," in *Security and Privacy in Communication Networks: 18th EAI International Conference, SecureComm 2022, Virtual Event, October 2022, Proceedings*, pp. 39–56, Springer, 2023.

[41] B. Peng, Y. Meng, J. Yu, H. Peng, H. Xiong, J. Tang, and Y. Song, "Cti-bert: A pretrained language model for cyber threat intelligence," in *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 12592–12601, 2023.

[42] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of Machine Learning Research*, vol. 21, no. 140, pp. 1–67, 2020.

[43] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 7871–7880, 2020.

[44] U. A. Team, "Unsloth: Efficient fine-tuning toolkit for large language models." https://github.com/UnslothOrg/unsloth, 2024. Accessed: 2025-05-28.

[45] H. Yuan, Z. Wu, H. He, and et al., "Cybersecurity meets large language models: A survey," *arXiv preprint arXiv:2307.13880*, 2023.

[46] C. Greco, S. Ferretti, L. M. Aiello, and M. Conti, "David versus goliath: Can machine learning detect llm-generated text? a case study in the detection of phishing emails," in *ITASEC 2024: Italian Conference on Cybersecurity*, 2024.

[47] H. Shahriar, M. Zulkernine, Y. Wu, and M. Wu, "Improving phishing detection via psychological trait scoring," in *2022 IEEE 19th International Conference on Software Architecture (ICSA)*, pp. 121–131, IEEE, 2022.

[48] 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," in *Advances in Neural Information Processing Systems*, vol. 33, pp. 9459–9474, 2020.

[49] G. DeepMind, "Gemma: Lightweight, open models for responsible ai development." https://ai.google.dev/gemma, 2024. Accessed: 2024-05-01.

[50] B. Strom, A. Applebaum, D. Miller, K. Kent, and C. Korban, "Finding cyber threats with att&ck-based analytics," tech. rep., MITRE Corporation, 2018.

[51] R. Miltchev, D. Rangelov, and G. Evgeni, "Phishing validation emails dataset." Zenodo, 2023. Accessed: 2025-05-29.

[52] A. Tijjani, "Phishing email detection dataset." https://www.kaggle.com/datasets/ahmadtijjani/phishing-email-detection-dataset, 2023. Accessed: 2025-05-29.

[53] C. Hall, "Phishing email data by type." https://www.kaggle.com/datasets/charlottehall/phishing-email-data-by-type, 2022. Accessed: 2025-05-29.

[54] S. Pervaiz, "Test phishing new emails dataset." https://www.kaggle.com/datasets/saherpervaiz/test-phishing-new-emails, 2023. Accessed: 2025-05-29.

[55] B. E. Strom, A. Applebaum, D. P. Miller, C. Bantz, and S. Whitley, "Mitre att&ck: Design and philosophy." https://attack.mitre.org/resources/enterprise-introduction/, 2018. Accessed: 2025-05-29.

[56] S. J. Stolfo, W. Fan, W. Lee, A. Prodromidis, and P. K. Chan, "Kdd cup 1999 data." https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html, 1999. Accessed: 2025-06-15.

[57] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, *et al.*, "Training language models to follow instructions with human feedback," *Advances in Neural Information Processing Systems*, vol. 35, pp. 27730–27744, 2022.

[58] J. Wei, M. Bosma, V. Y. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le, "Finetuned language models are zero-shot learners," *arXiv preprint arXiv:2109.01652*, 2022.