Concept Drifts, Detector Evolves: Malware Detection Made Easy with *LLMalware*

Zijing Ma, Leming Shen, Xinyu Huang, Kai Zhou, Yuanqing Zheng*

The Hong Kong Polytechnic University, Hong Kong, China
{zijing.ma, leming.shen, unixy-xinyu.huang}@connect.polyu.hk,
{kai.zhou, yqzheng}@polyu.edu.hk

Abstract. Android malware pose severe threats to the mobile application ecosystem. Although well-trained malware detection models can initially achieve satisfactory performance, they struggle with unseen Android apps constantly emerging over time. Previous methods mitigate this problem by frequently collecting and labeling new apps to update the aging models. This process, however, necessitates highly specialized domain knowledge and incurs prohibitive retraining overhead. To address this problem, this paper presents *LLMalware*, which integrates three novel technical components. First, we propose *full-spectrum automated feature extraction*, which automatically extracts diverse malware features from various detection models to fully exploit their orthogonal malware detection capabilities. Next, we develop *cohesive feature fusion*, which constructively combines these features to build effective representations for malware detection. Lastly, we devise *agile knowledge update* engine to enable robust and efficient online malware detection via an LLM-based automated agent and a dynamically maintained malware knowledge base. Extensive experiments demonstrate *LLMalware* can significantly mitigate concept drift with an average improvement of approximately 10% in F1-score over state-of-the-art baselines.

Keywords: malware detection · concept drift · large language model

1 Introduction

Android malware pose severe threats to users with privacy leakage, data theft, and even financial losses [1–7]. As reported [8], nearly 35 million Android malware have been detected by June 2024, with a growth rate of nearly 100,000 per month. Existing malware detection methods train various machine learning models to extract and analyze features derived from static analysis [9–11] and dynamic analysis [12–14]. Manually analyzing and labeling such large amounts of malware requires substantial human efforts and incurs prohibitive costs.

As malware keep evolving to evade detection, current detection models quickly become obsolete with significant performance degradation over time due to the *concept drift* problem [15, 16]. As illustrated in Fig. 1, concept drift arises

* Corresponding author.

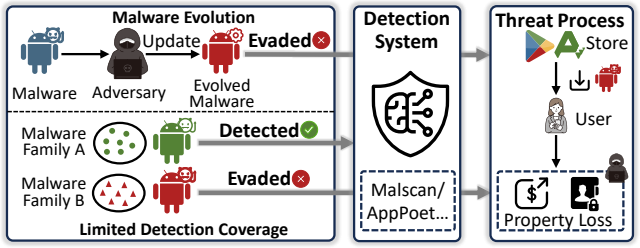


Fig. 1: Limitations of existing malware detection methods. The evolved and unseen malware can evade detection systems and appear in app stores, posing significant risks to users.

since malware adversarially mutate API usages and malicious behaviors [17] to evade detectors. Moreover, an individual detector may be effective in detecting certain types of malware but struggle with others, as detectors tend to focus on distinct and narrow characteristics of known malware. Thus, relying solely on a single detector suffers from the risk of overlooking malware with diverse unseen and emerging patterns. Prior solutions attempt to mitigate the concept drift problem via continual learning [18–20]. These solutions constantly retrain models which incurs high overhead and inherently suffer the limitations of each individual detector.

One possible solution is to fuse static and dynamic features from diverse detection methods to fully exploit the complementary detection capabilities. The key insight is that fusion expands the multi-dimensional feature space. Evading detection in this expanded feature space becomes drastically difficult, as malware evasion requires evading multiple orthogonal feature representations simultaneously. This feature fusion approach, however, requires substantial human efforts and domain expertise: First, reports generated by dynamic analysis typically contain abundant information about apps' routine operations, with only a small portion capturing essential malicious behaviors. Identifying and extracting relevant behaviors from these reports require highly specialized domain expertise, making such a detection process prohibitive and impractical for large-scale real-world deployment [21, 22]. Second, the evolving nature and diversity of malware introduce unforeseen behavioral mutations, necessitating adequate and timely adaptation of detection models to stay effective [17, 19, 23].

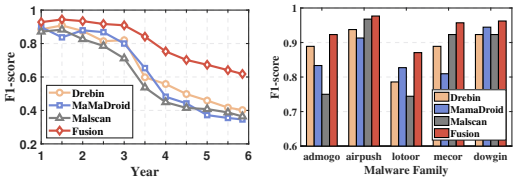
In this paper, we present *LLMalware*, an LLM-powered malware detection framework that leverages the latest advances of LLMs to enhance malware detection. To reduce the human effort involved in feature extraction and fusion, *LLMalware* automates report interpretation and essential information extraction with LLMs. This approach substantially improves the efficiency of report analysis. Moreover, *LLMalware* develops a new fusion network to fuse the features extracted from diverse detection methods and thereby mitigate the concept drift problem. To support adequate and timely adaptation, *LLMalware* further transforms the conventional detection process to on-demand and agile knowledge updates of the malware knowledge base, which facilitates the continuous evolution of detection models at low costs.

Although leveraging powerful LLMs to enhance report analysis and malware detection shows significant potential, we face a few technical challenges. ① *How can we instruct the LLMs to interpret the analysis reports and extract behavior-related information effectively?* Although LLMs have general world knowledge, they struggle with comprehending highly specialized domain-specific detection reports. As shown in Fig. 3, directly inputting an analysis report to LLMs generates general responses and cannot extract malware information. To tackle this challenge, we instruct LLMs with in-context learning enhanced Chain-of-Thought (CoT) prompts [24] to identify highly relevant behavior information [25,26]. ② *How can we align and complement the features extracted by various detection models when performing fusion?* Since detection models extract diverse features with distinct semantics that are not inherently aligned within a unified feature space, existing works [27, 28] relying on direct concatenation suffer from feature bias issues during feature fusion. This approach results in substantial performance degradation [29]. To address this problem, *LLMalware* deploys a novel autoencoder that projects and aligns these disparate features into a unified latent space, enabling the features to complement each other effectively. ③ *How can we accurately retrieve the most relevant information from the knowledge base to detect apps?* Relying solely on simple similarity metrics such as cosine distance can potentially lead to misclassification. To address this issue, we propose a multi-metric search strategy that simultaneously employs three different metrics during the retrieval process. A voting mechanism is further applied to determine the final retrieved information.

We design and implement *LLMalware* and carry out extensive evaluations with more than 130,000 Android apps collected over 6 years. We observe the evolution of Android malware and substantial concept drift during such a long period. Results show that while other baselines fail to keep up with the evolution of malware, *LLMalware* can sustain high detection rates against the latest malware with substantial mutations. Specifically, *LLMalware* achieves an 8.3% higher F1-score and a much lower aging rate, with cumulative gaps reduced by 210% compared to baselines.

The contributions of our work are summarized as follows:

- We propose *LLMalware*, an LLM-powered automation framework to enhance the robustness and efficiency of malware detection against concept drift.
- We carefully design a novel autoencoder-based fusion network to align the heterogeneous features and generate robust malware features, fully utilizing the superior detection abilities of various malware detectors.
- We develop an LLM agent that can automatically interpret and process dynamic analysis reports and continuously update the knowledge base at a low cost.
- Extensive experiments demonstrate the robustness and efficiency of *LLMalware*, which significantly outperforms the baselines and effectively mitigates the impact of concept drift over time.



(a) (b)

Fig. 2: Preliminary experiments: (a) The performance of detection methods over time; (b) The performance of detection methods across malware families.

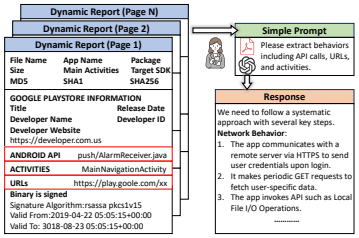


Fig. 3: Illustration of dynamic reports and general responses due to the limitation of naive prompting.

2 Background & Motivation

2.1 Concept Drift in Malware Detection

Concept drift. Machine learning models typically assume that the training and test sets are independent and identically distributed (iid) [16]. In the context of malware detection, this assumption does not hold: 1) Evolving malware exhibit unseen behavioral patterns that deviate from the ones captured in the training data. 2) Trained on a specialized malware dataset, a single detection model could lack generalizability and cannot effectively handle malware from other classes that fall outside its scope.

Impact of concept drift. We conduct experiments and evaluate the impact of concept drift. We faithfully reproduce three representative machine learning-based detection methods, including Drebin [9], MamaDroid [10], and Malscan [11]. These methods extract distinct features from the apps and employ different machine learning models for detection. To assess their performance against concept drift, we collect over 130,000 Android apps released between 2020 (Year 0) and 2025 (Year 5) (§ 4.2). Then, we train the baselines with the apps released in Year 0 to classify the new apps released in the following 5 years (*i.e.*, from Year 1 to Year 5). As shown in Fig. 2(a), the F1-score of all the detection methods gradually drops by about 0.1 in late Year 1. Moreover, the F1-score fluctuates and decreases dramatically after Year 2, with an average drop of 0.3. These results reveal that as new malware evolves, the detection models suffer from performance degradation over time. Furthermore, as shown in Fig. 2(b), there are significant variations across different malware families among the three detectors. For example, Drebin performs better on the Admogo malware family, while MamaDroid excels in Lotoor and Dowgin malware detection. Such diverse detection capabilities across different malware families can result in the misdetection of certain types of malware by individual detectors.

Insights & Initial Attempts. Based on the above observations, our key insight is to fully exploit the strengths of different detection models. As the above initial experiment reveals, malware outside the detection scope of a model may

be covered by another model focusing on distinct feature patterns. Therefore, by integrating heterogeneous detectors, we capture complementary behavioral features, enhancing the detection robustness against diverse unseen malware. As demonstrated in Fig. 2(a), even by orchestrating a simple ensemble of the three models (*e.g.*, direct feature concatenation), one may mitigate the performance degradation caused by concept drift to some extent. Moreover, as shown in Fig. 2(b), feature fusion can improve detection performance across different malware families by approximately 4% compared to the best individual models.

2.2 LLM-Powered Automated Detection

Large Language Models. Large Language Models (LLMs) (*e.g.*, ChatGPT [30] and Llama [31]) represent a significant advancement in language models. Trained on extensive corpora datasets, LLMs demonstrate remarkable capabilities in world knowledge comprehension and reasoning [32–37].

Report Interpretation. As mentioned in § 1, report interpretation becomes a bottleneck for efficient malware analysis. Analyzing such specialized reports requires substantial human effort and domain expertise. As shown in Fig. 3, these analysis reports are typically lengthy, spanning dozens of pages. While some approaches apply lightweight NLP models [31,38] to automate this process [39], they are constrained by limited context windows (*e.g.*, 512 tokens) and weak reasoning capabilities, inadequate for accurately capturing critical behavioral semantics (§ 5.4). Moreover, the report contains a mix of unstructured natural language and heterogeneous, redundant information. This renders conventional methods, such as regular expressions [40], ineffective for accurately extracting malware behaviors. In contrast, equipped with much larger context windows and extensive world knowledge, LLMs can interpret the semantic meaning of these reports, freeing them from the limitations of unstructured and redundant text. Therefore, we employ an LLM to automatically analyze the reports and extract relevant app behavior information from the reports, improving efficiency and relieving developers of the monotonous tasks involved in malware analysis.

Domain Knowledge Adaptation. Vast general knowledge of LLMs enables them to perform a wide range of downstream tasks. However, updating LLMs' internal knowledge is costly due to their immense model size, hindering their ability to incorporate the latest knowledge and adapt to new downstream tasks (*e.g.*, malware detection). Retrieval-Augmented Generation (RAG) tackles this problem by constructing a knowledge base and proactively fetching the latest relevant information during LLM inference [41]. Armed with the knowledge base, LLMs can generate accurate and relevant responses without costly retraining.

The rapid evolution of Android malware urgently necessitates detection models that can be updated timely with minimal overhead, which aligns well with the capabilities of RAG. Motivated by this, *LLMalware* leverages the RAG technique to transform conventional malware detection into an adaptive knowledge evolution process and enable agile updates of the malware knowledge base. By automatically and dynamically maintaining such a knowledge base, *LLMalware*

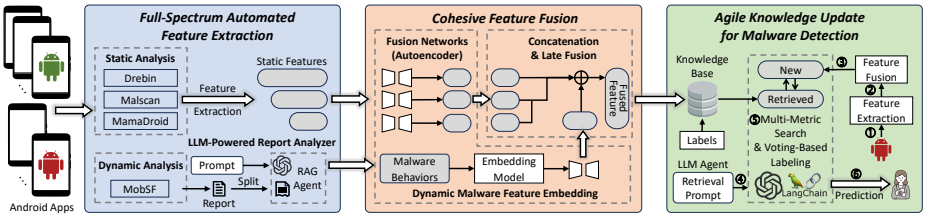


Fig. 4: The overall workflow and the key technical components of *LLMalware*.

can efficiently keep up with emerging malware without prohibitive retraining from scratch or fine-tuning. This approach ensures that the LLM continuously acquires knowledge about unseen malware and thereby enhances its robustness. However, a key challenge for knowledge retrieval is that previous retrieval methods rely on a single similarity metric, which is vulnerable to the quality of representations. For example, features with high cosine similarity may still correspond to large Euclidean distances, leading to misclassification.

To overcome this challenge, we propose a *multi-metric search* strategy to jointly evaluate multiple similarity metrics for robust retrieval, as each metric captures distinct geometric aspects of the features and their combination mitigates the retrieval bias introduced by a single similarity metric. A voting mechanism is then applied to aggregate the final results. This strategy mitigates the impact on representation quality, thereby improving retrieval performance.

3 System Design

3.1 *LLMalware* Overview

In this section, we present *LLMalware*, an LLM-powered robust and efficient Android malware detection framework. As shown in Fig. 4, to fully exploit the diverse detection capabilities, in *full-spectrum automated feature extraction*, we first extract malware features from multiple static detection methods. Meanwhile, we run the app in a sandbox and generate a report based on its runtime behaviors. Then, *LLMalware* uses an LLM-powered report analyzer to automatically extract behavior-related information. The extracted information is further transformed into text embeddings and input into our *cohesive feature fusion* module. This module aligns these heterogeneous features and generate robust fused features. Finally, our *agile knowledge update* component enriches a knowledge base by storing fused features alongside their corresponding labels. Given a new app, *LLMalware* extracts fused features using an automated LLM agent (①-④). The agent employs a multi-metric retrieval strategy to calculate the feature similarity between the new app and those in the knowledge base (⑤) for malware classification (⑥).

3.2 Agile Knowledge Update

The goal of agile knowledge update is to detect malware using the features of the apps and output the predicted label, while minimizing the cost of updating

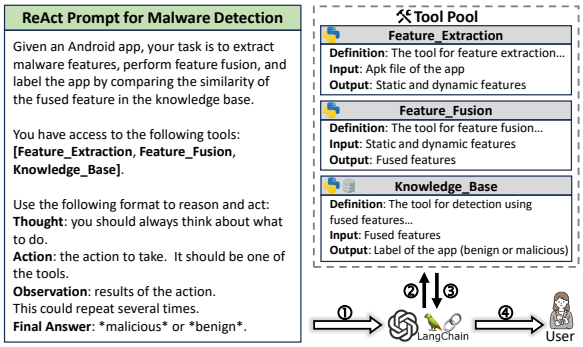


Fig. 5: The workflow of the LLM-based agent for *agile malware detection*.

the detection models. In *LLMalware*, we propose a novel RAG-based malware detection framework that supports agile knowledge updates at low costs. Specifically, we first develop a knowledge base to store the apps' features and labels, serving as external memory. Then, we build an LLM agent to automatically perform similarity-driven inference by querying the knowledge base through a multi-metric search strategy, thereby alleviating the burden of script development and maintenance for developers, especially when tools are updated or their interfaces change. This framework transforms conventional malware detection into an on-demand knowledge update process, where the LLM adapts through reasoning instead of static classification. As a result, *LLMalware* can progressively incorporate new malware features, supporting robust detection for unseen malware without extra model retraining.

Malware Knowledge Base. *LLMalware* first constructs a knowledge base to store apps' features. The base initially comprises features and labels from known apps (*e.g.*, apps in the training set), serving as prior knowledge, and can be dynamically updated with features and labels of new and available apps, where labels are generated either manually by experts or through anti-malware software. Thereafter, the base enables the LLM agent to retrieve the most semantically similar features and labels as output for further detection. However, a key challenge for robust retrieval lies in the varying quality of feature representations, which affects the accuracy of retrieving knowledge from the base.

LLM-Based Agent for Malware Detection. We build an LLM-based agent to automatically detect malware using knowledge retrieved from the base. Given a test app, the agent produces a binary prediction, either benign or malicious. Instead of relying on rigid and hard-coded execution scripts [42], the agent decouples the high-level detection process from low-level tool implementation to perform adaptive reasoning and facilitate future extensibility.

Fig. 5 illustrates the workflow of the agent. We first encapsulate the three components of *LLMalware* (*i.e.*, *full-spectrum automated feature extraction*, *cohesive feature fusion*, and *malware knowledge base*) into three callable Python functions (named *feature_extraction*, *feature_fusion*, and *knowledge_base*). These functions are associated with a functional definition that clarifies their purpose

and expected behavior. These callable tools (*i.e.*, functions), along with their definitions, are then explicitly provided to the LLM agent in a prompt, ensuring that the agent can understand the specific role of tools and invoke them via function calls.

Subsequently, we employ the ReAct framework [43] to decompose the malware detection task into a structured thought-action-observation cycle. Rather than following a static and hard-coded sequence, the agent dynamically evaluates the detection progress. Specifically, the agent first analyzes the task requirements and identifies a lack of raw features (**Thought**), so it invokes the *feature_extraction* tool (**Action**). Upon receiving the app's raw features from the tool (**Observation**), the agent validates the feature availability and reasons that heterogeneous features must be aligned before retrieval. Consequently, it autonomously selects the *feature_fusion* tool to generate robust fused features. Finally, the agent queries the *knowledge_base* tool to retrieve the final label and classify the app. The cycle ends once the task is completed (*i.e.*, the app is classified). This automated process enables the agent to dynamically select the most appropriate tool based on the context, capable of adapting to new detection tools in future updates. However, we find that the final labels are vulnerable to the retrieval metrics, which poses a critical challenge for robust malware detection.

Challenges. When the agent retrieves knowledge from the base, existing retrieval methods rely on simple similarity metrics (*e.g.*, cosine distance). However, these methods are vulnerable to the quality of the feature representations and fail under geometric misalignment [41], resulting in inaccurate retrieval (shown in Fig. 14). For example, features are typically not normalized to retain magnitude information. Therefore, two features may exhibit high cosine similarity but a large Euclidean distance, as two features are far apart in feature space with a small cosine angle. To overcome this challenge, we propose a *multi-metric search* strategy that simultaneously evaluates three distinct metrics for retrieval, including cosine similarity, Manhattan distance, and Euclidean distance, capturing different aspects of the retrieved representations. A majority voting mechanism is then employed to aggregate consistent retrieved information across metrics.

Multi-Metric Search Strategy and Voting Mechanism. We employ a *multi-metric search* strategy to retrieve semantically similar features from the knowledge base. Specifically, the agent queries the base for three labeled samples most similar to the features of the test sample, using cosine similarity, Manhattan distance, and Euclidean distance. These metrics capture complementary geometric properties of the feature space. For instance, cosine similarity focuses on directionality to capture the similarity of behavior patterns, while Euclidean distance provides a global assessment of overall similarity. Therefore, it can mitigate the limitations of relying on any single metric (*e.g.*, K-nearest neighbor (KNN) algorithms [44]). Theoretically, the strategy has a linear time complexity of $O(N \cdot d)$, where N is the number of stored features in the base and d is the feature dimension. This complexity ensures efficient searching. A majority voting mechanism is then applied to determine the final label. Specifically, the label that appears at least twice is chosen as the final label. The final label is

subsequently returned to the agent. Finally, the test sample is assigned the same label as the final label.

With the strategy, we can mitigate the influence of representation quality during the retrieval process, retrieving more accurate knowledge.

Knowledge Base Maintenance. To dynamically update the knowledge base, we first select the top-K apps with the smallest similarity to the features in the knowledge base, along with their features. We then add them to the base at regular intervals (*e.g.*, monthly), as their features and labels can provide new behavior patterns. These selected apps are manually annotated by experts or an anti-malware solution (*e.g.*, VirusTotal [19]). Following prior studies [19], we set K to 50, which empirically balances adaptation efficiency and detection accuracy. This approach, unlike previous works [19] that require high retraining costs, ensures the base can be continuously updated with new malware knowledge at low costs, as we only update the external base but not the immense parameters of the detection models.

In the following sections, we illustrate how *LLMalware* automatically extracts informative features using LLMs and consolidates the features through a fusion network to generate robust representations stored in the knowledge base.

3.3 Full-Spectrum Automated Feature Extraction

Detection Methods Given an Android app, *LLMalware* first applies multiple detection methods to extract features that focus on different aspects of malware. Specifically, we adopt four typical detection methods based on static analysis (*i.e.*, Drebin [9], MamaDroid [10], and Malscan [11]) and dynamic analysis (*i.e.*, MobSF [45]). These methods are not selected to favor our design, as they are standard and commonly used detectors. The details of detection methods are listed in the appendix (Table 3). Specifically, Drebin uses static features such as requested permissions [46], while MamaDroid and Malscan utilize the function call graph (FCG) to model the apps' API call behaviors. In contrast, MobSF captures runtime behaviors that may be overlooked by static analysis. The rationale behind choosing these methods is to ensure that each feature set emphasizes distinct aspects, providing a comprehensive representation of malware behaviors across different families. By integrating the features of the four methods, we can fully leverage the strengths of detectors and generate more robust fused features with a broader detection coverage, thereby mitigating the impact of concept drift. Note that *LLMalware* flexibly integrates new detection methods to enhance the overall performance and robustness, without being constrained to any specific approach (§ 5.3). This enables *LLMalware* to be open and augmentable with a wide range of detection methods.

LLM-Powered Automatic Report Analyzer We leverage a dynamic analysis tool (*i.e.*, MobSF) to capture the runtime behaviors of the apps in a sandbox. During runtime, we record app behaviors and generate an analysis report. We exploit the powerful language comprehension and processing capability of LLMs

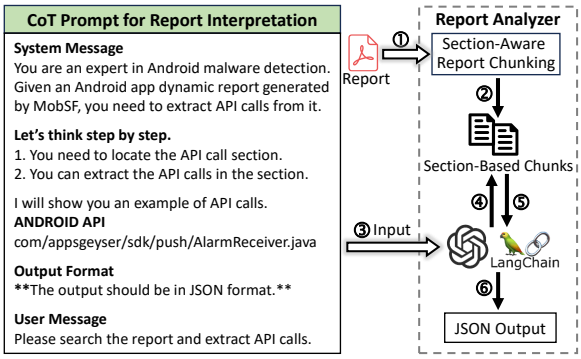


Fig. 6: The workflow of LLM-powered automatic report analyzer.

to extract useful information related to app behaviors from the report. The information is then converted into embeddings using an embedding model.

Dynamic Analysis. MobSF supports high Android API levels and provides RESTful APIs for batch processing. This makes MobSF ideal for efficiently analyzing large volumes of apps. After completing the dynamic analysis, MobSF returns an analysis report detailing the runtime information of the apps. We then extract runtime information from the report, including API calls, URLs, TLS/SSL configuration, and activities, as they provide insights about the app's potentially malicious behaviors [13]. By analyzing and extracting the dynamic behaviors of the apps, we can improve feature granularity where static analysis falls short, enhancing the robustness of malware detection. Therefore, we employ LLMs to process the reports automatically. However, we find it challenging for LLMs to directly interpret the reports due to the excessive length and specialized domain content.

Challenges. LLMs exhibit limitations in interpreting the reports due to the highly specialized and nuanced nature of analysis reports, leading to general or imprecise responses. Moreover, the lengthy reports dilute LLM's attention and exceed the token limit of LLMs, preventing them from accessing subsequent useful information. As shown in the right part of Fig. 3, when we directly input the entire report into the LLM and prompt it to extract the app's behavior information, it only produces a general response with limited information for malware detection. To tackle these challenges, we propose *section-aware report chunking* to narrow down the context input to the LLM, and *CoT-based report interpretation* to instruct the LLMs to interpret the report step by step.

Section-Aware Report Chunking. Before feeding the reports to the LLM, we intuitively split them into multiple chunks based on the section title to fit within the LLM's context window. Specifically, each chunk contains the entire contents of a section in the reports. For example, "ANDROID API" in Fig. 3 contains API calls at runtime. The resulting chunks are then input to the LLM for interpretation.

CoT-Based Report Interpretation. We leverage an LLM to interpret the chunked reports and extract behavior-related information using CoT prompting

and one-shot in-context learning. The workflow is illustrated in Fig. 6. After the report is segmented, the LLM interprets the chunks and extracts information using a CoT-based prompt. Specifically, the prompt includes four parts, *i.e.*, role definition, task instructions, an example of extracted content, and output format. We take the prompt for extracting API calls in Fig. 6 as an example, and other information like URLs undergoes the same process. In the role definition section, we prompt the LLM to act as a malware detection expert since assigning a specific role to an LLM can significantly improve its performance on downstream tasks due to the elicitation of requirements and activation of domain-specific knowledge [25]. We then decompose the task into two steps to guide the LLM in executing the process step by step. First, the LLM locates the API call section from the chunked reports and then extracts the API calls from the identified section. Second, we provide an example of API calls to help the LLM better comprehend the semantic meaning of API calls. Through this one-shot in-context learning, the LLM can extract relevant information from the corresponding sections. Finally, we specify the JSON format as the output format at the end of the prompt to maintain consistent output formatting. Through careful prompt engineering, the LLM can efficiently extract behavior-related text and reduce human effort.

Behavior-to-Embedding Transformation. After obtaining the behavior-related text, *LLMalware* leverages an OpenAI embedding model [47] to transform them into embeddings. These embedding vectors contain rich semantic information from the reports, revealing distinguishable patterns across different apps. They are subsequently input into the *cohesive feature fusion* for further fusion.

3.4 Cohesive Feature Fusion

The goal of performing feature fusion is to integrate the strengths of diverse detection methods. However, prior solutions that directly concatenate the features with distinct semantics suffer from feature bias issues and yield marginal performance gains, since these features are not inherently aligned in a unified feature space before fusion [28]. Therefore, aligning and complementing features from various detection models remains a critical challenge.

Challenges. Features extracted by different detection methods exhibit distinct semantics that capture diverse aspects of apps . For example, Drebin produces sparse symbolic features while MamaDroid generates features that capture behavior transition using the FCG. Conventional fusion approaches [27,28] directly concatenate the raw features, but they suffer from feature bias issues due to semantic feature misalignment, resulting in loss of discriminative information and marginal performance gain [29,48]. Additionally, long feature vectors exacerbate the curse of dimensionality, causing longer features to dominate shorter ones and leading to overfitting [49]. Therefore, unseen malware that drifts in one feature modality cannot be compensated for by others, rendering the fused features less effective and robust.

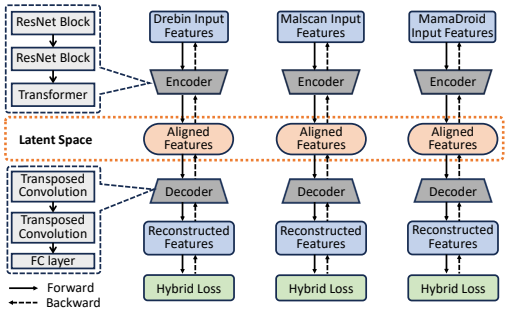


Fig. 7: The architecture of the designed *cohesive feature embedding network*.

To address the challenges, we propose a *cohesive feature embedding network* using a customized autoencoder to align heterogeneous features in a unified feature space, thereby mitigating the feature bias issue. Late fusion is then applied to the aligned features to generate fused features that can fully integrate the strengths of detection methods, enabling mutual compensation of diverse features.

Cohesive Feature Embedding Network. As illustrated in Fig. 7, the autoencoder-based fusion network consists of an encoder and a decoder. The encoder projects the original input features into a low-dimensional latent space to align diverse features, and the decoder reconstructs the original features from the aligned features to retain information.

In *LLMalware*, the encoder adopts a CNN-Transformer backbone to capture crucial information of the original input features for fusion. Specifically, it first contains two ResNet blocks [50] and a two-layer Transformer [51]. A linear layer then projects the extracted features to a unified latent space for feature alignment. The rationale behind the design is that CNNs excel in capturing local spatial information by applying convolutional kernels to features. In contrast, Transformers are adept at capturing global dependencies through self-attention mechanisms, enabling them to model deep relationships between any two positions in the features. This backbone ensures the encoder captures the most crucial information. During this process, the aligned features are not normalized, as normalization removes magnitude information of features and hinders the decoder from accurately reconstructing the original input features [52].

The aligned features are further input into a decoder to reconstruct the original input features. The rationale is that the decoder can reconstruct the original input features with the aligned features once the critical information of the original input features is projected into the latent space. Unlike the encoder, the decoder consists of two transposed convolutional layers and a fully connected layer. We further train this encoder-decoder architecture using a hybrid loss function to ensure the aligned features can capture the most crucial information of the original input features.

Hybrid Loss Function. We first train the autoencoder by minimizing the mean square error (MSE) between the original input features and the reconstructed features to capture essential information of the original input features. The MSE

loss function is defined as $\mathcal{L}_{MSE} = \frac{1}{N}\sum_{i=1}^{N} \|f_i - \hat{f}_i\|_2^2$, where N is the number of training samples, f_i and $\hat{f}_i$ are the original input features and the reconstructed features, respectively. $\|\cdot\|_2^2$ denotes the squared L2 norm between f_i and $\hat{f}_i$.

Nevertheless, relying solely on MSE fails to extract discriminative features, as it does not optimize the inter-class separation and thus weakens the discriminability of apps. In other words, we aim to minimize intra-class distances and maximize inter-class distances, pushing samples away from the decision boundary to reduce misclassification. To this end, we leverage the triplet loss function [53] to optimize the sample distances between malware and benign apps, thereby extracting discriminative features. Specifically, we treat each training sample as an *anchor* sample. Then, we select a sample with the same label as a *positive* sample and a sample with a different label as a *negative* sample. The goal is to minimize the distance between the *anchor* and the *positive* sample while maximizing the distance between the *anchor* and the *negative* sample, which can be formalized as follows:

$$\mathcal{L}_{Tri} = \frac{1}{N}\sum_{i=1}^{N} \max(\|f_{a_i} - f_{p_i}\|_2^2 - \|f_{a_i} - f_{n_i}\|_2^2 + m, 0), \tag{1}$$

where a_i, p_i, and n_i denote the i-th *anchor*, *positive*, and *negative* sample, respectively. f_a, f_p, and f_n are features of the samples. m represents the lower bound, ensuring extreme samples do not affect the loss to prevent overfitting.

To improve training effectiveness, we introduce a hard negative sample selection strategy to select the most challenging negative samples. Instead of randomly selecting negative samples to pair with the anchor, we choose those closer to the anchor than the positive sample, as they are more challenging to classify and provide discriminative information. This avoids training on easy negative samples that provide limited information, speeding up model convergence [54]. This can be formalized as follows: $\|f_a - f_p\|_2 \geq \|f_a - f_n\|_2$.

Finally, we formalize the hybrid loss function as:

$$\mathcal{L} = \alpha \cdot \mathcal{L}_{MSE} + \beta \cdot \mathcal{L}_{Tri}, \tag{2}$$

where α and β are hyperparameters used to balance the contributions of MSE and triplet loss, both set to 1 for fairness [55]. The network is optimized by the Adam optimizer [56].

Late Fusion. After training, we employ a late fusion [57, 58] to integrate the orthogonal detection capabilities of heterogeneous features. Unlike previous solutions that integrate raw features without alignment, we first align the features and then perform fusion. Specifically, we concatenate semantically aligned features in the latent space. This two-stage process preserves the essential information of features and enables them to contribute within an aligned feature space. Therefore, it can mitigate feature bias issues caused by feature misalignment and enhance robustness. When one detection method underperforms, the features of the other methods can compensate in the aligned space, yielding a robust detection performance. These fused features, along with their labels, are then stored in the knowledge base as external knowledge for *agile knowledge update* (§ 3.2).

4 Implementation

4.1 Experimental Setup

Hardware & Software Configurations. We implement *LLMalware* using the PyTorch 2.3.0 framework [59] and train the models on a server equipped with an NVIDIA RTX 4090 GPU and 32 GB RAM with Intel(R) Xeon(R) E-2236 CPU @ 3.40GHz processors. *LLMalware* uses *androguard* [60] to perform static analysis and LangChain [61] to invoke LLMs.

Hyperparameter settings. For the *cohesive feature embedding network*, we set the length of the latent feature space to 256 and use a convolutional kernel size of 5. The batch size and learning rate are set to 128 and 0.001, respectively. We leverage the OpenAI embedding model (*i.e.*, *text-embedding-ada-002* [47]) to transform the text into embeddings.

LLMs. We select GPT-4o [62] as the default LLM due to its superior performance. GPT-3.5 [30] and open-source Llama3.1-8B [63] are also evaluated in the ablation study. We fix the temperature to 0 to minimize randomness.

4.2 Dataset

We construct a new Android app dataset by randomly sampling apps from AndroZoo [64], consisting of more than 130,000 apps collected from 2020 (Year 0) to 2025 (Year 5). We label apps detected by more than three antivirus detectors as malware, and those that pass all detection methods as benign samples. This consensus-based labeling aligns with previous malware research [17, 19, 65, 66] to improve ground truth reliability, as apps flagged by only 1–2 detectors represent ambiguous cases and their maliciousness requires further manual validation [67, 68]. Furthermore, we maintain the ratio of malware to benign apps to 1:9, reflecting the practical imbalance [17,19]. Details of the dataset and concept drift measurement are shown in the appendix (Table 4). Note that the number of apps from 2022 to 2025 is lower since Androzoo has not updated all apps for those years.

4.3 Baselines

We compare *LLMalware* with six detection methods:

- **Drebin** [9] classifies samples using a support vector machine based on eight kinds of static features, such as hardware components and requested permissions.
- **MamaDroid** [10] builds a Markov chain based on the FCG of apps and regards the transition probabilities of the chain as malware features. A random forest (RF) classifier is then trained to perform detection.
- **Malscan** [11] computes the degree centrality of nodes in FCG as features and classifies samples using an RF model.

– **Transcendent** [69] mitigates the concept drift problem by leveraging conformal prediction to detect aging malware classifiers and reject new examples that deviate from the training distribution. In line with the original paper, we compare the performance of samples that are not rejected.
– **CL** [19] is a SOTA continuous learning-based online malware detection framework to mitigate the concept drift problem. It selects new apps with the highest uncertainty to retrain the model, leading to high retraining costs.
– **AppPoet** [70] leverages LLMs to generate semantic summaries from multi-view static features as embedding features and uses an MLP for detection.

4.4 Metrics

Considering the class imbalance in the dataset, we use classification metrics including *macro* F1-score, *macro* false positive rate (*FPR*), *macro* false negative rate (*FNR*), and *macro* balanced error rate (*BER*). A higher F1-score indicates better detection performance, while lower FPR, FNR, and BER reduce misclassifications. These metrics are expressed as: $F1 = \frac{1}{n}\sum_{i=1}^{n}\frac{2\times P_i \times R_i}{P_i+R_i}$, $FPR = \frac{1}{n}\sum_{i=1}^{n}\frac{FP_i}{FP_i+TN_i}$, $FNR = \frac{1}{n}\sum_{i=1}^{n}\frac{FN_i}{FN_i+TP_i}$, and $BER = \frac{1}{n}\sum_{i=1}^{n}\frac{FPR_i+FNR_i}{2}$, where n is the number of classes. TP_i, TN_i, FP_i, and FN_i are true positives, true negatives, false positives, and false negatives of the i-th class, respectively. P_i and R_i are the precision and recall rate of the i-th class, respectively. To measure the detection performance over time, we also compute the cumulative gap (CG) of the above metrics: $CG_{baseline}(t) = \sum_{i=1}^{t}(F_{LLMalware}(i) - F_{Baseline}(i))$, where $F_{LLM}(i)$ and $F_{base}(i)$ is the values of *LLMalware* and baselines, respectively. Unlike the area under time (AUT) metric [69], which does not consider baselines for comparison, CG highlights the cumulative differences between *LLMalware* and baselines over time. Note that the CG of *LLMalware* remains constant at zero over time. Moreover, since CG is the difference between consecutive months, its timeline is one month shorter than the total duration. All the results are averaged over 100 independent runs.

5 Evaluation

In this section, we focus on the following research questions:

– **RQ1:** How effective is *LLMalware* in improving malware detection performance within the same period? (§ 5.1)
– **RQ2:** How robust and efficient is *LLMalware* in detecting new Android apps over time? (§ 5.2)
– **RQ3:** How do the components of *cohesive feature fusion* contribute to the robustness of *LLMalware*? (§ 5.3)
– **RQ4:** How do the LLM-based agents contribute to dynamic report interpretation and agile malware detection? (§ 5.4)

Table 1: The overall performance of *LLMalware* and the baselines within the same period (F1(↑), FPR(↓), FNR(↓), %).

Method	Year 0			Year 0 - Year 1			Year 0 - Year 2			Year 0 - Year 3			Year 0 - Year 4			Year 0 - Year 5		
	F1	FPR	FNR	F1	FPR	FNR	F1	FPR	FNR	F1	FPR	FNR	F1	FPR	FNR	F1	FPR	FNR
Drebin	85.5	11.7	15.7	86.8	7.0	8.7	88.7	3.6	13.9	89.2	5.4	11.9	89.9	5.8	9.9	90.4	6.5	9.2
MamaDroid	86.4	7.9	14.9	88.9	7.8	7.9	89.2	6.4	9.5	90.3	4.2	8.7	90.6	4.0	11.3	91.1	5.5	8.6
Malscan	86.0	4.7	12.9	87.5	7.7	10.8	89.3	4.2	13.5	90.5	5.3	9.0	90.7	10.2	7.0	90.5	9.2	11.8
Transcendent	87.7	7.4	10.4	88.6	9.6	8.1	90.5	4.2	9.7	91.2	2.9	10.6	91.8	7.3	9.5	92.2	7.9	5.5
CL	86.8	8.2	14.4	87.8	5.9	10.4	90.7	5.1	11.1	91.7	4.4	9.1	92.1	5.8	8.2	92.2	4.0	8.9
AppPoet	85.2	7.3	18.1	88.0	5.5	11.5	90.1	9.1	9.5	91.6	9.0	10.7	92.7	6.8	7.9	93.3	5.9	5.5
LLMalware	**93.8**	**2.9**	**8.1**	**94.3**	**4.5**	**5.6**	**94.7**	**2.5**	**8.7**	**95.8**	**1.7**	**8.4**	**96.0**	**2.2**	**3.8**	**96.3**	**2.4**	**3.0**

5.1 RQ1: Performance in the Same Period

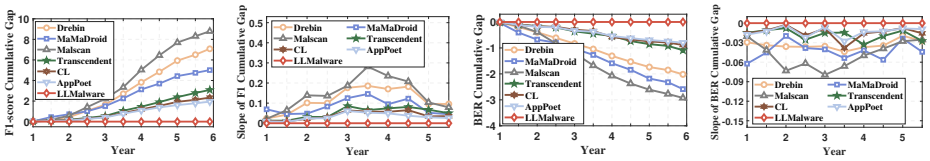
We first evaluate the overall performance of *LLMalware* and baselines in the same period. During the same period, we do not invoke *LLMalware*'s knowledge update or CL's model retraining (*i.e.*, only CL's detection model is invoked). Specifically, we organize the dataset into six chronologically expanding subsets (*e.g.*, Year 0-Year 5). Each subset is further divided into training and test sets with an 8:2 ratio, and we use the training sets to initialize the knowledge base and the test sets for evaluation. We select GPT-4o as the LLM for the report analyzer and the agent in *LLMalware*.

The results are shown in Table 1. *LLMalware* consistently outperforms the baselines in three metrics across all periods, showcasing the effectiveness of the fused features. For example, in Year 0, *LLMalware* achieves an F1-score of 0.938, surpassing Drebin and CL by 8.3% and 7%, respectively. Moreover, when evaluated on the samples in Year 0 - Year 5, *LLMalware* outperforms the best baseline CL by 4.1% in F1-score. Furthermore, *LLMalware* achieves lower FPR and FNR, with maximum reductions of 8.8% and 7.6% in Year 0, respectively. *LLMalware* also outperforms the LLM-assisted baseline AppPoet by 8.6% and 3% in Year 0 and Year 0 - Year 5, respectively. These results highlight the effectiveness and necessity of fused features in *LLMalware* for malware detection.

The superior performance of *LLMalware* is attributed to the *cohesive feature embedding network* and *multi-metric search strategy*. The network extracts robust and discriminative fused features from diverse detection methods to improve robustness. Moreover, the search strategy mitigates representation variance across features and retrieves semantically similar information from the knowledge base, enabling reliable retrieval from the base. Therefore, *LLMalware* can exploit the strengths of diverse detection methods, enhancing the detection performance and robustness.

5.2 RQ2: Robustness to Concept Drift

We evaluate the detection performance of *LLMalware* across different periods. Different from RQ1, we invoke *LLMalware*'s knowledge update and CL's model retraining to update the models monthly. Specifically, samples collected in Year 0 are selected as the training set, and samples collected in Year 1 - Year 5 are first divided into monthly subsets. Each monthly subset is further divided



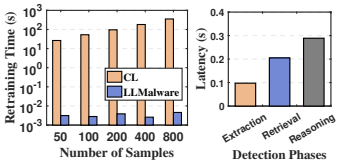
(a) F1-score cumulative gap. (b) Slope of F1-score CG. (c) BER cumulative gap. (d) Slope of BER cumulative gap.

Fig. 8: The overall performance of *LLMalware* and the baselines over time.

into a new sample set and a test set [19]. The new sample set and the test set are completely separate to avoid data leakage. From the new sample set, we select the $K = 50$ most dissimilar samples to the ones in the knowledge base based on cosine similarity (for CL, we use uncertainty scores). These samples are labeled by experts or an anti-malware solution, and then added to *LLMalware*'s knowledge base for updating and CL model retraining.

The results are shown in Fig. 8. We measure the F1-score cumulative gap of different malware detection methods. As shown in Fig. 8(a), the cumulative gaps of baselines like Drebin and MaMaDroid increase to 54% and 76%, while the SOTA baselines, including Transcendent, CL, and AppPoet, reach gaps of around 20%, indicating a significant model deterioration compared to *LLMalware*. By Year 5, the gaps of Transcendent, CL, and AppPoet further expand to around 210%, highlighting their declining robustness and the model aging problem over time. Moreover, as observed in Fig. 8(b), the cumulative gap slopes of all baselines exceed *LLMalware* 's and increase over time. The results indicate that the performance of baselines decreases more sharply than *LLMalware*. In contrast, *LLMalware* maintains more consistent and robust performance over time, as *LLMalware* enhances the detection robustness by integrating the heterogeneous features of various detection methods. Moreover, as shown in Fig. 8(c), the BER cumulative gaps of all baselines increase. For example, the gaps of baselines such as MaMaDroid and CL increase from 70% and 20% in Year 0 to 230% and 79% in Year 5, indicating significant misclassification. Fig. 8(d) further shows that the slopes of baselines' gaps exceed the slope of *LLMalware*, leading to larger performance gaps over time compared to *LLMalware*. Overall, *LLMalware* achieves low performance degradation over extended periods, indicating its robustness to the concept drift problem.

Finally, we measure the execution time to update the knowledge base in *LLMalware* compared with model retraining (*i.e.*, CL). In the experiments, we first simulate a practical scenario where the base and models are initialized on samples collected from Year 0, and then update them with new samples from subsequent months. We test with varying batch sizes of new samples, up to 800 samples per update, and set the epoch to 250 [19]. As shown in Fig. 9(a), *LLMalware* takes only 0.0045 seconds to complete the knowledge update with 800 new samples, while CL requires 354.22 seconds to retrain the model. This performance gap becomes even more significant in the industry, where thousands of new apps with many potential malware are published on a daily basis [71].



(a) Retraining time. (b) Phase latency.

Fig. 9: Latency analysis of CL and *LLMalware*.

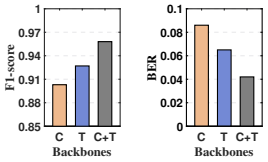


Fig. 10: Different backbones where C, T denote CNN and Transformer.

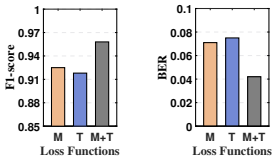


Fig. 11: Different loss functions where M, T denote MSE and Triplet loss.

LLMalware takes less time, as it can efficiently retrieve and update the features in the base without model retraining. The results also validate that the linear search complexity is acceptable. Moreover, as shown in Fig. 9(b), the entire inference latency of *LLMalware* using GPT-4o is about 0.8 seconds per app. The LLM response takes about 0.77 seconds, which is the longest among the three phases, as *LLMalware* leverages the LLM to retrieve knowledge and detect malware. Nevertheless, we argue that *LLMalware* targets app vetting before store release rather than on-device real-time detection. Therefore, the bottleneck under this scenario is the retraining time, while the available time window sufficiently accommodates the inference latency.

5.3 RQ3: Impact of Cohesive Feature Embedding Network

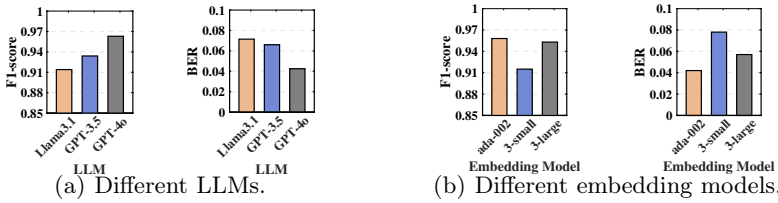
In this section, we conduct experiments to explore the impact of *cohesive feature embedding network*. We follow the experimental setup used in RQ1 and modify the components of the *cohesive feature embedding network*.

We first evaluate the benefits of integrating multiple detection methods and the effectiveness of the proposed fusion network. As shown in Table 2, incorporating features from detection methods using the fusion network consistently improves detection performance. Individual detectors (D, M, S) achieve F1-scores between 0.904 and 0.913, while integrating one more detector (*e.g.*, D+S) boosts the F1-score up to 0.927 with lower FPR and FNR. When all three methods are integrated (D+M+S w/ fusion), the F1-score improves dramatically to 0.963. An ablation study further highlights the effectiveness of the fusion network. The naive concatenation (D+M+S w/o fusion) yields merely an F1-score of 0.908, which is 5.5% lower than that using the fusion network. The improvement stems from our fusion network to integrate features from heterogeneous detection methods. Optimized with a hybrid loss function, it learns to generate robust and discriminative fused features, effectively mitigating feature bias arising from naive concatenation. Hence, *LLMalware* enhances detection performance and robustness against concept drift.

We also evaluate the impact of different network backbones and loss functions. As shown in Fig. 10, compared to the CNN-Transformer backbone, using either CNN or Transformer alone as the backbone results in a significant drop in

Table 2: Experiments on integrating different detection methods and w/ or w/o our fusion network. D, M, S denote **D**rebin, **M**amaDroid, and Mal**S**can.

Method	F1 ↑	FPR ↓	FNR ↓
D	0.904	0.076	0.088
M	0.913	0.081	0.067
S	0.909	0.031	0.090
$D + M$	0.915	0.078	0.070
$D + S$	0.927	0.075	0.068
$M + S$	0.923	0.065	0.074
$D + M + S$ (w/o fusion)	0.908	0.040	0.119
$D + M + S$ (w/ fusion)	**0.963**	**0.024**	**0.030**



(a) Different LLMs. (b) Different embedding models.

Fig. 12: Impact of different LLMs and embedding models in report interpretation.

F1-score by 5.5% and 3.1%, respectively, and an increase in BER by 4.4% and 2.3%. This degradation occurs because neither individual architecture effectively captures both local and global features, leading to information loss during alignment. Moreover, to evaluate the impact of different loss functions, we train two additional fusion networks using MSE and triplet loss, respectively. As shown in Fig. 11, the combination of MSE and Triplet loss outperforms the individual losses by improving the F1-score by up to 3.5% and reducing the BER by 3.3%. The performance boost indicates the hybrid loss function optimizes the network to extract discriminative fused features, thereby improving the performance.

5.4 RQ4: Impact of LLM-Based Agents

We assess the impact of the LLM agents on performance in report interpretation and malware detection. We also follow the experimental setup from RQ1.

We first evaluate the performance of LLMs in report interpretation. Specifically, we use three LLMs, including Llama3.1-8B, GPT-3.5, and GPT-4o to extract information from the reports for further detection. Experimental results reveal a critical insight into the role of LLMs: the quality of extracted information significantly impacts detection performance. As shown in Fig. 12(a), *LLMalware* achieves a higher F1-score with GPT-4o, outperforming Llama3.1 and GPT-3.5 by 4.9% and 2.9%, and reducing BER by around 3%. The gain stems from GPT-4o's powerful semantic understanding and larger context window, enabling it to capture complex behavioral information across the specialized malware analysis reports, while smaller models with a limited context window (*e.g.*, Llama3.1-8B) may overlook information during interpreting the reports.

Additionally, we leverage various text embedding models, including text-embedding-3-small, text-embedding-ada-002, and text-embedding-3-large from OpenAI [47], to assess which models best preserve the information of the extracted text. As shown in Fig. 12(b), text-embedding-ada-002 achieves a higher detection F1-score about 0.96. These models effectively preserve the semantic information in the reports during embedding due to more powerful architectures. However, text-embedding-3-small exhibits a 4.3% F1-score drop and a 3.6% BER increase due to its weaker embedding capabilities. Therefore, we select the text-embedding-ada-002 as the embedding model for *LLMalware*.

We further evaluate the impact of in-context learning on the LLM's ability to interpret the reports. As shown in Fig. 13, without an illustrative example to instruct the LLM, the F1-score drops to 0.904, which is 5.4% lower than that with an example, as the LLM may overlook or output inaccurate information without proper guidance, introducing noise into the embedding. As the number of examples increases, the performance stabilizes, indicating that an example suffices for effective report interpretation, while additional examples incur higher cost but marginal gains. This finding underscores that effective prompt design greatly enhances the reliability of LLM-based malware analysis. Therefore, we adopt one-shot in-context learning for optimal efficiency and robustness.

For *agile knowledge update* in malware detection, we first evaluate the effectiveness of the multi-metric retrieval against a conventional cosine similarity baseline (Cos) and a KNN baseline that selects K candidates based on Euclidean distances. As shown in Fig. 14, our multi-metric search outperforms both the cosine similarity and KNN baselines by 6.4% and 4.5% in F1-score, and 5.1% and 4.9% in BER, respectively, demonstrating the effectiveness of the strategy.

We further assess the impact of different LLM agents. Specifically, we integrate GPT-4o, GPT-3.5, and Llama3.1-8B as the agent for malware detection § 3.2. As shown in Fig. 15, the agent using GPT-4o outperforms Llama3.1 and GPT-3.5 by 4.4% and 2.3% in F1-score, respectively due to its superior text comprehension and reasoning capabilities, enabling accurate tool invocation for detection. Moreover, we evaluate the LLM's performance across different temperature settings in malware detection. The temperature controls the randomness of response [72], with higher values increasing randomness. We vary the temperature from 0 to 1 with a step size of 0.2. As shown in Fig. 16, *LLMalware* achieves the highest F1-score of 0.958 at a temperature of 0. As the temperature increases, the F1-score gradually decreases to 0.914. Meanwhile, the FPR steadily increases to 7.3% and the FNR increases to 8.3% as the temperature rises to 1. Since higher temperatures introduce more randomness in responses, the LLM may generate malformed outputs. Therefore, we fix the temperature to 0 for all experiments to ensure stable reasoning and outputs.

6 Related Work

Android Malware Detection. Existing Android malware detection methods leverage machine learning for detection. [9–11, 28, 73–76]. Recently, LLMs have

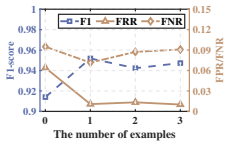


Fig. 13: Impact with in-context learning.

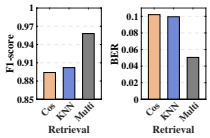


Fig. 14: Impact of retrieval approaches.

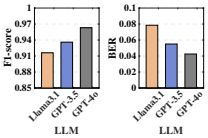


Fig. 15: Impact of different LLM agents.

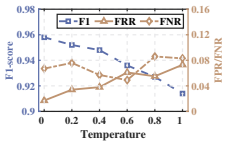


Fig. 16: Impact of temperature settings.

exhibited remarkable capabilities in downstream tasks without manual intervention [77–81]. Therefore, some studies exploit LLMs to enhance malware detection [70,82,83]. For example, AppPoet [70] utilizes the LLMs to generate concise summaries from multi-view static features and convert them into embeddings, which are input to an MLP for malware detection. Compared to these LLM-based detectors, *LLMalware* leverages LLMs to accurately extract behavioral information from highly specialized analysis reports and incorporates an RAG framework to reduce LLM update costs.

Concept Drift. Although these detectors achieve impressive detection performance, they suffer from the concept drift problem [84–86] as malware evolves to bypass detection. Solutions such as continual learning retrains models on a few new samples to capture unseen malware patterns [18–20, 23, 65]. For example, Chen *et al.* [19] calculate and select new samples with higher uncertainty for retraining. Other approaches reject new samples when the confidence falls below a threshold [69, 87, 88] or apply data augmentation to improve robustness [7]. Unlike these methods requiring extensive retraining, *LLMalware* leverages the strengths of diverse detection methods to mitigate the concept drift.

7 Conclusion

We propose *LLMalware*, an LLM-powered robust and efficient Android malware detection framework against the concept drift problem. The key insight of *LLMalware* is that by fully exploiting the orthogonal detection capabilities of diverse detection methods, we can improve the robustness and efficiency of malware detection. *LLMalware* develops and incorporates three novel technical components: *full-spectrum automated feature extraction* to extract static and dynamic features, *cohesive feature fusion* to constructively integrate features of distinct detection methods, and *agile knowledge update* to continuously update the knowledge of LLM for unseen malware detection. Experimental results demonstrate that these technical components jointly help mitigate the concept drift problem in malware detection.

Acknowledgments. We sincerely thank the anonymous reviewers for their constructive comments and invaluable suggestions. This work is supported by Hong Kong GRF Grant No. 15204926, and by Hong Kong CRF Grant No. C5085-25G.

References

1. Y. Yan, Z. Li, Q. A. Chen, C. Wilson, T. Xu, E. Zhai, Y. Li, and Y. Liu, "Understanding and detecting overlay-based android malware at market scales," in *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2019, pp. 168–179.
2. H. Li, Z. Yao, B. Wu, C. Gao, T. Xu, W. Yuan, and X. Luo, "Automated mass malware factory: The convergence of piggybacking and adversarial example in android malicious software generation," in *Proceedings of the 32nd Network and Distributed System Security Symposium (NDSS)*, 2025.
3. L. Shi, J. Ming, J. Fu, G. Peng, D. Xu, K. Gao, and X. Pan, "Vahunt: Warding off new repackaged android malware in app-virtualization's clothing," in *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security (CCS)*, 2020, pp. 535–549.
4. H. Zhou, X. Luo, H. Wang, and H. Cai, "Uncovering intent based leak of sensitive data in android framework," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2022, pp. 3239–3252.
5. S. Ma, C. Chen, S. Yang, S. Hou, T. J. Li, X. Xiao, T. Xie, and Y. Ye, "Careful about what app promotion ads recommend! detecting and explaining malware promotion via app promotion graph," in *Proceedings of the 32nd Network and Distributed System Security Symposium (NDSS)*, 2025.
6. Y. Xiao, C.-C. Liu, F. Mo, and Q. Liang, "A novel lightweight binary-level malware hybrid obfuscation against black-box anti-malware engines," *High-Confidence Computing*, p. 100368, 2025.
7. J. Tian, W. Kong, D. Gao, T. Wang, T. Gu, K. Qiu, Z. Wang, and X. Kuang, "Density boosts everything: A one-stop strategy for improving performance, robustness, and sustainability of malware detectors." in *Proceedings of the 32nd Network and Distributed System Security Symposium (NDSS)*, 2025.
8. https://www.av-test.org/en/, AV-TEST.
9. D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket," in *21st Annual Network and Distributed System Security Symposium (NDSS)*, 2014, pp. 23–26.
10. E. Mariconti, L. Onwuzurike, P. Andriotis, E. D. Cristofaro, G. J. Ross, and G. Stringhini, "Mamadroid: Detecting android malware by building markov chains of behavioral models," in *24th Annual Network and Distributed System Security Symposium (NDSS)*, 2017.
11. Y. Wu, X. Li, D. Zou, W. Yang, X. Zhang, and H. Jin, "Malscan: Fast market-wide mobile malware scanning by social-network centrality analysis," in *34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2019, pp. 139–150.
12. L. Gong, Z. Li, F. Qian, Z. Zhang, Q. A. Chen, Z. Qian, H. Lin, and Y. Liu, "Experiences of landing machine learning onto market-scale mobile malware detection," in *Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys)*, 2020, pp. 1–14.
13. H. Zhu, X. Chen, L. Wang, Z. Xu, and V. S. Sheng, "A dynamic analysis-powered explanation framework for malware detection," *IEEE Transactions on Knowledge and Data Engineering*, vol. 36, no. 12, pp. 7483–7496, 2024.
14. Y. Lin, J. Wong, X. Li, H. Ma, and D. Gao, "Peep with a mirror: Breaking the integrity of android app sandboxing via unprivileged cache side channel," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 2119–2135.

15. M. Zhang, H. Marklund, N. Dhawan, A. Gupta, S. Levine, and C. Finn, "Adaptive risk minimization: Learning to adapt to domain shift," *Advances in Neural Information Processing Systems (NIPS)*, vol. 34, pp. 23 664–23 678, 2021.
16. L. Yuan, H. Li, B. Xia, C. Gao, M. Liu, W. Yuan, and X. You, "Recent advances in concept drift adaptation methods for deep learning," in *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence (IJCAI)*, 2022, pp. 5654–5661.
17. X. Zhang, Y. Zhang, M. Zhong, D. Ding, Y. Cao, Y. Zhang, M. Zhang, and M. Yang, "Enhancing state-of-the-art classifiers with api semantics to detect evolved android malware," in *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security (CCS)*, 2020, pp. 757–770.
18. Q. Qiao, R. Feng, S. Chen, F. Zhang, and X. Li, "Multi-label classification for android malware based on active learning," *IEEE Transactions on Dependable and Secure Computing*, no. 01, pp. 1–18, 2022.
19. Y. Chen, Z. Ding, and D. Wagner, "Continuous learning for android malware detection," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 1127–1144.
20. T. Sun, N. Daoudi, W. Pian, K. Kim, K. Allix, T. F. Bissyandé, and J. Klein, "Temporal-incremental learning for android malware detection," *ACM Transactions on Software Engineering and Methodology*, 2024.
21. K. Satvat, R. Gjomemo, and V. Venkatakrishnan, "Extractor: Extracting attack behavior from threat reports," in *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2021, pp. 598–615.
22. S. Feng and C. Chen, "Gifdroid: Automated replay of visual bug reports for android apps," in *Proceedings of the 44th International Conference on Software Engineering (ICSE)*, 2022, pp. 1045–1057.
23. N. Daoudi, K. Allix, T. F. Bissyandé, and J. Klein, "Guided retraining to enhance the detection of difficult android malware," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, 2023, pp. 1131–1143.
24. J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in Neural Information Processing Systems (NIPS)*, vol. 35, pp. 24 824–24 837, 2022.
25. S. Min, X. Lyu, A. Holtzman, M. Artetxe, M. Lewis, H. Hajishirzi, and L. Zettlemoyer, "Rethinking the role of demonstrations: What makes in-context learning work?" in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2022, pp. 11 048–11 064.
26. Y. Zhou, J. Li, Y. Xiang, H. Yan, L. Gui, and Y. He, "The mystery of in-context learning: A comprehensive survey on interpretation and analysis," in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2024, pp. 14 365–14 378.
27. G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez, and A. Ribagorda, "Evolution, detection and analysis of malware for smart devices," *IEEE communications surveys & tutorials*, vol. 16, no. 2, pp. 961–987, 2013.
28. Z. Liu, L. F. Zhang, and Y. Tang, "Enhancing malware detection for android apps: Detecting fine-granularity malicious components," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2023, pp. 1212–1224.

29. Y. Dai, F. Gieseke, S. Oehmcke, Y. Wu, and K. Barnard, "Attentional feature fusion," in *Proceedings of the IEEE/CVF winter conference on applications of computer vision (WACV)*, 2021, pp. 3560–3569.
30. "Chatgpt," https://openai.com/blog/chatgpt/, OpenAI.
31. H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.
32. H. Xu, L. Han, Q. Yang, M. Li, and M. Srivastava, "Penetrative ai: Making llms comprehend the physical world," in *Proceedings of the 25th International Workshop on Mobile Computing Systems and Applications*, 2024, pp. 1–7.
33. M. Li, R. Wang, X. Zhou, Z. Zhu, Y. Wen, R. Tan, H. Zheng, and S. Kennedy, "Chatdc: Geometric-aware data center digital twin generation via large language models," *ACM Transactions on Internet of Things*, 2025.
34. L. Shen, Q. Yang, X. Huang, Z. Ma, and Y. Zheng, "Gpiot: Tailoring small language models for iot program synthesis and development," in *Proceedings of the 23rd ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2025, pp. 199–212.
35. X. Cheng, J. Zhang, N. Ding, N. Li, Y. Li, T. Wu, W. Xu, J. Zhang, and Q. Sun, "Integrated ai and communications: A two-way catalysis toward 6g and beyond," *Journal of Communications and Information Networks*, vol. 10, no. 3, pp. 191–200, 2025.
36. G. Aguzzi, N. Farabegoli, and M. Viroli, "A language-based approach to macroprogramming for iot systems through large language models," *ACM Transactions on Internet of Things*, 2025.
37. X. Huang, Q. Yang, L. Shen, Z. Ma, and Y. Zheng, "Jailbreaking embodied llms via action-level manipulation," in *Proceedings of the 2026 ACM/IEEE International Conference on Embedded Artificial Intelligence and Sensing Systems (SenSys)*, 2026, pp. 1057–1071.
38. Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou, "Mobilebert: a compact task-agnostic bert for resource-limited devices," *arXiv preprint arXiv:2004.02984*, 2020.
39. M. T. Alam, D. Bhusal, Y. Park, and N. Rastogi, "Looking beyond iocs: Automatically extracting attack patterns from external cti," in *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, 2023, pp. 92–108.
40. L. N. Q. Do and E. Bodden, "Explaining static analysis with rule graphs," *IEEE Transactions on Software Engineering*, vol. 48, no. 2, pp. 678–690, 2020.
41. P. S. H. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Advances in Neural Information Processing Systems (NIPS)*, 2020.
42. T. Xu, M. Pan, Y. Pei, G. Li, X. Zeng, T. Zhang, Y. Deng, and X. Li, "Guider: Gui structure and vision co-guided test script repair for android apps," in *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, 2021, pp. 191–203.
43. S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," in *The Eleventh International Conference on Learning Representations (ICLR)*, 2023.
44. S. Zhang, "Challenges in knn classification," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 10, pp. 4663–4675, 2021.
45. https://github.com/MobSF/Mobile-Security-Framework-MobSF.git, MobSF.

46. B. Rashidi, C. Fung, A. Nguyen, and T. Vu, "Android permission recommendation using transitive bayesian inference model," in *European Symposium on Research in Computer Security (ESORICS)*. Springer, 2016, pp. 477–497.
47. https://platform.openai.com/docs/guides/embeddings, Open AI Embedding Models.
48. M. Zhang, T. Liu, Y. Piao, S. Yao, and H. Lu, "Auto-msfnet: Search multi-scale fusion network for salient object detection," in *Proceedings of the 29th ACM international conference on multimedia (ACM MM)*, 2021, pp. 667–676.
49. S. S. Ghosal, Y. Sun, and Y. Li, "How to overcome curse-of-dimensionality for out-of-distribution detection?" in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 38, no. 18, 2024, pp. 19 849–19 857.
50. K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2016, pp. 770–778.
51. A. Vaswani, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017.
52. J. Ricker, D. Lukovnikov, and A. Fischer, "Aeroblade: Training-free detection of latent diffusion images using autoencoder reconstruction error," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024, pp. 9130–9140.
53. A. Hermans, L. Beyer, and B. Leibe, "In defense of the triplet loss for person re-identification," *arXiv preprint arXiv:1703.07737*, 2017.
54. T. Formal, C. Lassance, B. Piwowarski, and S. Clinchant, "From distillation to hard negative sampling: Making sparse neural ir models more effective," in *Proceedings of the 45th international ACM SIGIR conference on research and development in information retrieval (SIGIR)*, 2022, pp. 2353–2359.
55. C. Fifty, E. Amid, Z. Zhao, T. Yu, R. Anil, and C. Finn, "Efficiently identifying task groupings for multi-task learning," *Advances in Neural Information Processing Systems (NIPS)*, vol. 34, pp. 27 503–27 516, 2021.
56. D. P. Kingma, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
57. Y. Zhang, L. Gong, L. Fan, P. Ren, Q. Huang, H. Bao, and W. Xu, "A late fusion cnn for digital matting," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, 2019, pp. 7469–7478.
58. T. Zhang, X. Liu, E. Zhu, S. Zhou, and Z. Dong, "Efficient anchor learning-based multi-view clustering–a late fusion method," in *Proceedings of the 30th ACM International Conference on Multimedia (ACM MM)*, 2022, pp. 3685–3693.
59. A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems (NIPS)*, vol. 32, 2019.
60. https://github.com/androguard/androguard, androguard.
61. https://github.com/langchain-ai/langchain, Langchain.
62. A. Hurst, A. Lerer, A. P. Goucher, A. Perelman, A. Ramesh, A. Clark, A. Ostrow, A. Welihinda, A. Hayes, A. Radford *et al.*, "Gpt-4o system card," *arXiv preprint arXiv:2410.21276*, 2024.
63. A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.
64. K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "Androzoo: Collecting millions of android apps for the research community," in *Proceedings of the 13th In-*

ternational Conference on Mining Software Repositories (MSR). ACM, 2016, pp. 468–471.
65. K. Xu, Y. Li, R. Deng, K. Chen, and J. Xu, "Droidevolver: Self-evolving android malware detection system," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2019, pp. 47–62.
66. S. Dambra, Y. Han, S. Aonzo, P. Kotzias, A. Vitale, J. Caballero, D. Balzarotti, and L. Bilge, "Decoding the secrets of machine learning in malware classification: A deep dive into datasets, feature extraction, and model performance," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2023, pp. 60–74.
67. B. Andow, A. Nadkarni, B. Bassett, W. Enck, and T. Xie, "A study of grayware on google play," in *2016 IEEE Security and Privacy Workshops (SPW)*, 2016, pp. 224–233.
68. S. Chen, L. Fan, C. Gao, F. Song, and Y. Liu, "Peeking into the gray area of mobile world: An empirical study of unlabeled android apps," in *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*, 2021, pp. 579–590.
69. F. Barbero, F. Pendlebury, F. Pierazzi, and L. Cavallaro, "Transcending transcend: Revisiting malware classification in the presence of concept drift," in *2022 IEEE Symposium on Security and Privacy (S&P)*, 2022, pp. 805–823.
70. W. Zhao, J. Wu, and Z. Meng, "Apppoet: Large language model based android malware detection via multi-view prompt engineering," *Expert Systems with Applications*, vol. 262, p. 125546, 2025.
71. M. Guo, S. Demetriou, J. Yang, M. Leighton, D. Hu, T. Bao, A. Adhikari, T. Kooburat, A. Kim, and C. Tang, "{MobileConfig}: Remote configuration management for mobile apps at hyperscale," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, 2024, pp. 1867–1882.
72. Z.-H. Qiu, S. Guo, M. Xu, T. Zhao, L. Zhang, and T. Yang, "To cool or not to cool? temperature network meets large foundation models via dro," in *Forty-first International Conference on Machine Learning (ICML)*, 2024.
73. K. Zhao, H. Zhou, Y. Zhu, X. Zhan, K. Zhou, J. Li, L. Yu, W. Yuan, and X. Luo, "Structural attack against graph based android malware detection," in *Proceedings of the 2021 ACM SIGSAC conference on computer and communications security (CCS)*, 2021, pp. 3218–3235.
74. B. Kondracki, B. A. Azad, N. Miramirkhani, and N. Nikiforakis, "The droid is in the details: Environment-aware evasion of android sandboxes," in *29th Annual Network and Distributed System Security Symposium (NDSS)*, 2022.
75. Y. Shen, P.-A. Vervier, and G. Stringhini, "A large-scale temporal measurement of android malicious apps: Persistence, migration, and lessons learned," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 1167–1184.
76. P. He, Y. Xia, X. Zhang, and S. Ji, "Efficient query-based attack against ml-based android malware detection under zero knowledge setting," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2023, pp. 90–104.
77. S. Schwartz, A. Yaeli, and S. Shlomov, "Enhancing trust in llm-based ai automation agents: New considerations and future challenges," *arXiv preprint arXiv:2308.05391*, 2023.
78. Z. Yu, M. Wen, X. Guo, and H. Jin, "Maltracker: A fine-grained npm malware tracker copiloted by llm-enhanced dataset," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, 2024, pp. 1759–1771.

79. H. Wen, Y. Li, G. Liu, S. Zhao, T. Yu, T. J.-J. Li, S. Jiang, Y. Liu, Y. Zhang, and Y. Liu, "Autodroid: Llm-powered task automation in android," in *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2024, pp. 543–557.
80. S. Lee, J. Choi, J. Lee, M. H. Wasi, H. Choi, S. Ko, S. Oh, and I. Shin, "Mobilegpt: Augmenting llm with human-like app memory for mobile task automation," in *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2024, pp. 1119–1133.
81. J. Yuan, C. Yang, D. Cai, S. Wang, X. Yuan, Z. Zhang, X. Li, D. Zhang, H. Mei, X. Jia *et al.*, "Mobile foundation model as firmware," in *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2024, pp. 279–295.
82. X. Qian, X. Zheng, Y. He, S. Yang, and L. Cavallaro, "Lamd: Context-driven android malware detection and classification with llms," in *2025 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2025, pp. 126–136.
83. M. Dong, L. Liu, Q. Guo, H. Bai, R. Gong, Y. Bai, W. He, Z. Wang, G. Xu, and J. Zhang, "Leodroid: An llm-based few-shot multi-label detection for android malware," in *2025 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2025, pp. 294–306.
84. K. Panchal, S. Choudhary, S. Mitra, K. Mukherjee, S. Sarkhel, S. Mitra, and H. Guan, "Flash: concept drift adaptation in federated learning," in *International Conference on Machine Learning (ICML)*, 2023, pp. 26 931–26 962.
85. M. Kim, S.-H. Hwang, and S. E. Whang, "Quilt: Robust data segment selection against concept drifts," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2024, pp. 21 249–21 257.
86. Q. Wen, W. Chen, L. Sun, Z. Zhang, L. Wang, R. Jin, T. Tan *et al.*, "Onenet: Enhancing time series forecasting models under concept drift by online ensembling," *Advances in Neural Information Processing Systems (NIPS)*, vol. 36, 2024.
87. R. Jordaney, K. Sharad, S. K. Dash, Z. Wang, D. Papini, I. Nouretdinov, and L. Cavallaro, "Transcend: Detecting concept drift in malware classification models," in *26th USENIX security symposium (USENIX security 17)*, 2017, pp. 625–642.
88. H. Li, G. Xu, L. Wang, X. Xiao, X. Luo, G. Xu, and H. Wang, "Malcertain: Enhancing deep neural network based android malware detection by tackling prediction uncertainty," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (ICSE)*, 2024, pp. 1–13.
89. H. Zhang, S. Li, P. Wang, D. Zeng, and S. Ge, "M3d: Dataset condensation by minimizing maximum mean discrepancy," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 38, no. 8, 2024, pp. 9314–9322.
90. S. Patro and K. K. Sahu, "Normalization: A preprocessing stage," *arXiv preprint arXiv:1503.06462*, 2015.

A Details of Detection Methods

Table 3 illustrates the features extracted by the detection methods and the length of the feature vector of each method.

Table 3: Details of detection methods used in *LLMalware*.

Methods	Features	Length
Drebin	Hardware components, requested permissions, app components, filtered intents, restricted API calls, used permissions, suspicious API calls	121591
MamaDroid	Transition probabilities of function call graph (FCG)	121
Malscan	Degree centrality of FCG	21986
MobSF	API calls, URLs, activities, TLS/SSL configuration	1536

B An Illustrative Example of the Sample Selection Strategy

Fig. 17 illustrates an example of the strategy before and after training. The orange circle represents the decision boundary between two classes, with the anchor at the origin. Green and blue circles represent positive and negative samples, respectively. The blue circle within the boundary is selected as the hard negative sample because it is closer to the anchor than the green positive sample. After learning, the hard negative sample is pushed outside the boundary, and the positive sample is pulled closer to the anchor. Therefore, the decision boundary effectively separates the two classes, enabling the model to extract more discriminative features.

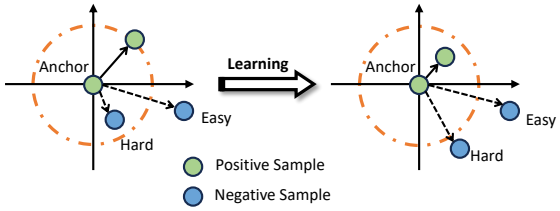


Fig. 17: Illustration of the hard negative sample selection strategy.

C Details of Dataset

We collected over 130,000 apps from Androzoo from 2020 to 2025. Note that the number of apps from 2022 to 2025 is lower since Androzoo has not updated all apps for those years.

Table 4: Dataset details.

Year	Benign Apps	Malicious Apps	Total
2020	48532	5393	53925
2021	44641	4949	49590
2022	12583	1400	13983
2023	5000	558	5558
2024	5910	642	6552
2025	1488	113	1601
Total	118154	13055	131209

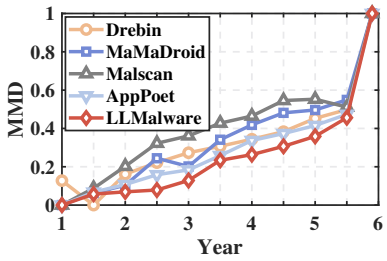


Fig. 18: Maximum mean discrepancy using various features over the years.

D Concept Drift Measurement

To quantify the concept drift, we compute maximum mean discrepancy (MMD) [89] to assess the extent of concept drift over the years. Higher MMD indicates greater distribution differences between samples from different months within the same feature set (*e.g.*, Drebin). Note that the MMD values of different detection methods are not directly comparable due to the varying feature spaces. Specifically, we use samples from Year 0 as the reference set and compare the MMD with samples collected in each month from Year 1 to Year 5. We compute the MMD using features from Drebin, MamaDroid, Malscan, AppPoet, and *LLMalware*. As Transcendent and CL use the same feature set as Drebin, their MMD values are identical. We normalize the lowest and highest MMD to 0 and 1 using min-max normalization [90].

As shown in Fig. 18, the MMD values with all the different features exhibit a significant increase over time, confirming the presence and severity of concept drift as new apps emerge. *LLMalware* shows a slower increase trend in the MMD values, indicating that *LLMalware* effectively enhances the robustness of features by integrating the strengths of diverse detection methods and optimizing the feature distances between benign and malicious apps, thereby mitigating concept drift. This growing distribution discrepancy highlights the evolving nature of malware and validates the necessity and credibility of our system design.