International Journal of Scientific Research and Modern Technology (IJSRMT) ijsrmt.com
Volume 2, Issue 12, 2023
DOI: https://doi.org/10.38124/ijsrmt.v2i12.900

Large Language Models- Powered Identification and Analysis of Kubernetes Misconfigurations

Ravikant Singh[1]

[1]Sr. Data Engineering Manager

Publication Date: 2023/12/29

Abstract

Kubernetes configuration files (KCFs) present a significant challenge due to their inherent complexity and proneness to errors, often leading to security vulnerabilities and operational disruptions. Traditional rule-based (RB) tools designed for detecting KCF misconfigurations rely on predefined, static rule sets, which limit their adaptability and ability to identify newly emerging misconfigurations. Moreover, RB tools frequently suffer from inaccuracies caused by errors in coding detection rules. Current approaches for identifying and analysing KCF misconfigurations are often constrained by limited scalability, narrow detection coverage, or high expertise demands, and they typically lack automated remediation alongside detection capabilities. Recent innovations utilizing large language models (LLMs) in this domain often depend on API-driven, general-purpose, and primarily commercial models, which introduce security concerns, inconsistent classification results, and elevated costs. In this paper, we present GenKubeSec, a robust and adaptive LLM-powered framework that not only detects a broad spectrum of KCF misconfigurations but also pinpoints their precise locations, provides detailed reasoning.

Empirical evaluations reveal that GenKubeSec achieves comparable precision compared to three leading industry-standard RB tools. Additionally, an independent review of a random sample of KCFs by a Kubernetes security expert validated that GenKubeSec's explanations for Identification and Analysis were 100% accurate, insightful, and highly beneficial.

Keywords: *Large Language Models, Kubernetes, Misconfiguration Detection, Automation, Artificial Intelligence.*

I. INTRODUCTION

The landscape of cloud-native computing has experienced a remarkable evolution, with the widespread adoption of container-based environments occurring in a relatively short time. Kubernetes (K8s) has become the leading platform for managing containerized applications, largely due to its numerous advantages, including efficiency, scalability, portability, isolation, and flexibility. Its widespread adoption is further bolstered by robust community support, which has solidified its central role in modern software development ecosystems. The availability of countless Kubernetes configuration files (KCFs) in public repositories has simplified the process of creating and deploying new cloud environments and applications, enabling developers to easily replicate open-source KCFs for their own use.

However, this ease of use introduces significant risks. Publicly available KCFs often contain undetected flaws, such as bugs, vulnerabilities, and misconfigurations (collectively referred to as "misconfigs"), which can leave deployed systems vulnerable to cyberattacks. Furthermore, as Kubernetes has been embraced across a wide range of production environments, and with open-source KCFs being readily accessible, attackers frequently target these files to identify misconfigs that could serve as entry points into clusters or applications. To address these risks, there is a critical need for effective tools capable of identifying and mitigating misconfigs in KCFs.

Rule-based (RB) static analysis tools such as SLI-Kube, Checkov and Terrascan are widely recognized for evaluating the security of KCFs based on predefined rules. However, these tools are inherently limited by their static nature, requiring the creation of new detection rules to handle emerging misconfigs, which can be error-prone and time-consuming. More recent research has explored innovative approaches to detecting KCF misconfigs, including the use of knowledge graphs and topological graphs. Despite their potential, these methods face challenges related to scalability, the need for domain expertise, reliance on high-quality data, and a lack of automated remediation capabilities. Recent advancements in large language models (LLMs) have demonstrated their effectiveness in various domains, including cybersecurity

Singh, R. (2023). Large Language Models- Powered Identification and Analysis of Kubernetes Misconfigurations. *International Journal of Scientific Research and Modern Technology*, *2*(12), 39–48. https://doi.org/10.38124/ijsrmt.v2i12.900

with several studies proposing their use for static analysis tasks. Since KCFs are semi-structured text files, a few studies have specifically explored LLM-based approaches for detecting and remediating KCF misconfigs. However, these studies exhibit several shortcomings:

- They focus only on specific sub-tasks rather than offering a comprehensive solution encompassing detection, localization and reasoning.
- They rely on general-purpose LLMs with limited task-specific fine-tuning, resulting in suboptimal performance.
- They use external LLMs via APIs, introducing privacy risks and additional overhead.
- Their evaluations are based on small datasets with limited misconfig types; and
- Their experimental results are sparse, leaving uncertainty about their performance in future scenarios.

In response to the need for robust solutions to KCF-related risks, the limitations of existing RB and graph-based approaches, and the potential of LLMs, this paper introduces GenKubeSec: an innovative and comprehensive LLM-based framework that addresses these challenges. The framework consists of three core components:

- Data Collection and Standardization: A large dataset of KCFs is compiled, labelled, and standardized using a unified misconfig index developed specifically for this purpose.
- GenKubeDetect: Recognizing the poor performance of generic pretrained LLMs, a base LLM is fine-tuned using an extensive and continually expanding dataset of labelled KCFs. This enables it to accurately detect a wide range of misconfigs.
- GenKubeResolve: A separate LLM is adapted using prompt engineering and few-shot learning to provide detailed localization of misconfigs, human-readable explanations, and actionable remediation recommendations.

Unlike previous approaches that used LLMs for misconfig detection, GenKubeSec delivers an end-to-end solution. It not only combines detection and remediation but also includes precise localization and reasoning for identified misconfigs.

Our evaluation demonstrates that GenKubeDetect achieves a precision of 0.990 (comparable to three leading RB tools) and a recall of 0.999 (outperforming any individual tool). Expert analysis of false positives revealed that over 83% were valid detections, showcasing the model's ability to generalize and identify misconfigs not covered by existing rule-based tools. This high performance is achieved with a lightweight LLM architecture, which reduces computational requirements. Moreover, GenKubeDetect supports both batch-mode learning during fine-tuning and continuous learning for new misconfigs with minimal examples. This comprehensive framework represents a significant step forward in mitigating the risks as sociated with Kubernetes misconfigurations, offering a scalable, efficient, and secure solution.

II. OVERVIEW

- *Kubernetes and Security Challenges, LLM Models*

- *Kubernetes and Kubernetes Configuration File:*

A Kubernetes Configuration File (KCF), also referred to as a manifest file is any document that specifies the configuration of various Kubernetes components. It outlines how an application or service should be deployed and operated within a Kubernetes cluster.

KCFs offer a humanoid-understandable format, typically written in YAML or JSON, to define the requirements of an application. These files ensure reproducible deployments and consistent application behaviour across environments. As foundational elements in containerized and cloud-native infrastructures, KCFs empower developers and administrators to efficiently define, manage, and scale applications. They are indispensable for maintaining reliability and flexibility in Kubernetes-based systems.

- *Kubernetes Configuration File Misconfigs and Vulnerabilities:*

The complexity involved in configuring Kubernetes Configuration Files (KCFs) largely arises from the intricate interdependencies among the numerous configuration parameters [86]. This complexity often results in misconfigurations (misconfigs) within KCFs. A KCF misconfig occurs when system components are improperly configured. Such errors can create security vulnerabilities and negatively impact system performance, integrity, and workloads.

Misconfigs in KCFs can manifest across various levels—edge, application, or cluster—providing attackers with opportunities to compromise containers and host systems. They may also involve parameters related to resource allocation, networking, security policies, and other critical configurations. The repercussions of such misconfigs can be severe, leading to application downtime, degraded performance, inefficient use of resources, and even non-compliance with industry standards and data protection regulations.

- *Large Language Models:*

Large Language Models (LLMs) signify a major leap forward in the field of natural language processing (NLP), improving the capacity of machines to comprehend and produce language that resembles human communication. By leveraging deep learning (DL) methodologies and vast collections of text data, LLMs perform exceptionally well in various applications, including sentiment analysis, text generation, and translation. Modern pretrained LLMs are constantly advancing, growing in both size and complexity, and can be tailored for specific areas of expertise employing techniques such as fine-tuning and continual learning, as will be detailed subsequently.

The process of fine-tuning a pretrained large language model (LLM) involves supplementary training that concentrates on particular objectives, such as aligning the model's outputs with user expectations, generating responses that are both ethical and accurate, and enhancing its performance on specific tasks. This approach not only bolsters the model's understanding and responsiveness to task-oriented instructions but also fortifies its adaptability to shifts in domain, without significantly raising computational expenses. Fine-tuning techniques can vary, encompassing transfer learning with task-specific datasets, instruction-tuning with formatted inputs, and alignment-tuning to ensure that model outputs reflect human values. These methodologies help ensure that LLMs perform effectively in a range of specific contexts.

Continuous learning (CL) is a crucial element in contemporary machine learning (ML). It allows systems to progressively acquire and assimilate various skills or information through an additional training phase that utilizes newly obtained data. This methodology is vital for creating resilient and flexible ML systems that can adapt to evolving requirements and efficiently integrate new insights. In the realm of domain adaptation, CL improves a large language model's (LLM) capacity to maintain original domain knowledge while also learning from the specific domain distribution, thereby facilitating effective adaptation without the risk of 'catastrophic forgetting.' Furthermore, CL ensures computational efficiency during the domain adaptation process. Techniques such as next sentence prediction (NSP), which aids the model in preserving context and coherence across sequences, and masking, which directs the model's focus to pertinent sections of the input, along with unsupervised learning, significantly enhance the comprehension and response abilities of LLMs.

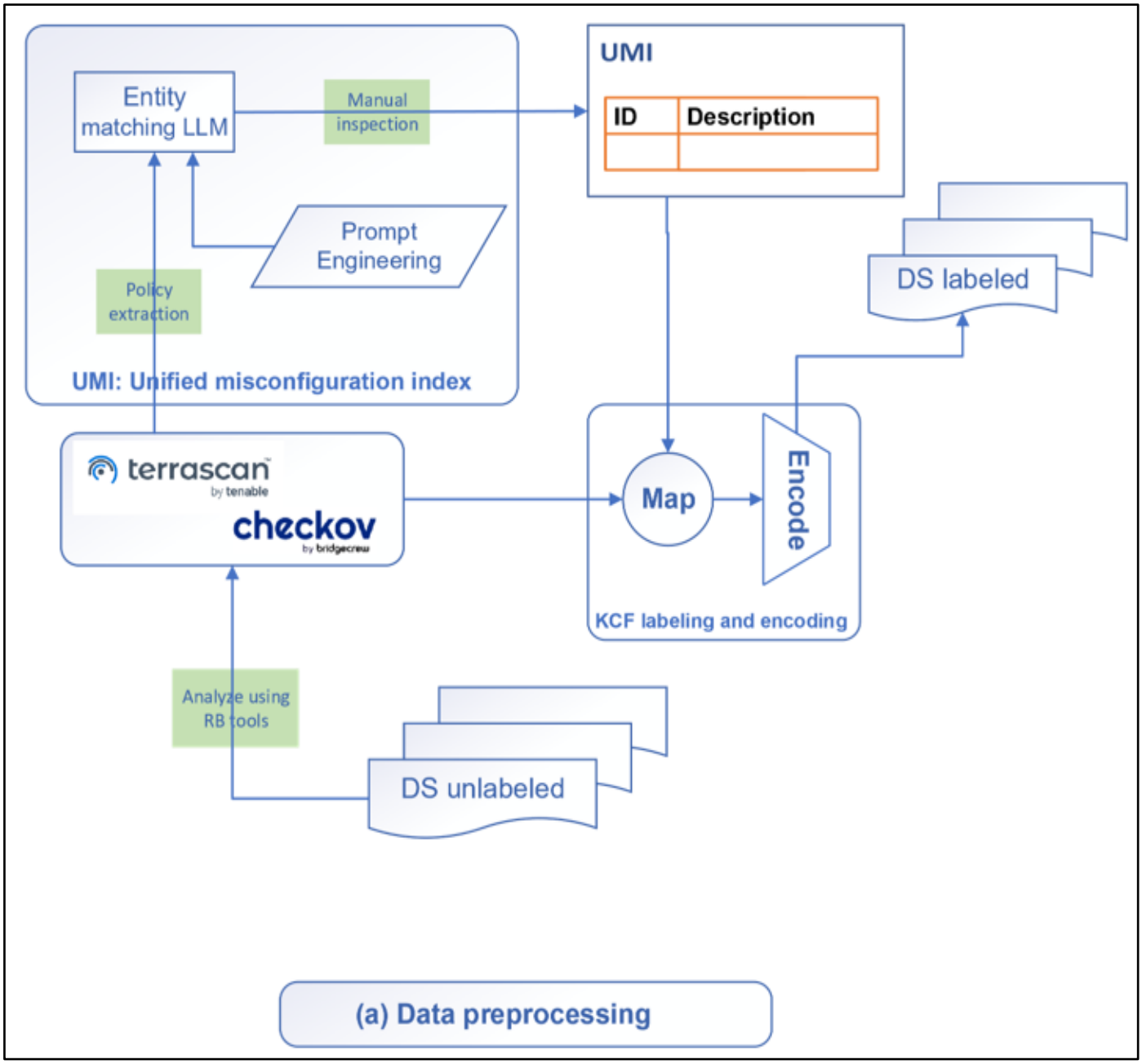

Fig 1A Data Processing Component: A Unified Misconfiguration Index (UMI) is Established, with KCFs Being Labelled and Encoded Accordingly.

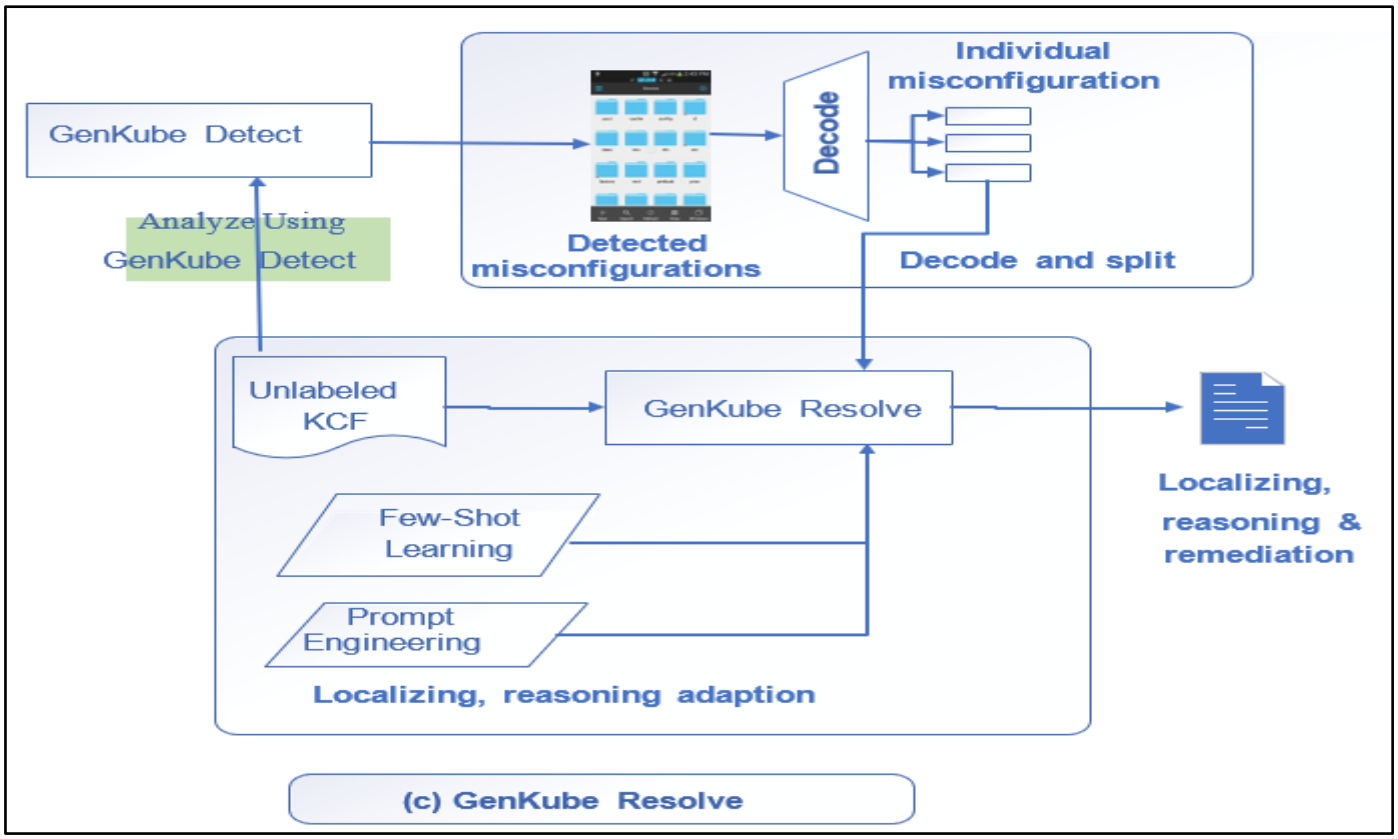

Fig 1B GenKubeDetect Component: The LLM has Undergone Training Aimed at Improving its Understanding of the KCF Structure and its Ability to Detect Misconfigurations.

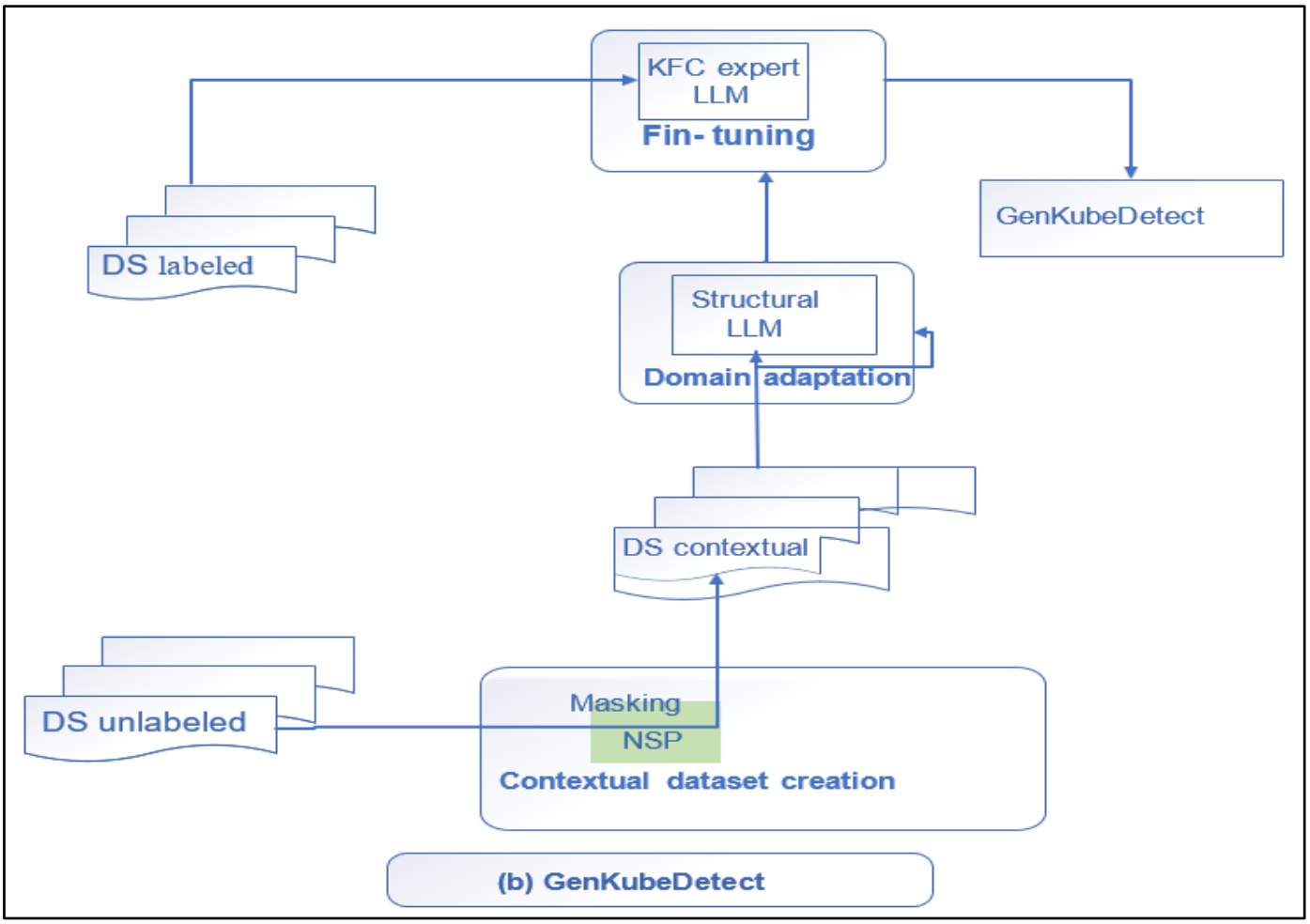

Fig 1C GenKube Resolve Component: The Localization, Reasoning, and Remediation of Detected Misconfigurations are Achieved Using a Pretrained Large Language Model that Incorporates Prompt Engineering and Few-Shot Learning Methodologies

III. PROJECTED METHODS

➢ *GenKubeDetect Component*

The GenKubeDetect module, part of GenKubeSec, is an advanced large language model (LLM) tailored for the detection of misconfigurations in KCFs. Its core purpose is to analyze a provided KCF and generate a list of identified misconfigurations, formatted according to the encoded labels outlined earlier. This is done by choosing a foundational LLM that is ideally pretrained on English semi-structured data, which reflects the hierarchical format characteristic of KCFs, and subsequently fine-tuning it for misconfiguration detection. The fine-tuning process is executed in two distinct phases: the first phase focuses on enabling the model to understand the general structure of KCFs, while the second phase is dedicated to enhancing its capability to identify misconfigurations within those KCFs.

- *To Enhance the Understanding of KCF Structures Within the Base LLM,*

GenKubeDetect's model undergoes fine-tuning with the $DScontextual$ dataset (Refer to Figure 1B). The $DScontextual$ dataset is generated through the application of two techniques on $DSunlabeled$: masking and Next Sentence Prediction (NSP) (Sec. 2.1.3). Masking is applied randomly to 16% of each KCF, a percentage chosen based on common practices in similar scenarios. Through NSP, GenKubeDetect is trained to anticipate the second half of a KCF using the first half as context. This initial fine-tuning phase equips the LLM with the capability to identify and interpret the intricate details and relationships within KCFs, significantly improving its performance in subsequent domain-specific tasks, such as misconfiguration detection. The result of this phase is the structural LLM, illustrated in Figure 1B.

- *The Structural LLM is Refined for the Detection of KCF Misconfigurations*

Once it Has been adapted to the KCF framework, the model is further fine-tuned using $DSlabeled$, allowing it to operate as a multi-label classifier. Consequently, when a KCF is input during inference, the LLM generates text that represents the predicted class labels for any identified misconfigurations. To improve this process, we utilized Low-Rank Adaptation (LoRA). This technique enables the fine-tuning of a select group of parameters within a large pre-trained model by decomposing a large matrix into two smaller low-rank matrices in the attention layers, while keeping the original weights intact. This method enhances efficiency, requiring fewer resources and allowing for faster convergence without the need to retrain the entire model from the ground up. LoRA ensures the original model's weights remain unchanged, preserving its knowledge and capabilities while adapting to specific tasks or datasets. The output of this process is the KCF expert LLM, as depicted in Figure 1B.

➢ *The GenKubeResolve Component*

With a KCF as input, GenKubeDetect is trained to produce one or more misconfiguration labels in the previously defined encoded format. To further enhance GenKubeSec's functionality, GenKubeResolve is trained to deliver.

- The exact location of the identified misconfiguration,
- Insights into the nature or cause of the misconfiguration, and
- Proposed remediation measures.

This is executed in two stages: the first stage involves decoding and dividing the class labels, and the second stage entails adapting a pretrained LLM through prompt engineering techniques, as described in the following sections.

- *In the First Step of GenKubeResolve, the Encoded Labels Undergo a Decoding Process*

(Refer to Figure 1C) and are subsequently divided into pairs of (inspected_KCF, decoded_label). During this stage, we replace the misconfig_ID in each encoded label with the relevant misconfig_description sourced from the UMI. This adjustment provides the LLM with enhanced contextual insights, which significantly improves its response accuracy. The following division, as depicted in Figure 1C, aims to prevent the LLM from becoming overwhelmed by a multitude of misconfigurations within the same KCF. Furthermore, this splitting of labels facilitates the component's ability to deliver specific localization, reasoning, and remediation advice for each misconfiguration independently.

- *The Process of Adapting a Large Language Model (LLM) for*

Tasks Such as Localization, Reasoning, and remediation is crucial. The detection of various Kubernetes misconfigurations by GenKubeDetect is a highly specialized function for a general-purpose pretrained LLM, as demonstrated by the experimental findings in Section 4.3. Consequently, we opted to implement GenKubeDetect through fine-tuning methods, although these methods come with considerable overhead. In contrast, initial tests with GenKubeResolve indicated that fine-tuning is not necessary for the task of localization, reasoning, and remediation of identified KCF misconfigurations. Instead, we chose to adapt a pretrained LLM for this purpose by employing few-shot learning techniques, which involve providing the LLM with relevant examples for in-context learning, along with prompt engineering strategies. An example of this adaptation process is illustrated in Figure 1C.

The foundational model selected for GenKubeResolve is a pretrained Mistral LLM. This choice was made due to its exceptional recall and F1-score, as demonstrated in our initial experiments (Section 4.3). Additionally, Mistral is recognized as one of the leading open-source LLMs, and it offers the advantage of local usage at no cost. Utilizing Mistral locally ensures that KCF data remains secure, as it is not transmitted to external APIs, unlike web-based pretrained LLMs. Furthermore, local deployment enhances operational efficiency by

removing the delays and complexities associated with API interactions. To tailor Mistral for tasks related to localization, reasoning, and remediation, we refined a system prompt that directs Mistral to identify the precise line number of any detected misconfiguration, explain the rationale behind the detection, and provide recommendations for resolution. To further improve Mistral's capabilities, we supplemented this system prompt with several examples of KCFs, identified misconfigurations, and the anticipated outputs, employing a few-shot learning approach.

➢ *Data Preparation Module*

- *Developing a UMI:*

The development of a unified misconfiguration index (UMI) was initiated to address the diverse types of KCF misconfigurations identified by various tools. Research studies highlighted the necessity for standardizing misconfiguration labels. Our UMI is designed to include multiple unique misconfiguration class labels that are consistently formatted. This means that if a specific misconfiguration is labeled differently across various sources whether through differing textual descriptions or varying misconfiguration IDs, the UMI consolidates all these variations under a single identifier. This index not only aids our research efforts but also enhances the creation and comparison of effective solutions. The initial phase of UMI creation involved gathering a comprehensive set of policies and detection rules from three trusted KCF misconfiguration detection tools: Checkov, and Terrascan. Subsequently, we utilized a large language model for entity matching through prompt engineering (refer to Figure 1A. for details).

- *The Process of Gathering and Labelling*

KCFs is essential for training the LLMs utilized in GenKubeDetect and GenKubeResolve. We compile a comprehensive and varied collection of KCFs, which we initially annotate as *DSunlabeled* (refer to Figure 1A.). Subsequently, we develop *DSlabeled* by linking accurate misconfiguration labels to the KCFs in *DSunlabeled* using Checkov and Terrascan, which form the basis of the UMI. The decision to employ three detection tools stems from the constraints associated with RB detection methods. However, it is important to note that *DSlabeled* may still include some misconfigurations that remain undetected due to the limitations of the unified rule sets of these tools. In practice, when any of these tools identifies a misconfiguration for a KCF in *DSunlabed*, the identified misconfiguration is linked to its respective UMI misconfig_ID and is represented as a pair of values formats.

IV. METHODS AND RESULTS

➢ *Investigational Setup*

In our quantitative assessment, we employed the Python programming language as our primary tool. The Hugging Face transformers library was instrumental in training and deploying advanced large language models (LLMs), with the complete code available in the project's GitHub repository. For intensive computational tasks, particularly during the fine-tuning of LLMs, we utilized NVIDIA RTX 4090 and RTX 6000 Ada GPUs. To develop our Unique Misconfigurations Index (UMI), we initiated a focused web crawling of the online documentation, specifically the policy indexes, of three widely recognized open-source RB tools: Checkov, and Terrascan. Subsequently, we leveraged OpenAI's GPT-4 for entity matching, resulting in a UMI comprising 169 standardized unique misconfigurations, including one for KCFs where no misconfiguration was detected. To generate the dataset *DSunlabeled*, we gathered 276,520 unlabelled KCFs from the K8s Manifest Subset.

➢ *Data Division and Performance Metrics*

To evaluate the effectiveness of different KCF misconfiguration detectors, whether based on rules or large language models (LLMs), we conducted a random division of the dataset *DSlabeled* into predetermined segments: 80% for training, 10% for validation, and 10% for testing. The performance of these detectors was assessed using standard classification metrics, including precision, recall, and the F1 score, all weighted according to the frequency of various KCF misconfigurations present in the test set. For clarity, the terms precision, recall, and F1 used throughout this paper will refer to their weighted counterparts. The calculation of these classification metrics is based on the following definitions:

- True positives (TPs) are defined as instances where both the LLM and at least one rule-based (RB) tool successfully identified a misconfiguration.
- False positives (FPs) refer to instances where the LLM incorrectly identified a misconfiguration that was not detected by any RB tools. It is important to note that RB detectors may have limitations due to their predefined rules, which means that a false positive could represent a variant of a known misconfiguration that does not fully align with any existing detection rules.
- False negatives (FNs) are cases in which the LLM failed to identify a misconfiguration that was recognized by one of the RB tools.
- True negatives (TNs) occur when there is an agreement between the LLM and the RB tools regarding the lack of any misconfiguration.

➢ *Pretrained LLMs & Selecting the Modelling Architecture and Base Model*

The primary objective of the initial experiment was to investigate whether existing pretrained large language models (LLMs) are sufficient for detecting misconfigurations in Kubernetes Configuration Files (KCFs). To explore this, we tested four leading pretrained LLMs, Claude 2.1, GPT-4-Turbo, Mistral-7B and Gemini 1.0 Pro. These models are accessible through online chat or API calls but cannot be fine-tuned, necessitating the use of prompt engineering and few-shot learning techniques. We established a fixed few-shot training set by randomly selecting 9 labelled KCFs, ensuring that each KCF contained at least two misconfiguration labels to enhance diversity. This training set was uniformly applied across

all pretrained LLMs. Additionally, a consistent test set of 30 arbitrarily experimented KCFs, featuring the identical 39 misconfigurations as the training set, was utilized to assess the performance of the LLMs. As illustrated in Figure 2, the results from our experiments with the pretrained LLMs were disappointing, with the highest precision, recall, and F1 scores reaching approximately 0.79, 0.44, and 0.49, respectively. This suggests that detecting misconfigurations in KCFs may be too specialized for a general-purpose pretrained LLM, as indicated by previous research. Furthermore, the potential of LLMs in this domain may require more than a limited few-shot learning approach with a small set of labelled KCFs. Notably, Figure 2 shows that Mistral achieved a significantly higher recall compared to the other models, albeit with lower precision. In traditional machine learning contexts, this combination of low precision and high recall typically suggests an excessively low classification threshold, indicating that the model is over-predicting positive labels. However, in our LLM context, we observed that Mistral tends to produce responses that are 2-3 times longer.

Based on the findings from the previous experiment and the necessity for enhanced performance and wider coverage of misconfigurations, this subsection investigates the application of specialized open-source LLMs with various architectures. These LLMs have been optimized and fine-tuned specifically for the task of misconfiguration detection in KCFs. We evaluated three widely recognized LLM architectures: encoder-only, decoder-only, and encoder-decoder. We anticipated that the encoder-decoder architecture would be the most effective for our requirements, as.

- The encoder-only architecture is proficient in context comprehension,
- The decoder-only architecture excels in token generation, and
- The encoder-decoder architecture is capable of both understanding context and generating relevant tokens.

The studies have highlighted the advantages of the encoder-decoder architecture in tasks related to code understanding and completion; our scenario is analogous in that (1) the LLM must thoroughly comprehend a KCF, and (2) the LLM's output is utilized to produce suitable misconfiguration labels.

In our quest to determine the most effective LLM architecture and base model for detecting KCF misconfigurations, we conducted experiments with various open-source LLMs featuring the architectures (encoder-only, decoder-only, and encoder-decoder). To optimize efficiency, we fine-tuned each LLM using a stratified random sample of 69,000 labelled KCFs, which constitutes 31% of the total 2,30.000 labelled KCFs that fit within the token limits of the LLMs. Of this sample, 86% was designated for training purposes, while the remaining 14% was reserved for validation. For consistency in our experiments, we employed the same test set outlined in Section 4.2, consisting of 22,988 instances that adhere to the token limits. As illustrated in Table 1, Code Llama did not achieve convergence following fine-tuning; their outputs (the generated text serving as KCF labels) did not align with the intended misconfiguration labels, preventing us from calculating metrics for these models. Conversely, the other LLMs, primarily encoder-decoders as anticipated, exhibited adaptability to the task at hand: their outputs progressively aligned with the misconfiguration labels from the training set, achieving significant precision.

Among the encoder-decoder base models evaluated, we opted to proceed with CodeT5p 770M, which is specifically trained for code understanding and completion and is highly relevant to our field. This model demonstrated superior performance compared to others, including the larger CodeT5p 6B. Additionally, its smaller size makes it more resource-efficient and quicker to train. However, a limitation of CodeT5p 770M is its maximum token capacity of 508. In the dataset *DSlabeled*, approximately 84% of the 276,786 instances contain fewer than 508 tokens. We utilized these 2,32,500 KCFs and labels in our experiments.

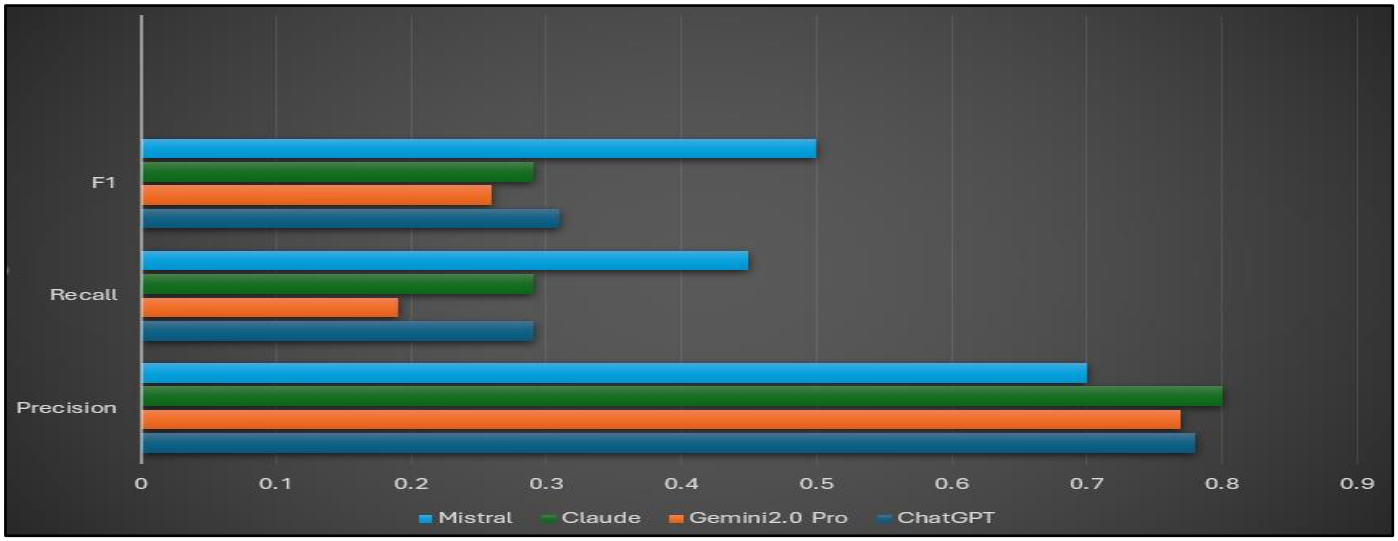

Fig 2 Evaluating the Capabilities of Pretrained Large Language Models for Misconfiguration Detection Utilizing Few-Shot Learning Techniques.

Table 1 Evaluation of Performance Using Various LLM Architectures.

Base model	Precision	Recall	F1	Architecture
CodeT5p 6B	0.903±0.067	0.380±0.014	0.544±0.027	Encoder-decoder
Code Llama	—	—	—	Decoder-only
LED	0.974±0.033	0.264±0.249	0.369±0.257	Encoder-decoder
CodeGen	0.981±0.041	0.575±0.197	0.706±0.169	Decoder-only
CodeT5p 770M	0.986±0.022	0.998±0.018	0.993±0.018	Encoder-decoder

➢ *Identifying Suitable Data Sources for the Purpose of fine-Tuning*

To determine the most effective combination of data sources for fine-tuning, we conducted experiments using three distinct data sources:

- *Labelled KCFs:*

In line with standard classification tasks, model training necessitates labeled instances, which in our scenario consist of KCFs paired with their corresponding misconfigurations.

- *Unlabelled KCFs:*

This data source was incorporated to potentially enhance the model's capability to grasp the typical structures and patterns found in KCFs overall.

- *Free Text:*

This dataset, comprising around 229 documents collected through targeted web crawling, includes a variety of K8s-related materials, such as the CIS Benchmarks, authoritative literature, and official documentation. This dataset is accessible in our GitHub repository and was included to potentially expand GenKubeDetect's understanding of K8s-specific terminology and concepts. To achieve this, we investigated domain adaptation strategies utilizing contrastive learning methods, including techniques like next sentence prediction and masking.

To determine the most effective data sources and training pipeline for enhancing model performance through fine-tuning, we developed four separate models based on the CodeT5p 770M base model. In each configuration, labelled KCFs were utilized in the final fine-tuning phase. The findings illustrated in Table 2 indicate that the approach of first employing unlabelled KCFs followed by labelled KCFs yielded the highest performance metrics, including maximum mean and/or minimum standard deviation for all the scores. This implies that initial exposure to the general framework of KCFs, succeeded by specific pairs of KCFs and their corresponding misconfiguration labels, slightly enhances the model's capability to identify misconfigurations compared to fine-tuning with labelled KCFs alone. On the other hand, incorporating free text seemed to slightly hinder performance, likely due to its unstructured format conflicting with the structured inputs expected by the LLM. Consequently, free text was excluded from our experiment, which also significantly reduces the training time required for fine-tuning the model. Importantly, the combination of unlabelled KCFs with free text led to better performance than fine-tuning without unlabelled KCFs. Nevertheless, it is clear that optimal performance, with minimal input overhead, is achieved when fine-tuning is conducted solely with KCFs, both unlabelled and labelled.

➢ *Analysis of Existing RB Tools.*

Our architecture selection involved the encoder-decoder framework, with CodeT5p 770M serving as the base model, and we incorporated both unlabelled and labelled KCFs as our data sources. We then carried out experiments aimed at hyperparameter tuning, specifically targeting the learning rate and weight decay. The LoRA adaptation was confined to 13%, with r set to 130 and LoRA_alpha at 256. For a detailed overview of the hyperparameters we adjusted to measure the effectiveness of our approach, please refer to our GitHub repository.

By applying the optimized hyperparameters and utilizing the same training set, which includes both KCFs, we retrained the LLM of GenKubeDetect and evaluated its performance on the test set. The findings from our evaluation, alongside those of industry-standard RB tools for detecting KCF misconfigurations Checkov, and Terrascan are summarized in Table 2. To ensure an equitable comparison, each RB tool was assessed based on its ability to detect the specific subset of misconfigurations it is intended to identify. This strategy allows us to showcase the strengths of each tool without penalizing them for undetected misconfigurations that lie outside their configured rules.

Table 2 Analysis of Different Tools for Detecting KCF Misconfigurations: GenKubeDetect Compared to Standalone Industry-Standard RB Tools and a Combination of These RB Tools.

Tool	#Labels	Precision	Recall	F1
GenKubeDetect	170	0.992±0.030	0.998±0.025	0.993±0.028
Checkov	120	1.0±0.0	0.850±0.317	0.865±0.318
Terrascan	42	1.0±0.0	0.588±0.470	0.598±0.473
RB-Ensemble	170	1.0±0.0	1.0±0.0	1.0±0.0

Table 2 illustrates that GenKubeDetect outperforms each RB tool in terms of metrics, while maintaining a precision level that is nearly identical. Additionally, GenKubeDetect's capacity to identify a total of 170 KCF misconfigurations surpasses that of any individual RB tool assessed, highlighting its thoroughness and versatility. In

contrast to RB tools that are limited to a narrow range of specific misconfigurations, achieving perfect precision but low recall, GenKubeDetect effectively detects a wide array of both common and rare misconfigurations across diverse scenarios, demonstrating both high precision and high recall.

- *Analysis of GenKubeDetect Errors*

To enhance our understanding of the false positives (FPs) reported by GenKubeDetect, as illustrated in Table 3, we undertook a manual inspection with the guidance of a Kubernetes security expert. The investigation uncovered that FP misconfigurations were identified in 558 Kubernetes Configuration Files (KCFs) within the test dataset, acknowledging that a single KCF may contain multiple misconfigurations. Given the limitations of manually reviewing all KCFs, we randomly selected a sample of 100, which revealed 490 FPs. Our K8s expert meticulously analyzed these KCFs and concluded that 411 out of 490, equating to 84%, were true positives (TPs). These results imply that GenKubeDetect's precision is indeed higher than what is presented in Table 3. Additionally, this indicates that GenKubeDetect has successfully broadened its knowledge base, offering accurate insights that surpass the conventional standards set by a trio of industry-standard rule-based tools. Specifically, GenKubeDetect excelled in identifying misconfigurations that were overlooked by the RB tools, as they represented variations of detection rules that did not fully align with any established coded rule.

To analyze the learning curve of GenKubeDetect in relation to the size of its training dataset, we progressively expanded the training set, varying the number of misconfiguration instances from 50 to 2,000 for each misconfiguration label. In instances where the number of available misconfigurations was limited, we utilized all existing instances. Each iteration involved training GenKubeDetect from the ground up, employing the optimized hyperparameters outlined in this section, and assessing its performance with the same test set utilized previously.

Validating the output of GenKubeResolve for each detected misconfiguration manually is a time-intensive endeavor that necessitates the involvement of an expert. As a result, a Kubernetes security specialist evaluated GenKubeResolve's capabilities on a random sample of 25 identified KCF misconfigurations, with each case requiring approximately 5 minutes for validation. The expert affirmed that GenKubeResolve excels in its explanatory functions, achieving a flawless score of 25 out of 25 for both its explanations and remediation advice. In terms of localization, it was noted that in 16 of the 25 cases, GenKubeResolve correctly identified the absence of necessary lines in the KCFs, which hindered its ability to provide line numbers. In the remaining 9 cases, it successfully identified the issues along with the correct line numbers.

V. DISCUSSION

This research yielded several important findings. We discovered that employing basic prompt engineering and few-shot learning methods to modify pretrained large language models (LLMs) for KCF misconfiguration detection leads to suboptimal results. However, for the specialized task of multi-label classification within (semi-structured) KCFs, a well-optimized and finely-tuned LLM can deliver nearly flawless performance, achieving precision that meets industry benchmarks and demonstrating enhanced recall.

Furthermore, it appears that by utilizing an appropriate LLM architecture, a base model relevant to the domain, and well-optimized hyperparameters, GenKubeDetect can attain satisfactory performance with relatively small amounts of training data. This is evident in the initial fine-tuning phase, where the learning curve demonstrates high precision.

One of the more counterintuitive insights we discovered is that during the fine-tuning process of GenKubeDetect, prioritizing KCFs alone results in the best precision, recall, and F1 scores. Interestingly, integrating free-text information about K8s with KCFs does not improve these metrics and may slightly hinder performance. This finding implies that (1) when the LLM is designed to interpret structured data (like KCFs), there is no advantage in training it with unstructured data; (2) fewer inputs (specifically KCFs) are adequate for generating accurate results.

VI. CONCLUSION

This research focuses on the role of large language models (LLMs) in enhancing the security of Kubernetes (K8s) systems. GenKubeSec offers more than just detection of various misconfigurations in Kubernetes Configuration Files (KCFs); it identifies their precise locations within the KCFs, provides clear reasoning that aids KCF developers in their understanding, and delivers actionable remediation recommendations. By not utilizing external APIs, GenKubeSec ensures that KCFs are safeguarded from online exposure, which enhances security and reduces associated costs. In contrast to previous investigations, we have developed a comprehensive LLM-based strategy and quantitatively assessed its effectiveness.

Future initiatives in this sector might focus on (1) the automatic production of a KCF devoid of misconfigurations to rectify the identified problems, and (2) enhancing GenKubeSec's functionality to effectively prioritize the detected misconfigurations according to their severity levels.

REFERENCES

[1]. Md Zahangir Alom, Tarek M Taha, Chris Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Mahmudul Hasan, Brian C Van Essen,

Abdul AS Awwal, and Vijayan K Asari. 2019. A state-of-the-art survey on deep learning theory and architectures. electronics 8, 3 (2019), 292.

[2]. Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7B. arXiv preprint arXiv:2310.06825 (2023)

[3]. Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. Advances in neural information processing systems 33 (2020), 1877–1901.

[4]. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. Advances in neural information processing systems 30 (2017).

[5]. Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. [n. d.]. Language Models are Unsupervised Multitask Learners. ([n. d.]).

[6]. Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. arXiv preprint arXiv:2308.12950 (2023).

[7]. Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. 2020. Big bird: Transformers for longer sequences. Advances in Neural Information Processing Systems 33 (2020)

[8]. Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2022. CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis. In The Eleventh International Conference on Learning Representations

[9]. Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The Long-Document Transformer. arXiv:2004.05150 (2020).

[10]. Yue Wang, Hung Le, Akhilesh Deepak Gotmare, Nghi DQ Bui, Junnan Li, and Steven Hoi. 2023. CodeT5+: Open Code Large Language Models for Code Understanding and Generation. In The 2023 Conference on Empirical Methods in Natural Language Processing.

[11]. Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692 (2019).

[12]. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018).

[13]. Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. Journal of machine learning research 21, 140 (2020), 1–67.

[14]. Ralph Peeters and Christian Bizer. 2023. Entity Matching using Large Language Models. arXiv preprint arXiv:2310.11244 (2023)

[15]. Sundar Pichai and Demis Hassabis. 2023. Introducing Gemini: our largest and most capable AI model. Google. Retrieved December 8 (2023), 2023.

[16]. Tongtong Wu, Massimo Caccia, Zhuang Li, Yuan-Fang Li, Guilin Qi, and Gholamreza Haffari. 2021. Pretrained language model in continual learning: A comparative study. In International conference on learning representations.

[17]. Hanqing Zhang, Haolin Song, Shaoyu Li, Ming Zhou, and Dawei Song. 2023. A survey of controllable text generation using transformer-based pre-trained language models. Comput. Surveys 56, 3 (2023), 1–37.

[18]. Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2020. Transformers: State-of-the-art natural language processing. In Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations. 38–45

[19]. Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. arXiv preprint arXiv:2303.08774 (2023).