Hochschule Karlsruhe
University of Applied Science

Fakultät für Informatik und Wirtschaftsinformatik
Wirtschaftsinformatik

Masterthesis

Generative AI for Security Automation in Hyperscale Cloud Platforms

Von	Daniel Vera Gilliard
Matrikelnr.	72988
Arbeitsplatz	Karlsruhe
Erstbetreuer	Prof. Dr. rer. nat. Udo Müller
Zweitbetreuer	Prof. Dr. Andreas P. Schmidt
Abgabetermin	September 30, 2025

Karlsruhe, September 8, 2025

Vorsitzender des Prüfungsausschusses

Declaration of Authorship

I, Daniel VERA GILLIARD, in lieu of an oath that I have written the Master's thesis presented here independently and exclusively using the literature and other aids provided. The thesis has not been submitted in the same or a similar form to any other examination authority for the award of an academic degree.

Signed:

Date:

Declaration on the Use of Generative AI

I, Daniel VERA GILLIARD, hereby declare that generative artificial intelligence (AI) was employed as a writing assistant in the development of this manuscript. The use of these tools was exclusively for linguistic enhancement, such as refining sentence structure, correcting grammar, and improving overall style. The conceptual framework, original ideas, research methodology, data analysis, and final conclusions presented in this work are the product of my own intellectual effort. I have critically reviewed, edited, and validated all content to ensure its accuracy and originality, and I bear complete responsibility for the entirety of this thesis.

Signed: _____

Date: _____

"*Thanks to my solid academic training, today I can write hundreds of words on virtually any topic without possessing a shred of information, which is how I got a good job in journalism.*"

Dave Barry

HOCHSCHULE KARLSRUHE

Abstract

Faculty Name
Business Information Systems

Master of Business Information Systems

Generative AI for Security Automation in Hyperscale Cloud Platforms

by Daniel VERA GILLIARD

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too...

Acknowledgements

The acknowledgments and the people to thank go here, don’t forget to include your project advisor. . .

Contents

List of Figures

List of Tables

Listings

Glossary

cloud-native An approach to building and running applications that exploits the advantages of the cloud computing delivery model. Cloud-native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. 34

false positive An alert or detection that incorrectly identifies normal, benign activity as malicious or threatening. High false positive rates can overwhelm security teams and reduce the effectiveness of security systems. 37, 46

orchestration The automated arrangement, coordination, and management of complex computer systems, middleware, and services. In cloud computing, this often refers to managing containers, services, and infrastructure. 35, 38, 43

Rego A high-level declarative language used by the Open Policy Agent (OPA) to express policies over complex data structures, such as JSON.. 35, 37–40, 46–48

security automation The use of technology to perform security tasks with minimal human intervention. It includes automated threat detection, incident response, compliance monitoring, and vulnerability management. 38

Acronyms

AI Artificial Intelligence. 4–8, 10, 11, 13–18, 24–26, 30, 31, 35, 38, 40, 44, 48

AI RMF Artificial Intelligence Risk Management Framework. 5–8, 14–16

API Application Programming Interface. 7, 26, 35, 37, 39, 40, 42, 43, 46, 48

APP Australian Privacy Principles. 7

AWS Amazon Web Services. 5, 8, 35–37, 39, 42, 43, 46–48

CI/CD Continuous Integration/Continuous Deployment. 21, 23, 25, 31, 32, 34, 35, 37, 38, 46–48

CoE Center of Excellence. 6–8

DevOps Development and Operations. 21

DevSecOps Development, Security, and Operations. 10, 11, 34, 46, 48

GCP Google Cloud Platform. 36, 39

GDPR General Data Protection Regulation. 12

GenAI Generative Artificial Intelligence. 3–8, 10–18, 23–26, 30–39, 46–48

GitHub GitHub platform. 35, 38, 46

HITL Human-in-the-Loop. 30, 31, 33–35, 37, 48

IaC Infrastructure-as-Code. 18, 21, 23, 24, 26, 31, 33, 35, 37, 38, 47

IAM Identity and Access Management. 38

IDPS Intrusion Detection and Prevention System. 10

IP Internet Protocol. 8

ISO International Organization for Standardization. 7

IT Information Technology. 9

JSON JavaScript Object Notation. 35

KB Knowledge Base. 38, 39, 46, 48

KMS Key Management Service. 38

LLM Large Language Model. 6–8, 10, 12, 14, 15, 18, 21, 24–27, 33, 37, 39, 40, 42, 47, 48

MLOps Machine Learning Operations. 7, 8, 25, 35

MTTD Mean Time to Detect. 11

MTTR Mean Time to Resolve. 11

NIST U.S. National Institute of Standards and Technology. 5–8, 14–16

OPA Open Policy Agent. 35, 37, 39, 42, 43, 46, 47

OWASP Open Web Application Security Project. 6–8

PaC Policy-as-Code. 21, 25, 31, 35

PG Policy Generation. 33, 46

PRISMA Preferred Reporting Items for Systematic Reviews and Meta-Analyses. 11

RAG Retrieval-Augmented Generation. 7, 10, 26, 33, 37–39, 46, 48

RMF Risk Management Framework. 14, 15

S3 Simple Storage Service. 37, 38

SAIF Secure AI Framework. 6–8

SAST static analysis security testing. 23, 34, 37–39, 46

SDK Software Development Kit. 39

SOC Security Operations Center. 5

SP System Prompt. 37

SQL Structured Query Language. 12

SRM Shared Responsibility Model. 8, 38

Terraform Terraform infrastructure as code. 33–35, 37–39, 46–48

YAML YAML Ain't Markup Language. 9, 35

ZTA Zero Trust Architecture. 15

For/Dedicated to/To my. . .

Chapter 1

Introduction

1.1 Motivation and Problem Statement

TBD

1.2 Research Objectives and Questions

This thesis explores how Generative AI (GenAI) can help solve the security challenges in modern cloud environments. The research focuses on using GenAI to automate security tasks and asks the following central question:

How can Generative AI technologies be effectively leveraged to automate security operations across hyperscale cloud platforms?

To comprehensively answer this overarching question, the research is decomposed into four distinct but interrelated sub-questions. The first sub-question explores the effectiveness and automation of GenAI for security policy generation: *How can Generative AI technologies be effectively leveraged to automate security policy generation and management across hyperscale cloud platforms?* The second investigates the architecture and orchestration needed for trust and accuracy: *What specific architectural patterns and validation mechanisms are required to ensure trust, accuracy, and effective multi-cloud orchestration in GenAI-driven security automation?* The third seeks to determine how to measure and validate the effectiveness of this automation: *How can the effectiveness of GenAI-driven security automation be quantitatively measured and validated, particularly in terms of accuracy, reliability, and efficiency gains?* Finally, the fourth sub-question explores the role of human oversight: *What is the optimal balance between automation and human oversight to maximize security outcomes and mitigate the risks of GenAI-driven policy generation?*

These questions guide the structure of the thesis. The current state of the art is examined in Chapter 2. The conceptual framework to address the question of architecture and orchestration is detailed in Chapter 4. The practical implementation and validation of the framework, which addresses the remaining questions, are described in Chapter 5, and the results are evaluated in Chapter 6.

By methodically addressing these questions, this thesis aims to provide a comprehensive, empirically grounded contribution to the field of automated cloud security.

1.3 Contribution and Significance

TBD

1.4 Methodology Overview

TBD or separate section?

1.5 Outline of the Thesis

TBD

Chapter 2

Background and Related Work

This chapter provides the foundational knowledge required to understand the core concepts of this thesis. It begins by exploring the fundamental principles of cloud security and generative AI. Subsequently, it reviews the current landscape of major cloud providers and their security ecosystems. The chapter culminates in a comprehensive review of the state-of-the-art literature, identifying key research trends and existing gaps in the field of GenAI-driven security automation.

2.1 Foundational Concepts in Cloud Security

This section provides an overview of the fundamental concepts in cloud security, which form the basis for the subsequent discussion on security automation.

TBD

2.2 Foundational Concepts in Generative AI

This section introduces the core concepts of Generative AI, setting the stage for its application in the context of cloud security.

TBD

2.3 State of Cloud Provider Ecosystems

This section reviews the current landscape of major cloud providers, highlighting their security services and the evolving ecosystem.

TBD

2.4 Literature State of the Art

The convergence of Generative Artificial Intelligence (GenAI) and hyperscale cloud platforms presents a rapidly evolving frontier for cybersecurity. As organizations increasingly rely on complex cloud environments, the scale and sophistication of threats necessitate advanced automation capabilities. GenAI offers promising avenues for enhancing security posture through intelligent automation, but its integration also introduces challenges and risks. A comprehensive understanding of the existing research landscape is therefore essential to identify established practices, emerging trends, and critical gaps in knowledge.

This literature review synthesizes current academic and industry contributions pertinent to the application of GenAI for security automation within hyperscale and

multi-cloud contexts. It begins by outlining the methodology employed to select and analyze relevant works. Subsequently, the review dives into several key thematic areas: the evolution from traditional security methods to Artificial Intelligence (AI)-driven approaches, specific frameworks for scoping and managing GenAI security implementations, architectural patterns and techniques for security automation, a critical examination of the unique security risks associated with GenAI itself, risk management frameworks and the ongoing crucial discussion regarding the necessary balance between automation and human oversight.

By examining these facets, this review aims to provide a foundational understanding of the state-of-the-art, highlighting both the potential of GenAI in cloud security automation and the significant considerations that must be addressed for its responsible and effective deployment. This synthesis will builds the basis for the subsequent research presented in this thesis.

2.4.1 Methodology

This literature review followed a structured approach to identify relevant publications, focusing on peer-reviewed articles addressing GenAI applications in hyperscale cloud security published primarily within the last five years. The search utilized academic databases with key search terms related to generative AI, cloud security automation, hyperscale platforms, and multi-cloud orchestration.

Papers were selected based on their relevance to:

- GenAI applications specifically in cloud security contexts
- Hyperscale or multi-cloud environments
- Technical solutions for security automation
- Empirical evidence or theoretical frameworks with substantial methodological rigor

The selection process involved initial screening of titles and abstracts followed by full-text review of promising papers. The analysis employed a thematic approach, identifying recurring concepts, methodological approaches, and gaps in existing research. Particular attention was paid to identifying the theoretical foundations underpinning GenAI applications in security contexts, empirical evidence of effectiveness, and limitations of current approaches.

2.4.2 AI-Driven Security Approaches

The landscape of cloud security is undergoing a significant transformation, shifting from traditional, often reactive methods towards more proactive and adaptive strategies powered by AI, particularly GenAI. This evolution marks a move beyond basic anomaly detection towards sophisticated security postures capable of learning from and responding dynamically to novel threat vectors in complex cloud environments.

Foundational work by Khanna [1] explores the integration of GenAI into cloud security, outlining its core applications. According to Khanna, modern GenAI implementations focus on key capabilities such as processing vast amounts of data (e.g., log entries, network packets) for advanced anomaly detection and threat intelligence, enabling automated response mechanisms that dynamically adjust security protocols, and facilitating predictive security measures to forecast potential vulnerabilities. While highlighting these advancements, Khanna also acknowledges inherent

challenges, including the need for large datasets and mitigating potential adversarial manipulation.

Building upon these foundational capabilities, [2] also concludes that the integration of GenAI represents a significant leap beyond conventional rule-based security systems. Its research indicates that GenAI enhances security automation, particularly within multi-cloud and hybrid architectures. It allows systems to adapt infrastructure dynamically in response to varying traffic patterns and implements AI-powered defenses against continuously evolving cyber threats. This adaptive capability directly addresses persistent challenges in cloud security related to optimizing workload distribution, ensuring performance, and managing costs effectively.

Furthermore, recent studies underscore the practical impact of integrating GenAI with established cloud-native security tools. Patel et al. [3] demonstrate how layering GenAI onto platforms like Amazon Web Services (AWS) GuardDuty and Google Cloud Security Command Center significantly boosts automated threat detection, enables real-time incident response, and improves comprehensive vulnerability management across distributed cloud infrastructures. Their work provides empirical evidence, citing examples like Netflix and JPMorgan Chase, which reported measurable improvements in detection accuracy and notable reductions in security incidents following the adoption of GenAI-driven security automation strategies [3]. This convergence of GenAI with existing security frameworks highlights its potential to transform Security Operations Centers (SOCs) by enhancing both efficiency and effectiveness.

2.4.3 GenAI Security Frameworks

The rapid integration of GenAI into various domains, including security automation within hyperscale cloud platforms, necessitates robust frameworks to govern its development, deployment, and operation securely. Unlike traditional software, GenAI systems introduce unique risks stemming from their data dependency, complexity, potential for emergent behaviors, and socio-technical nature [4]. These risks include prompt injection, data leakage through model inversion or training data extraction, adversarial attacks, generation of harmful or biased content, and insecure code generation [5, 6]. Consequently, a multi-faceted approach to security is required, encompassing foundational risk management principles, organizational governance structures, technical control implementation, and context-specific guidance. This subchapter reviews several key frameworks and guides that collectively address the challenge of securing GenAI systems.

A foundational element in managing AI risks is provided by the Artificial Intelligence Risk Management Framework (AI RMF) 1.0 developed by the U.S. National Institute of Standards and Technology (NIST) [4]. This voluntary, non-sector-specific framework aims to help organizations manage AI risks and promote trustworthy and responsible AI development and use. It is structured around four core functions: GOVERN, MAP, MEASURE, and MANAGE. The GOVERN function is cross-cutting, establishing a risk management culture and processes throughout the organization. MAP involves contextualizing risks and understanding potential impacts. MEASURE employs qualitative and quantitative tools to analyze, assess, and track AI risks. MANAGE focuses on prioritizing and acting on risks based on the previous functions [4]. The NIST AI RMF emphasizes characteristics of trustworthy AI systems, including validity and reliability, safety, security and resilience, accountability and transparency, explainability and interpretability, privacy-enhancement, and fairness with

harmful bias managed [4]. It acknowledges the challenges specific to AI risk management, such as difficulties in risk measurement (especially with third-party components and emergent risks), defining risk tolerance, prioritizing risks, and integrating AI risk management into broader enterprise strategies [4].

Building upon similar principles but offering a perspective from a major cloud provider, Google's Secure AI Framework (SAIF) presents a conceptual framework inspired by security best practices applied to software development, adapted for AI-specific risks [6]. SAIF proposes six core elements for secure AI systems[6]:

1. Expand strong security foundations to the AI ecosystem by applying and adapting existing controls.
2. Extend detection and response to include AI-specific threats and outputs.
3. Automate defenses using AI itself where appropriate, while keeping humans in the loop.
4. Harmonize platform-level controls to ensure consistency and prevent fragmentation.
5. Adapt controls with faster feedback loops, incorporating insights from red teaming and novel attack awareness.
6. Contextualize AI system risks within surrounding business processes, including model risk management and shared responsibility.

SAIF advocates a practical implementation approach involving understanding the specific use case, assembling a multidisciplinary team, providing an AI primer for all stakeholders, and applying the six core elements iteratively [6]. Like the NIST AI RMF, SAIF highlights the socio-technical nature of AI risks and the importance of context.

While NIST and Google provide overarching frameworks, securing GenAI, particularly Large Language Models (LLMs), requires specific organizational structures and practices. The LLM and Generative AI Security Center of Excellence (CoE) Guide from OWASP addresses this by outlining how to establish a dedicated CoE [7]. The primary objective of such a CoE is "to develop and enforce security policies and protocols for generative AI applications, facilitate cross-departmental collaboration to harness expertise from various fields, educate and train teams on the ethical and secure use of generative AI technologies, and serve as an advisory body for AI-related projects and initiatives within the organization" [7, p.4]. This guide emphasizes the necessity of a multidisciplinary team, bringing together expertise from Cybersecurity, AI/ML Development, IT Operations, Legal, Compliance, Ethics, Governance, Risk Management, Data Science, and various other user groups[7]. Establishing clear objectives, Key Performance Indicators, roles, and responsibilities is crucial for the CoE's success. The guide suggests a phased implementation (Planning, Integration, Operationalization, Evaluation) and highlights the importance of leveraging both internal and external expertise to address challenges like communication barriers, resistance to change, and skill gaps [7]. The CoE structure directly supports the GOVERN function described in the NIST AI RMF and aligns with SAIF's recommendation to assemble a cross-functional team.

To translate high-level governance and risk management principles into concrete verification steps, the Open Web Application Security Project (OWASP) LLM Applications Cybersecurity and Governance Checklist provides a practical, actionable tool

[8]. Derived from the well-known OWASP Top 10 for LLM Applications project, this checklist offers a structured approach to assessing the security posture of LLM-based systems. It covers a wide array of control areas critical for LLM security, extending beyond purely technical vulnerabilities to encompass essential governance aspects [8]. Key domains addressed include Input Validation and handling, Output Encoding and Handling, Access Control and Authorization, Data Privacy and Confidentiality, Model Training and Fine-tuning Security, Application Programming Interface (API) and Integration Security, robust Logging and Monitoring, Incident Response Planning, and overall Governance, Risk, and Compliance[8]. The checklist serves as a valuable resource for development teams, security assessors, and governance bodies to systematically identify potential weaknesses, guide the implementation of specific security controls, and perform gap analyses against established best practices [8]. In the context of broader frameworks, this checklist acts as a practical instrument for the MEASURE and MANAGE functions outlined in the NIST AI RMF [4], providing specific checks aligned with the risks highlighted by SAIF [6] and the technical controls advocated by SecGenAI [5]. It furnishes the CoE[8] with a concrete tool for enforcing security policies and protocols. Regarding a specific GenAI architecture, the SecGenAI framework focuses on enhancing the security of cloud-based GenAI applications, particularly Retrieval-Augmented Generation (RAG) systems, within the context of Australian critical technologies [5]. SecGenAI adopts an end-to-end perspective covering Functional, Infrastructure, and Governance requirements. Functionally, it analyzes RAG components and identifies root causes for security concerns like injection attacks, data leakage, and model inversion [5]. It proposes specific countermeasures such as input validation, robust access controls, data protection techniques, and model security measures[5]. On the infrastructure side, SecGenAI details requirements for sandboxing, secure database connections, network segmentation, external attack prevention, and disaster recovery within a cloud environment[5]. Governance requirements emphasize alignment with Australian AI Ethics Principles and Australian Privacy Principles (APP), advocating for fairness, accountability, traceability, data protection, regular audits, reliability, transparency, and compliance, structured using the International Organization for Standardization (ISO) 38500 Evaluate-Direct-Monitor cycle [9]. SecGenAI explicitly incorporates the Shared Responsibility Model, detailing Cloud Service Provider and customer obligations for GenAI security[5]. This framework provides a detailed blueprint for securing a specific, common GenAI patters by integrating technical and governance controls, reflecting a practical application of the principles found in NIST AI RMF and SAIF.

The successful implementation of these security frameworks inherently depends on the underlying platform architecture. The paper "Integration patterns in unified AI and cloud platforms" provides context by reviewing how AI, Machine Learning Operations (MLOps), workflow orchestration, and data processing converge in cloud-native environments[10]. It highlights the importance of MLOps frameworks for lifecycle management, workflow orchestration engines for process automation, and robust data processing systems as core components [10]. Security considerations must be embedded within these components and their integration patterns. For instance, securing data pipelines within MLOps, ensuring secure communication in federated learning setups, or implementing access controls within workflow orchestration are crucial [5, 6, 10]. The paper implicitly underscores that security cannot be an afterthought but must be integrated into the design and automation processes of these unified platforms, aligning with the principles of secure-by-design advocated by frameworks like SAIF and SecGenAI. Facilitating this integration within a specific

hyperscale cloud platform, the AWS GenAI Security Scoping Matrix serves as a practical aid for organizations utilizing AWS[11]. This matrix is designed explicitly to help customers navigate the complexities of the Shared Responsibility Model (SRM) as it applies to GenAI workloads deployed on AWS. It systematically maps common GenAI architectural components and layers against key security domains[11]. For each intersection of a GenAI component and a security domain, the matrix clarifies whether the responsibility for implementing controls lies primarily with AWS, the customer, or is shared between them [11]. This structured delineation is crucial for organizations to understand their specific security obligations when building and operating GenAI systems on AWS. Furthermore, the matrix guides customers in identifying and scoping the relevant AWS security services needed to fulfill their responsibilities [11]. It acts as a translator, converting the high-level principles of frameworks like NIST AI RMF [4] and SAIF [6], and the specific control requirements suggested by SecGenAI [5] or the OWASP Checklist [8], into actionable configurations and service selections within the AWS ecosystem. It directly supports the practical implementation of the SRM, a concept emphasized across multiple frameworks [4–6, 8], thereby enabling organizations to effectively manage risks within their AWS environment.

In summary, these frameworks and guides offer a layered approach to GenAI security. The NIST AI RMF provides a foundational, risk-based structure and defines trustworthiness. Google's SAIF offers a conceptual implementation path with core security elements, emphasizing adaptation and integration. The OWASP LLM CoE Guide focuses on the essential organizational and collaborative structures needed for effective governance. SecGenAI provides a detailed, integrated blueprint for securing a specific architecture, demonstrating how broader principles can be applied in context, including alignment with regional regulations. Practical tools like the OWASP LLM Checklist and provider-specific resources like the AWS Scoping Matrix aid in the MEASURE and MANAGE functions by providing concrete checks and configurations. The insights from [10] remind us that these security measures must be seamlessly integrated within the complex fabric of unified AI and cloud platforms, particularly within MLOps and automation workflows, to be effective. The recurring theme of the SRM across multiple frameworks [4–6, 8] highlights the collaborative nature of securing GenAI in cloud environments. Collectively, these resources provide a comprehensive toolkit for organizations aiming to leverage GenAI for security automation and other critical tasks on hyperscale cloud platforms, enabling them to manage risks effectively and build trustworthy AI systems.

2.4.4 Approaches for Automated Cloud Security

This subsection details specific technical approaches and architectural patterns crucial for enabling automated cloud security. It reviews research on unified platforms integrating AI and MLOps across multi-cloud environments, techniques for automated policy orchestration in complex Kubernetes setups, and the application of digital twins for robust security policy validation. These approaches represent concrete mechanisms for realizing the potential of automation in dynamic cloud infrastructures. For organizations operating containerized workloads across multiple clusters, particularly in multi-domain architectures involving different administrative entities, research from 2023 proposes an automated approach for generating network security policies in Kubernetes deployments[12]. Manually configuring security in such environments is complex, often leading to inconsistencies between policies defined in different clusters and requiring domain administrators to possess knowledge about other domains' configurations (like service locations or Internet Protocol (IP) addresses), which is not

always feasible[12]. This approach addresses two critical challenges in multi-cluster security: reducing the configuration errors commonly made by human administrators and creating transparent cross-cluster communications without requiring extensive information sharing between domains[12].

The proposed solution involves a top-level entity named the "Multi-Cluster Orchestrator"[12]. This orchestrator acts as a central management point, receiving inputs from managers of different domains[12]. These inputs include:

- A description of each domain's structure[12].
- High-level security requirements specifying allowed communications[12]. These requirements can be defined using an extended YAML Ain't Markup Language (YAML) syntax with special labels that abstract away low-level details[12].

Based on these inputs, the Multi-Cluster Orchestrator refines the high-level requirements into concrete configurations through a two-step process[12]:

1. It generates a "Global Configuration" that tracks communication pairs between services and required links between clusters, optimizing the overall cluster mesh setup[12].
2. It derives "Single Configurations" for each individual cluster, containing the specific parameters needed to connect the cluster to the mesh, the Kubernetes Network Policies to enforce the desired security rules, and commands to create local service entries that enable transparent name resolution for services located in external clusters[12].

The implementation, known as Multi-Cluster Orchestrator, demonstrates how automated policy generation can improve security consistency across distributed environments while reducing the cognitive load on security administrators by handling the complexity of multi-domain interactions transparently[12]. This research is particularly relevant for hyperscale cloud platforms and organizations that utilize container orchestration technologies like Kubernetes to manage numerous workloads across multiple clusters, potentially spanning different regions, availability zones, or administrative boundaries[12].

Another approach to security automation in the context of policies involves the use of digital twins for validating security policies before deployment in production environments[13]. This approach utilizes an emulation system specifically designed to create high-fidelity digital replicas of target Information Technology (IT) infrastructures[13]. These digital twins replicate key functionalities of the corresponding physical or virtual systems, allowing security teams to play out complex security scenarios, such as intrusion attempts and defense responses, within a safe and controlled environment[13]. This capability avoids impacting operational workflows on the real-world infrastructure[13]. The digital twin approach, following the research by Hammar and Stadler, enables a closed-loop learning process for crafting and refining security policies[13]. It starts with generating a digital twin of the target infrastructure. This is achieved using an emulation system constructed with virtualization tools like Docker containers, alongside virtual links and switches[13]. Within this digital twin, various security scenarios involving emulated attackers, defenders, and client populations are executed[13]. During these runs, monitoring agents collect detailed system measurements and logs, channeling this data through pipelines for analysis[13]. The gathered data and statistics are then used to build simulations[13]. Reinforcement learning techniques are applied to these simulations to learn potentially optimal security policies[13]. Finally, the performance of these learned policies is rigorously

evaluated back in the digital twin, allowing for validation and further iteration[13]. This methodology provides continuous, iterative feedback and improvement cycles, as the results from validation can inform further scenario runs and learning phases, enhancing policy effectiveness over time[13]. The authors demonstrate this by applying the approach to an intrusion response scenario, showing that the digital twin provided the necessary evaluative feedback to learn near-optimal policies that outperformed baseline systems like the SNORT Intrusion Detection and Prevention System (IDPS)[14]. This represents a significant advancement in validation mechanisms, particularly relevant for potentially complex GenAI-driven security automation strategies, by bridging the gap between simulation-based learning and real-world applicability[13].

Regarding policies, ensuring the trustworthiness and accuracy of GenAI-generated security policies and responses remains a significant challenge. The already mentioned SecGenAI framework demonstrates how advanced machine learning techniques can be combined with robust security measures to enhance the reliability of GenAI systems while maintaining compliance with regulatory requirements.[5] As described, this approach integrates continuous validation processes throughout the AI lifecycle, from model development to deployment and monitoring, creating multiple checkpoints that verify the integrity and effectiveness of security responses. By emphasizing explainability alongside accuracy, the framework addresses one of the primary concerns associated with GenAI applications in security contexts: the "black box" nature of complex models.[5]

While not specifically focused on cloud security, research on GenAI applications in the energy sector offers transferable insights into implementation approaches for complex operating environments. This comprehensive literature review identifies how GenAI enhances productivity through data creation, forecasting, optimization, and natural language understanding, while also addressing challenges such as hallucinations, data biases, privacy concerns, and system errors [15]. The proposed solutions including improving training data quality, implementing system fine-tuning processes, establishing human oversight mechanisms, and deploying robust security measures provide a valuable framework for GenAI implementations in cloud security contexts. These approaches are particularly relevant for hyperscale environments where scale and complexity amplify both the benefits and risks of GenAI adoption [15].

2.4.5 Agent-Based Approaches

A recent paper from 2024 introduces and validates the concept of employing GenAI-driven agentic workflows to achieve comprehensive security automation, particularly in complex modern environments. A notable example is the Development, Security, and Operations (DevSecOps) Sentinel system[16], specifically designed to address the mounting security challenges inherent in modern software supply chains. Challenges coming from microservices, containerization, and cloud-native architectures that often outpace traditional DevSecOps practices[16].

The DevSecOps Sentinel system exemplifies this approach by utilizing intelligent agents integrated into automated workflows. These agents are powered by advanced GenAI models, such as LLMs enhanced with RAG, enabling sophisticated analysis capabilities[16]. Key characteristics of these agents include:

- **Autonomy:** Operating independently based on predefined goals and policies.
- **Reactivity:** Responding in real-time to environmental changes like new vulnerability disclosures.

- **Proactivity:** Taking initiative, such as preemptively scanning for risks or suggesting improvements[16].

These agents execute critical security tasks throughout the software development lifecycle, including:

- **Automated Vulnerability and Impact Analysis:** Leveraging GenAI to analyze code, dependencies and infrastructure configurations for potential threats, assessing their potential impact in context[16].
- **Adaptive Compliance and Release Gating:** Enforcing security policies and compliance requirements dynamically, acting as automated checks before deployment[16].
- **Predictive Security:** Utilizing AI to identify potential future risks based on historical data and emerging threat patterns[16].

The implementation and testing of DevSecOps Sentinel demonstrate several key points relevant to broader security automation:

1. **Viability for Complexity:** Agentic workflows powered by GenAI are shown to be a viable and effective method for tackling the intricate and rapidly evolving security issues found in modern, distributed systems[16].
2. **Synergy of AI and Agents:** The integration of GenAI's deep analysis capabilities with the autonomous, proactive nature of agentic systems offers a powerful paradigm for strengthening organizational security posture[16]. While Sentinel focuses on the supply chain, the principle applies broadly to automating security operations in complex cloud environments.
3. **Measurable Improvements:** Such systems can contribute to building and deploying software that is simultaneously faster, safer, and more reliable. The DevSecOps Sentinel study reported significant quantitative improvements in key security and operational metrics, including reduced Mean Time to Detect (MTTD) and Mean Time to Resolve (MTTR) for vulnerabilities, lower false positive rates, increased compliance pass rates, higher deployment frequency, and reduced change failure rates[16].

This approach, exemplified by DevSecOps Sentinel, highlights a promising direction for leveraging GenAI to automate and enhance security functions, moving beyond traditional limitations to offer more adaptive, context-aware, and efficient security management in demanding environments like hyperscale clouds.

2.4.6 Security Risks

The increasing integration of GenAI into various domains, including cybersecurity, presents significant opportunities but also introduces complex and multifaceted risks. Insights from recent literature reviews highlight these emerging challenges. A systematic literature review by Nyoto et al. [17], analyzing 17 relevant studies according to Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) 2020 guidelines [18], identifies several significant cybersecurity threats stemming primarily from the irresponsible application of GenAI technology. Complementing this, Surathunmanun et al. [15], while reviewing GenAI in the energy sector, outline key challenges that possess direct and critical relevance to security implementations, particularly within cloud environments reliant on third-party models and data. Synthesizing these findings provides a comprehensive overview of the risks:

- **Enhanced Malicious Content Generation and Misuse:** GenAI significantly lowers the barrier for creating sophisticated malicious content and tools. It can be abused to generate highly personalized and convincing phishing messages and social engineering tactics, increasing their effectiveness even with minimal target information [17]. Furthermore, GenAI facilitates the creation of effective ransomware and diverse forms of malware, potentially empowering individuals with limited coding expertise to launch attacks [17]. Beyond typical malware, it can also generate other executable attack code, such as Structured Query Language (SQL) injection scripts [17]. This potential for misuse is a major concern, where uncontrolled access or improper application can lead to significant harm [15]. This includes leveraging GenAI to bypass security controls through techniques like prompt injection or jailbreaking, as highlighted in literature concerning LLMs. [15].

- **Information Integrity:** GenAI poses substantial risks to information integrity. It enables the creation of highly realistic deepfake audio and video content, often without clear legal frameworks or consent, leading to potential fraud, manipulation, reputational damage, and the spread of disinformation [17]. Concurrently, GenAI models are prone to generating plausible but factually incorrect or nonsensical information, known as hallucinations [15, 17]. This issue often arises from poor quality training data or suboptimal parameter settings [15] and can be exacerbated by data poisoning during the model training phase [17]. In a security context, hallucinations can manifest as faulty threat analyses, incorrect vulnerability assessments, or misleading security recommendations [15]. Compounding this is the issue of data bias, where biases inherent in training data or introduced during feature selection lead to skewed or unfair outputs [15]. For security applications, this could result in certain threat types being consistently overlooked or specific user groups being unfairly flagged, thereby undermining the reliability of automated systems [15]. These challenges are often exacerbated by the inherent 'black box' nature of many LLMs, characterized by their complexity, lack of transparency in internal decision-making, and limited explainability, making it difficult to fully diagnose or prevent issues like hallucinations or bias [19].

- **Data Privacy, Security Vulnerabilities, and Intellectual Property:** The foundation of GenAI models, vast datasets, introduces significant privacy and security risks. Models are often trained on data scraped without explicit consent, potentially including sensitive personal information or copyrighted material [17]. User interactions and prompts can also be incorporated into training data, leading to potential data leakage and privacy violations [17]. This raises substantial intellectual property concerns and challenges compliance with regulations like General Data Protection Regulation (GDPR) [17]. The lack of transparency and control over how data is utilized presents considerable privacy risks [17]. Furthermore, insecure data handling practices can create security vulnerabilities [15]. Specific risks associated with LLMs, often used in cloud-hosted GenAI services, include inference attacks, data extraction attacks, data poisoning, supply chain vulnerabilities [15] and vulnerabilities to adversarial attacks stemming from the models complex and often opaque nature [19].

- **Systemic and Operational Risks:** Beyond content generation and data issues, GenAI systems can introduce operational risks. Logical inconsistencies within the model or unforeseen external events can cause GenAI systems to

produce errors or fail entirely [15]. In automated security workflows operating in cloud environments, such errors could propagate rapidly, leading to service disruptions, critical misconfigurations, or a failure to respond effectively to genuine threats [15].

These diverse risks, spanning malicious misuse, information integrity compromises, privacy violations, intellectual property infringements, and operational failures, underscore the critical need for robust countermeasures and responsible governance. Addressing these challenges necessitates comprehensive approaches, including rigorous data governance frameworks, cross-verification of GenAI outputs, continuous model monitoring and updating, incorporating human-in-the-loop validation processes, implementing strong security measures [15] and architectures like Zero Trust [15, 19], establishing clear ethical guidelines, and potentially developing new regulations specific to GenAI development and deployment [17]. Ensuring the responsible use of GenAI is paramount to harnessing its benefits while mitigating the significant emerging cybersecurity challenges, particularly in sensitive contexts like cloud security where the consequences of unreliable or misused AI can severely impact organizational risk posture and operational integrity [15].

Another significant challenge in implementing GenAI for security automation is the comprehensive identification and management of the unique risks these systems introduce, which differ significantly from traditional software risks. The NIST Artificial Intelligence Risk Management Framework (AI RMF) 1.0 [4] provides a structured, voluntary approach to address these challenges.

The AI RMF defines an AI system as an "engineered or machine-based system that can, for a given set of objectives, generate outputs such as predictions, recommendations, or decisions influencing real or virtual environments" [4, p.1]. It acknowledges that while AI offers transformative potential, it also poses distinct risks due to factors like data dependency, complexity, opacity, and the socio-technical context of deployment [4].

In the paper, the NIST describes some key points relevant to GenAI Security Risks in Cloud Computing.

1. **Unique AI Risk Landscape:** The framework highlights that AI risks differ from traditional software risks. Appendix B specifically notes challenges pertinent to GenAI and cloud environments, including:

    - Dependency on vast datasets which may harbor biases or quality issues, and are susceptible to poisoning attacks [4].
    - Risks associated with using pre-trained models, which can "increase levels of statistical uncertainty and cause issues with bias management, scientific validity, and reproducibility" [4, p.38]. This is crucial in cloud settings where models might be sourced from third parties.
    - Increased opacity and difficulty in predicting failure modes or emergent behaviors, complicating security validation [4]. This aligns with the widely recognized 'black box' problems of LLMs, encompassing their complexity, lack of transparency, and limited explainability [19]. Specific security concerns not fully addressed by traditional frameworks, such as "evasion, model extraction, membership inference, availability, or other machine learning attacks" [4, p.39], including adversarial vulnerabilities common in LLMs [19].
    - Risks associated with "third-party AI technologies, transfer learning, and off-label use," which are highly relevant when using GenAI models hosted or integrated via cloud services [4, p.39].

2. **Trustworthiness Characteristics:** The Risk Management Framework (RMF) emphasizes achieving trustworthy AI by balancing several characteristics [4]. For security, the most critical are:

    - **Secure and Resilient:** AI systems should maintain "confidentiality, integrity, and availability" and be able to "withstand unexpected adverse events or unexpected changes" [4, p.15]. This includes protecting against data poisoning, adversarial examples, and model exfiltration key threats for GenAI. The RMF notes applicability of existing standards like the NIST Cybersecurity Framework here [4, p.15].
    - **Accountable and Transparent:** While distinct from security, transparency and accountability are vital for security incident analysis, understanding vulnerabilities, and assigning responsibility, especially in complex cloud supply chains [4].

- **Privacy-Enhanced:** GenAI often processes vast amounts of data, potentially including sensitive information. Privacy risks are intertwined with security, as data breaches impact both. The RMF advocates for privacy considerations throughout the lifecycle and mentions Privacy-Enhancing Technologies[4].
- **Valid and Reliable:** Systems must perform accurately and consistently. Unreliable GenAI could produce insecure code, faulty security recommendations, or fail in ways that create security openings [4].

3. **Risk Management Core Functions:** The RMF outlines four functions to operationalize risk management:

   - **Govern:** Establishing a risk management culture, policies, accountability structures, and processes. Crucially, this includes policies addressing risks from "third-party software and data and other supply chain issues", vital for cloud-based GenAI [4, pp.21-24].
   - **Map:** Establishing context, categorizing the AI system, understanding capabilities and limitations, and mapping risks/benefits, explicitly including those from third-party components[4].
   - **Measure:** Applying methods and metrics to assess risks and evaluate trustworthy characteristics, including specific evaluations for security and resilience and privacy[4].
   - **Manage:** Prioritizing and responding to risks, including managing risks from third-party entities and implementing incident response and recovery plans[4].

In essence, the NIST AI RMF 1.0 provides a comprehensive framework that, while voluntary and high-level, guides organizations in systematically considering the multifaceted risks, including significant security and privacy challenges, inherent in developing, deploying, and using complex AI systems like GenAI, particularly within the context of third-party dependencies common in cloud computing environments. It stresses the importance of integrating risk management throughout the AI lifecycle and addressing the unique characteristics and vulnerabilities of AI technologies. Adding to frameworks like the NIST AI RMF, specific architectural approaches are emerging to address the unique security challenges of GenAI in cloud environments. One prominent example is Zero Trust Architecture (ZTA) [19]. ZTA moves away from traditional perimeter-based security towards a model where trust is never assumed, and verification is continuously required[19]. This aligns well with the NIST RMF's emphasis on secure and resilient systems and proactive risk management, particularly given the "black box" nature and dynamic deployment of many GenAI models [4, 19]. Key tenets include strict identity verification, micro-segmentation to limit lateral movement, least privilege access control, and continuous monitoring [19]. Implementing ZTA for LLMs involves specific considerations such as unified identity management across cloud platforms, AI-driven dynamic access policies, automated network segmentation, robust data encryption and classification, continuous threat monitoring tailored to LLM vulnerabilities, and ensuring compliance [19]. Interestingly, AI itself can enhance ZTA through behavioral analytics for continuous authentication or threat intelligence processing[19]. However, implementing ZTA effectively presents its own challenges, including complexity, integration with legacy systems, resource requirements, and potential performance impacts [19].

2.4.7 Balance of Automation and Human Oversight

The integration of AI, particularly GenAI, into cybersecurity presents a significant paradigm shift, offering powerful automation capabilities to counter increasingly sophisticated cyber threats. A recurring theme in the literature, however, is the inherent tension between the compelling benefits derived from this automation and the indispensable necessity of human oversight [2]. While AI-powered security automation provides crucial safeguards against evolving cyber dangers, the unique characteristics and potential risks associated with AI systems, especially GenAI, underscore the continued importance of human expertise and intervention [2, 3].

A fundamental principle, strongly articulated within risk management frameworks, is that no "high-risk" AI system should be operated without substantial human oversight [4, p.7]. This necessitates careful deliberation regarding whether the potential benefits of deploying such systems truly outweigh the potential negative impacts and risks [4]. In cybersecurity contexts, high-risk applications might include automated incident response systems with the potential for disruptive countermeasures, security policy generation influencing critical infrastructure, or threat analysis tools whose outputs directly inform high-stakes decisions. The NIST AI RMF emphasizes that in situations where AI systems present unacceptable negative risk levels, such as imminent significant negative impacts or the occurrence of severe harms, their development and deployment should cease until these risks can be sufficiently managed [4].

Despite the promising applications of GenAI for security automation such as generating security reports, suggesting code fixes, or creating configuration scripts significant challenges remain in striking the right balance between automation and appropriate human oversight. Research highlights several critical issues stemming from the use of GenAI in automated security operations [3]. One major concern is the potential for over-dependence on AI tools, which could lead to complacency or a degradation of human skills [3]. Furthermore, GenAI models themselves are susceptible to adversarial risks, including data poisoning or prompt injection attacks designed to manipulate their outputs, presenting unique security challenges [3]. The inherent complexity and often opaque nature of decision-making processes within sophisticated AI systems, including GenAI, can also hinder effective oversight and accountability [4] [3].

Effectively managing GenAI in cybersecurity demands a recognition that complete automation without human intervention introduces unacceptable risks [3]. Human oversight is crucial not merely as a final checkpoint but throughout the AI lifecycle. This includes defining system goals and constraints, interpreting ambiguous or novel situations that fall outside the AI's training data, providing contextual understanding that the AI may lack, and making ethical judgments, particularly when potential actions have significant consequences [4]. The NIST AI RMF emphasizes the importance of clearly defined human roles and responsibilities within human-AI configurations, acknowledging the influence of human cognitive biases and the need for systems that are explainable and interpretable to those operating or overseeing them [4].

Frameworks like the NIST AI RMF provide structured approaches to managing these challenges. The GOVERN function stresses establishing a risk management culture, defining roles, and ensuring accountability [4, p. 21-24]. The MAP function requires establishing context, understanding system limitations, and defining processes for human oversight [4, p. 24-28]. MEASURE involves ongoing monitoring of

performance, safety, and fairness, incorporating feedback mechanisms [4, p. 28-31]. Crucially, the MANAGE function includes planning risk responses and implementing mechanisms to supersede, disengage, or deactivate AI systems demonstrating performance inconsistent with intended use, alongside robust post-deployment monitoring and incident response plans [4, p. 31-33].

Ultimately, the effective use of GenAI in cybersecurity hinges on achieving a balanced, symbiotic relationship between automated capabilities and human expertise. This balanced approach acknowledges the complementary strengths of humans and AI. GenAI can process vast amounts of data and automate repetitive tasks at scale and speed, while humans provide critical thinking, contextual awareness, ethical guidance, and ultimate accountability [3]. Preventive efforts and well-planned action plans, incorporating robust human oversight mechanisms, are essential to harness the benefits of GenAI for cybersecurity while mitigating its inherent risks [3].

2.4.8 Summary Literature State of the Art

This literature review demonstrates that GenAI represents a transformative technology for security automation within hyperscale cloud environments. The analysis reveals significant potential for GenAI to enhance security operations through automated threat detection, policy generation, and incident response, particularly across complex multi-cloud settings. Research highlights notable advancements in conceptual frameworks for multi-cloud policy orchestration, validation mechanisms to ensure trust and accuracy, and technical approaches for implementing GenAI at scale. The most promising strategies often leverage multi-cloud architectures, zero-trust principles, and comprehensive security frameworks, while necessarily acknowledging the unique infrastructure requirements of GenAI itself. However, despite this progress, persistent challenges related to trust, validation, data privacy and quality, and the crucial balance between automation and human oversight remain significant considerations. As this field continues its rapid evolution, interdisciplinary collaboration will be essential to develop robust ethical norms and innovative defense mechanisms, addressing current issues while guiding the responsible application of GenAI in cybersecurity.

2.5 Research Gaps

While the literature review demonstrates significant progress in applying Generative AI to cloud security, a deeper analysis reveals several critical research gaps. The convergence of GenAI and hyperscale security automation is still in its nascent stages, and many existing studies focus on high-level frameworks or narrow applications. This thesis addresses some of the underexamined areas by exploring the following gaps.

A primary gap is the practical scalability of these solutions to a true hyperscale level. Much of the current academic literature fails to fully address the operational challenges of implementing GenAI security in enterprise environments characterized by thousands of developers and high-velocity CI/CD pipelines [20]. The significant computational resources required for large-scale models and the potential for latency to create bottlenecks in rapid deployment cycles present substantial operational overhead. These questions of performance and resource management at scale remain largely unexamined in academic research. This work begins to address this gap by measuring the performance of the developed prototype (as detailed in Section 4.4.2)

and discussing the architectural trade-offs essential for a scalable implementation (Section 7.3), providing a basis for future research into optimizing these systems for performance in hyperscale environments. Furthermore, while the security risks of using GenAI are well-documented (Section 2.4.6), a more advanced and less-explored research area is the adversarial resilience of the automation pipeline itself [17, 19]. A critical gap exists in understanding and defending against attacks that manipulate a defensive GenAI into generating insecure policies. For instance, a malicious actor could craft Infrastructure-as-Code (IaC) designed to trigger a prompt injection, causing the LLM to produce a security policy with a hidden backdoor. There is a pressing need for research into building resilient GenAI security systems capable of withstanding such targeted adversarial inputs. This thesis contributes by establishing a multi-stage Validation Layer (Section 4.1.3), which includes a "self-scan" of generated policies, as a necessary architectural component. This opens the door for future work on more advanced defenses, such as using multiple AI models as cross-checks or developing AI-specific threat modeling for the automation pipeline.

Another significant challenge lies in the heterogeneity and complexity of multi-cloud orchestration [21, 22]. Despite strong industry demand, there is limited research on practical, fully-realized GenAI solutions that can effectively orchestrate security policy across the distinct technical ecosystems of the major cloud providers. The primary difficulty is managing the heterogeneity of services, APIs, and security best practices. A truly effective GenAI system must be able to translate a single high-level security requirement into multiple, provider-specific, and contextually-aware policy implementations. While the prototype developed in this thesis focuses on a single cloud provider for the implementation (Section 5.2), its conceptual framework is designed to be cloud-agnostic by leveraging tools like Terraform and Open Policy Agent (OPA). In doing so, this work lays the theoretical groundwork for a multi-cloud solution and highlights the specific challenges that must be solved such as building a multi-cloud RAG knowledge base thereby defining a clear roadmap for future research.

Finally, the concept of a feedback loop is central to improving AI systems, yet its application in security automation is critically underdeveloped, specifically concerning automated learning and adaptation from human feedback. While many frameworks, including the one proposed here, incorporate a Human-in-the-Loop (HITL) process, a gap exists in defining and evaluating robust mechanisms for the system to automatically learn from the human expert's decisions [23, 24]. Simply having a human approve or reject a policy is a control, not a learning process. Research is needed on how to safely use this feedback to refine system prompts, update the RAG knowledge base, or even fine-tune the model to improve its accuracy and contextual understanding over time without introducing new biases or vulnerabilities [25, 26]. The conceptualization of the HITL workflow (Section 4.5) and its integration into CI/CD pipelines (Section 4.6) within this thesis creates the necessary foundation for such a feedback loop. This work identifies the critical need for this capability and proposes an architecture for how such a learning system could be implemented, setting a clear direction for subsequent research and development.

Chapter 3

Methodology

This chapter outlines the research methodology employed in this thesis. It details the research design, the approach for developing the conceptual framework, the implementation of the prototype, and the methods used for evaluation. The rationale for each methodological choice is explained to ensure transparency and replicability.

3.1 Main Section 1

This section introduces the first main topic of the chapter. It provides a high-level overview of the concepts that will be discussed in the following subsections.

3.1.1 Subsection 1

This subsection delves into the first sub-topic, providing detailed explanations and examples.

3.1.2 Subsection 2

This subsection covers the second sub-topic, building upon the concepts introduced in the previous subsection.

3.2 Main Section 2

This section transitions to the second main topic of the chapter, presenting a new set of ideas and analyses.

Chapter 4

Conceptual Framework for GenAI-Driven Security Automation

This chapter introduces the conceptual framework designed to address the critical challenges of automated security analysis and policy generation for cloud infrastructure. The proposed architecture presents a comprehensive, multi-layered approach that systematically processes IaC artifacts and automatically generates corresponding security policies. At its core, the framework leverages the power of traditional static analysis tools and advanced LLMs to create a robust security automation pipeline.

4.1 Architectural Overview of the Proposed Framework

The proposed architecture is founded on a hybrid model intentionally designed for efficacy, combining the reliability of established security scanners for identifying known vulnerability patterns with the contextual intelligence of generative AI [1]. This approach allows for deeper analytical capabilities, necessary for uncovering complex, context-dependent security issues that traditional tools often miss [27]. Furthermore, the architecture is conceived for seamless integration into modern Development and Operations (DevOps) workflows, particularly Continuous Integration/Continuous Deployment (CI/CD) pipelines, to operationalize a Policy-as-Code (PaC) model and enforce security throughout the development lifecycle [1].

As illustrated in Figure 4.1, the framework operates as a cohesive GenAI Security System triggered by external development workflows. The process begins when a CI/CD pipeline initiates the workflow, providing IaC artifacts to the Ingestion Layer. This layer consumes the raw configurations and prepares them for analysis. The data is then passed to the Analysis Layer, which performs a two-stage evaluation. First, a Static Analysis component generates a baseline vulnerability report. This report is then fed into the GenAI Analysis component, which queries a curated Knowledge Base to produce an enriched, context-aware report, reducing false positives and identifying nuanced flaws.

The enriched findings proceed to the Policy Layer, where the Policy Generator component leverages both the enriched report and the Knowledge Base to manifest a preventative Generated Policy. This artifact is immediately forwarded to a Policy Validator for automated checks. Following validation, the policy is presented to a Human Reviewer, a critical step that ensures oversight. Once approved, the now Validated Policy is committed back to the repository by the CI/CD pipeline, closing the loop. This layered design provides a comprehensive and efficient system for

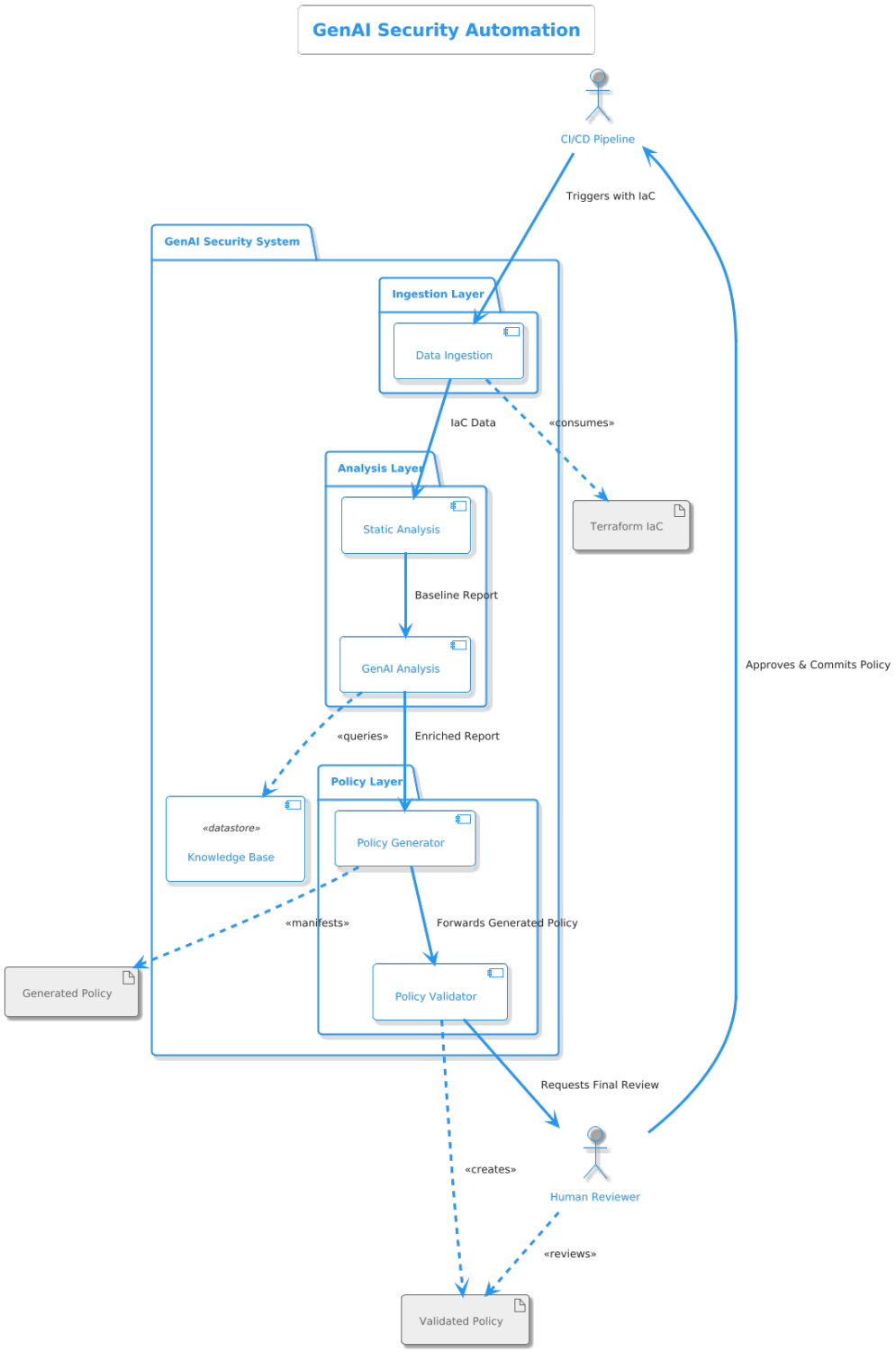

FIGURE 4.1: Component Diagram of the Proposed GenAI-Driven Security Automation Framework

enhancing cloud security posture by translating identified vulnerabilities directly into enforceable controls [28]. The following subsections will detail the specific roles and functions of each of these core layers.

4.1.1 Ingestion Layer

The Data Ingestion Layer serves as the foundational entry point for security artifacts into the automation framework. Its primary function is to ingest IaC configurations, a prevalent standard for provisioning and managing cloud infrastructure. The reliance on IaC, while enhancing automation and consistency, introduces significant risks such as misconfigurations, coding errors, and embedded secrets, making automated analysis a critical requirement for secure cloud operations.

This layer is designed to support both batch and real-time ingestion modes, a flexible approach that aligns with modern data pipeline architectures emphasizing scalability and performance[29]. Batch ingestion allows for comprehensive, scheduled scans of entire code repositories, while real-time ingestion facilitates immediate analysis within CI/CD pipelines [30]. The framework is designed to receive these IaC configurations via programmatic interfaces, ensuring seamless integration into existing developer workflows and automated systems.

Upon ingestion, the Data Ingestion Layer reads and parses the raw IaC configuration files, transforming them into a structured format suitable for further analysis. This foundational step ensures that the input artifacts are normalized and ready for systematic evaluation. Once parsing is complete, the processed configurations are handed off to the subsequent Analysis Layer, where both static and GenAI-driven analyses are performed to identify vulnerabilities and misconfigurations. The following subsection details the design and operation of this core analytical component.

4.1.2 Analysis Layer

Following the Data Ingestion Layer, the Analysis Layer is responsible for the core analysis of the ingested IaC artifacts. A central design principle of this framework is the segregation of processing activities into two distinct but complementary sublayers: a traditional Static Analysis Engine and an advanced GenAI Analysis Engine.

The rationale for this dual-layer architecture is to create a highly efficient and comprehensive security analysis pipeline. This approach leverages the respective strengths of each technology. Static analysis provides a rapid, reliable, and computationally inexpensive method for identifying a wide range of known, pattern-based vulnerabilities. By filtering out these common issues first, the framework can then employ the more resource-intensive GenAI engine to focus on complex, context-dependent security flaws that traditional tools are ill-equipped to detect [31]. This layered methodology optimizes analytical depth while maintaining operational efficiency, ensuring that both well-defined and nuanced vulnerabilities are addressed [1].

The first stage of this layer employs a suite of established static analysis security testing (SAST) tools to conduct an initial scan of the IaC configuration. This engine examines the configuration for syntactic and structural flaws by referencing curated databases of known vulnerabilities, common misconfigurations, and code smells. It validates the configuration against established security benchmarks and standards. The primary output of this stage is a baseline vulnerability report, which provides

a structured list of potential issues identified through deterministic, rule-based pattern matching. This report serves as a foundational input for the subsequent, more sophisticated analysis stage.

The second stage is the GenAI Analysis Engine, which represents the core innovation of this framework and directly addresses the research interest in applying generative AI to cloud security. This engine utilizes LLMs to perform a deeper, contextual analysis that transcends the limitations of traditional static scanners [32]. It takes as input both the original IaC configuration and the baseline vulnerability report from the previous stage, using the initial findings to enrich its analytical context [33].

This engine is designed to identify security weaknesses that require an understanding of developer intent, architectural relationships, and complex business logic [34]. Its capabilities include:

- **Identifying Context-Sensitive Flaws:** Detecting risks that emerge from the interaction of multiple configurations. For example, overly permissive network rules may appear acceptable in isolation but create a vulnerability when combined with a specific resource's placement within the network architecture[31].
- **Uncovering Logical and Policy Violations:** Identifying logical flaws in resource deployments, such as potential circular dependencies. The engine can also detect violations of complex, unwritten organizational policies like nuanced tagging and naming conventions [1].
- **Reducing False Positives:** Differentiating between genuine security risks and findings from the static analysis that are benign within a specific operational context. For example, a "hardcoded secret" may simply be a placeholder for a non-production environment.

By synthesizing information from the configuration and the initial scan, the GenAI Analysis Engine bridges the gap between traditional, rule-based detection and adaptive, context-aware threat identification, producing a consolidated and enriched vulnerability report.

4.1.3 Policy Layer

The Policy and Validation Layer operationalizes the insights derived from the Analysis Layer, acting as the primary action-oriented component of the framework. Its purpose is to automate the creation of security artifacts in the context of this prototype, preventative policies using GenAI. This layer directly addresses a core aspect of this research: leveraging LLMs to not only analyze but also actively generate security policies. The integration of GenAI into the security architecture in this manner marks a significant shift, promising to streamline development workflows and accelerate remediation cycles.

This layer leverages LLMs to generate security policies tailored to the vulnerabilities identified in the preceding analysis stages. The generated artifacts are formal policies written in a declarative, machine-readable language, designed for automated enforcement. The LLM is guided by system prompts and a curated knowledge base of security standards to produce precise, context-aware rules. This process of generating platform-specific policies from a higher-level analysis aligns with established methods in automated systems engineering, where abstract requirements are translated into concrete, executable artifacts for a target platform[28].

A critical aspect of this layer is its multi-stage validation process, which functions as an essential trust and safety mechanism. It is designed to rigorously verify the integrity, correctness, and security of the AI-generated security policies before they are committed to a repository or presented for human review. This mitigates risks associated with AI-generated artifacts, such as factual inaccuracies (hallucinations) or the introduction of new security flaws. Raw, unvalidated output is never trusted for deployment. The workflow includes several checkpoints:

- **Automated Validation:** Generated policies undergo initial automated checks. This is a critical quality assurance step to ensure that only high-quality, effective, and secure policies proceed. The process includes:
  - **Syntactic Validation:** The layer uses standard parsers and validators to confirm that the generated policy is syntactically correct and adheres to the relevant language specifications. Any policy that fails this check is immediately rejected and logged.
  - **Security Self-Scan:** To prevent the AI from inadvertently introducing new vulnerabilities, the generated security policy itself is subjected to a security scan using static analysis tools. This "self-scan" ensures the remediation policy does not create new security problems while attempting to solve another.
- **Human-in-the-Loop Review:** The framework mandates a human-in-the-loop review process, which is indispensable for high-impact changes or when the AI model's confidence in its output is low. This approach maintains a crucial balance between automation and human oversight, a central theme identified in the literature [35].
- **Advanced Testing:** For more accuracy, the architecture can incorporate further testing to detect subtle inconsistencies or unintended behaviors in the generated policies.

From a governance standpoint, the layer integrates robust security controls. Access controls and authentication mechanisms restrict the policy generation function to authorized entities and automated processes. Comprehensive audit logs are maintained for all generated and validated artifacts, ensuring traceability for compliance and forensic analysis, a key element in modern data architectures[36]. Only after a generated policy successfully passes all stages of this validation gauntlet is it considered "validated". The validated policy, along with its comprehensive audit report, is then passed to the CI/CD pipeline, where it can be reviewed and approved by a human expert before being enforced as part of the organization's PaC repository.

4.2 Integration of GenAI-Driven Security Automation

The core of the proposed security automation framework is centered around the integration of GenAI, specifically through the use of LLMs accessed as a managed cloud service. This approach was deliberately chosen over deploying and managing local, open-source models for several strategic reasons. Utilizing a hyperscale cloud provider's managed AI service offers access to powerful, state-of-the-art models without the substantial computational and financial overhead associated with self-hosting. It abstracts away the complexities of MLOps, such as infrastructure provisioning, scaling, and maintenance, allowing the focus to remain on the application

logic. Furthermore, this model aligns with the Shared Responsibility Model discussed in the literature review, where the cloud provider manages the security and availability of the underlying AI service.

To ensure the generation of accurate, contextually relevant, and reliable security policies, the framework employs a RAG architecture. This pattern is crucial for grounding the LLM's output in factual data, thereby mitigating the risk of model "hallucinations" a significant concern in GenAI systems where plausible but incorrect information may be generated [37]. The RAG process within this framework functions as follows:

1. Upon receiving a vulnerability finding from the Data Processing Layer, the system queries a dedicated Knowledge Base. This knowledge base is a curated repository containing relevant security standards, vulnerability information, best practices for the given IaC technology, and documentation for the target policy language [38].
2. The retrieved documents, which provide specific context for the detected vulnerability, are then combined with a custom System Prompt. This prompt instructs the LLM on its role, the task to be performed (e.g., "You are a security expert. Generate a precise security policy to prevent the following vulnerability"), and the required output format.
3. This enriched context, consisting of the vulnerability data, retrieved knowledge, and the system prompt, is then sent to the selected LLM via the managed service's API to generate the security policy.

This RAG-based approach ensures that the generated policies are not only syntactically correct but are also directly informed by authoritative and up-to-date security guidance, making the system more robust and trustworthy [37]. By externalizing the knowledge base, the framework can be easily updated to reflect new standards or threat intelligence without needing to retrain or fine-tune the underlying LLM [38]. A high-performance foundation model is utilized for its advanced reasoning and code generation capabilities.

4.3 Leveraging LLMs for Deeper Contextual Analysis

The deployment of LLMs within the security automation framework fundamentally transforms the depth and quality of IaC analysis. These models introduce a level of contextual understanding previously unattainable with traditional static analysis tools. Unlike rule-based scanners, which are limited to identifying known vulnerability patterns and syntactic misconfigurations, LLMs can synthesize information across multiple resources, configuration layers, and organizational policies. This enables them to surface nuanced, context-sensitive security issues.

By integrating LLMs with outputs from static code analyzers and a curated knowledge base, the framework is capable of identifying misconfigurations and policy violations that arise from complex interactions within the cloud environment [39]. This deeper insight is made possible by the LLM's ability to reason about architectural relationships, resource dependencies, and the intent behind configurations, allowing it to detect security weaknesses that would otherwise remain hidden.

A key advantage of this approach is the identification of context-sensitive security weaknesses. The LLM is able to analyze configurations in light of their broader environment and operational context, flagging settings that may be technically valid in

isolation but become risky when considered alongside other resources or data sensitivity. For example, an overly permissive network rule might not trigger an alert in a development environment, but if linked to production data or exposed to the public internet, it becomes a significant risk a nuance the LLM can discern by analyzing tags, naming conventions, and architectural metadata.

Beyond identifying outright vulnerabilities, the LLM can uncover suboptimal or inefficient configurations that deviate from best practices for performance, cost-efficiency, or resilience, tailored to the specific needs of the application. It can also interpret and enforce complex internal policies that are difficult to codify with static rules, such as intricate naming conventions, tagging strategies for governance, or architectural patterns mandated by the organization [40]. This capability extends to spotting logical flaws in resource deployment and interconnections, such as circular dependencies, misconfigured network routing, or resource configurations that do not align with their intended purpose.

The LLM's contextual reasoning also enables it to detect deviations from evolving best practices and industry standards by leveraging its knowledge base of security benchmarks and official documentation. This ensures that the framework remains adaptive to new vulnerability patterns and compliance requirements as they emerge. Furthermore, the LLM can analyze how combinations of individually acceptable configurations or permissions might aggregate into an elevated risk profile, identifying attack paths that arise only when multiple minor issues are considered together.

A practical example illustrates this capability: consider a storage resource that appears secure in isolation, with a restrictive access policy allowing permissions only to a specific role. The associated role is properly scoped, and the storage policy adheres to the principle of least privilege. However, a compute instance in a development environment, which has this role attached, is exposed to the internet via an open management port and is running a vulnerable operating system. While static analyzers might flag the open port and operating system vulnerability separately, and pass the storage resource and role as secure, the LLM can connect these findings. It recognizes that the instance's exposure and vulnerability, combined with its privileged role, create a critical attack path to sensitive data in the storage resource. This context-sensitive weakness would likely be missed or deprioritized by traditional tools, but the LLM's holistic analysis surfaces it as a high-priority risk.

In summary, leveraging LLMs for deeper contextual analysis enables the framework to move beyond pattern-based detection, offering a comprehensive understanding of security posture that accounts for the dynamic and interconnected nature of modern cloud environments. This results in the proactive identification of genuine risks, reduction of false positives, and the continuous alignment of security controls with evolving organizational and industry standards [41].

4.4 Metrics for Security Posture Assessment

To provide a robust and scientifically grounded evaluation of the prototype's performance, this section defines a set of quantitative metrics. These metrics are designed to assess two critical dimensions of the framework: its effectiveness in generating accurate and preventative security controls, and its operational efficiency in doing so at a speed compatible with modern DevOps workflows. By focusing on these key

performance indicators, the evaluation will provide concrete evidence of the framework's ability to automate security policy generation reliably and at scale. The following subsections detail the specific metrics for Automated Policy Efficacy and Policy Generation Speed.

4.4.1 Automated Policy Efficacy

To provide a scientifically sound assessment of the prototype's capabilities, this section introduces the metric of Automated Policy Efficacy. Unlike a traditional vulnerability reduction metric, which would measure changes in a live environment, this metric quantifies how effectively the framework's generated policies, if applied, would prevent the vulnerabilities identified during the analysis phase. This approach provides a direct and accurate measure of the prototype's core function: the automated generation of high-quality, preventative security controls. It evaluates the tangible output of the system, offering a clear indicator of its potential to mitigate risk before deployment.

The assessment follows a "before and after" model, where the "before" state is the baseline of identified vulnerabilities and the "after" state is the measure of the generated policies effectiveness in addressing them.

Before: Baseline of Identified Vulnerabilities

The process begins by establishing a clear baseline of the security risks present in a given set of Infrastructure-as-Code (IaC) configurations.

- **Action:** An initial scan is conducted by feeding the target IaC configurations into the prototype's *Data Ingestion* and *Data Processing Layers*. This invokes both the Static Analysis Engine and the GenAI Analysis Engine to produce a comprehensive vulnerability report.
- **Record:** The total number of unique vulnerabilities detected ($V_{\text{total, initial}}$) is documented, along with their distribution across severity levels: critical (V_c), high (V_h), medium (V_m), and low (V_l). This collection represents the complete set of issues that the prototype aims to address with generated policies.

After: Effectiveness of Generated Policies

Following the baseline analysis, the focus shifts to evaluating the quality and correctness of the policies produced by the prototype's *Code Generation Layer* and verified by the *Validation Layer*.

- **Action:** For each vulnerability identified in the "before" step, the prototype generates a corresponding security policy in Rego. These policies are then passed through the automated *Validation Layer*.
- **Measurement:** The effectiveness of the generated policies is determined using a combination of syntactic accuracy and preventative capability:
  - **Policy Accuracy ($A_{\textbf{policy}}$):** This confirms that a generated policy is syntactically correct and well-formed. A policy must pass this check to be considered for effectiveness testing.

– **Policy Effectiveness (E_{policy}):** This measures the percentage of syntactically valid policies that, when tested in a controlled staging environment or through simulated deployment checks, successfully prevent the specific misconfiguration they were designed to address.

Quantifiable Impact

The ultimate measure of efficacy is the percentage of initially identified vulnerabilities for which the prototype successfully generated a syntactically correct *and* effective preventive policy. This can be calculated for the total set and broken down by severity to provide a granular view of the system's performance against the most critical risks.

4.4.2 Policy Generation Speed

This metric quantifies the intrinsic efficiency of the prototype's automated capabilities, focusing on the absolute performance of the GenAI-driven Code Generation Layer. Rather than comparing against a manual baseline, this metric demonstrates the high-speed performance that is fundamental to enabling modern, agile security workflows. In the context of hyperscale cloud platforms, where the "scale and sophistication of threats necessitate advanced automation capabilities" (as discussed in Chapter 2.4), the ability to generate security controls rapidly is a critical enabler for maintaining a robust security posture. The value is not in being merely faster than a human, but in operating at a speed that makes "shift left" security and seamless CI/CD integration practical at scale [20].

The argument for this automated approach is grounded in the inherent limitations of manual processes when faced with the complexity and dynamism of modern cloud environments. Manual policy creation is a resource-intensive task that introduces cognitive load on security experts and creates bottlenecks in development lifecycles, slowing deployment speed and reducing productivity [42, 43]. Research confirms that manually integrating security into DevOps workflows can impede delivery speed, whereas AI-driven approaches promise to automate these workflows and reduce manual efforts [20]. By automating this function, the prototype reduces friction in the CI/CD pipeline, supports scalability, and allows human experts to shift their focus from routine generation to high-value oversight and review, as described in the Human-in-the-Loop workflow (Section 4.5).

Controlled studies provide compelling quantitative evidence for this acceleration. For instance, a notable experiment with GitHub Copilot, an AI pair programmer, found that professional developers with access to the AI assistant completed a complex task 55.8% faster than the control group [44]. This dramatic increase in efficiency is not isolated to general coding; similar improvements in developer velocity are measured in specialized cloud engineering tasks, including the generation of IAM policies, which is directly analogous to the prototype's function [45]. Furthermore, research frameworks are now being developed to specifically calibrate AI performance against human baselines for software tasks, measuring the time saved on work that would take a human anywhere from one minute to over eight hours to complete [46]. This ability to automate complex, time-consuming software engineering tasks with high rates of success is a key driver for adopting AI in DevOps [47].

To measure this, the framework focuses on the time elapsed from the moment a confirmed vulnerability is passed to the Code Generation Layer until a syntactically valid and effective policy is produced and passes its initial automated validation. This is captured by two key indicators:

- **Average Time per Policy (T_{gen}):** The core measure of speed for a single generation task.
- **Policy Throughput:** The number of valid policies the system can produce per minute, illustrating its capability to handle vulnerabilities at scale.

The measurement is conducted via controlled experimentation. A diverse, representative set of confirmed vulnerability inputs is run through the prototype multiple times to gather robust data. The start time (t_{start}) is recorded when the input reaches the Code Generation Layer, and the end time (t_{end}) is recorded when the validated policy is successfully output. The average generation time is then calculated using statistical analysis.
Calculation: The average time, T_{gen}, is calculated as:

$$T_{\text{gen}} = \frac{1}{N} \sum_{i=1}^{N} (t_{\text{end},i} - t_{\text{start},i})$$

where N is the total number of policies generated in the test set.

4.5 Human-in-the-Loop for Review and Approval

While the framework is designed to maximize automation, the integration of a Human-in-the-Loop (HITL) process for review and approval is a foundational principle, reflecting a core theme identified in the literature review regarding the balance between automation and human oversight [48]. The complete automation of security policy generation and enforcement without human intervention introduces unacceptable risks, particularly in complex cloud environments. This subsection outlines the conceptual design of the HITL workflow, which serves as a critical control point to ensure the safety, accuracy, and contextual appropriateness of the AI-generated security artifacts.

The necessity for human oversight is a principle strongly articulated within established risk management frameworks, which states that high-risk AI systems should be operated with a meaningful human role [48]. In this framework, the HITL process is not merely a final checkpoint but an integrated function designed to mitigate the inherent risks of GenAI, such as the generation of incorrect policies (hallucinations), the introduction of new security flaws, or the creation of overly restrictive rules that could impede business operations [48]. It operationalizes established governance principles by providing a mechanism to validate, override, or reject the AI's output before it can impact the production environment [24].

The HITL review and approval workflow is triggered under specific, risk-informed conditions. A manual review by a qualified security engineer is mandatory for any AI-generated policies that address high-severity or critical vulnerabilities. A review can also be triggered when the AI model indicates a low confidence score for its generated output or when the proposed change targets a particularly sensitive component of the cloud infrastructure [48]. This risk-based approach ensures that human expertise is focused where it is most needed, optimizing for both security and operational efficiency [24].

During the review process, the human expert is presented with a comprehensive set of information to facilitate an informed decision. This includes:

1. the original vulnerability report

2. the raw IaC snippet containing the vulnerability
3. the AI-generated security policy for remediation
4. the results of automated validation checks
5. an AI-generated explanation of the policy's logic and how it addresses the issue

This curated context allows the reviewer to assess the generated artifact's accuracy, effectiveness, and potential side effects. The reviewer can then approve the policy, allowing it to proceed to the CI/CD pipeline for enforcement, or reject it [24]. Rejected policies are flagged and can be used as part of a feedback loop to refine the system prompts and knowledge base used by the Code Generation Layer, contributing to the system's continuous improvement. Ultimately, this symbiotic relationship between the automated capabilities of GenAI and the contextual wisdom of human experts ensures that the framework operates not only with speed and scale but also with the necessary accountability and safety [24].

4.6 Integration with CI/CD Pipelines for Policy-as-Code

The ultimate objective of the conceptual framework is to translate its analytical outputs and AI-generated artifacts into tangible, preventative controls that are seamlessly embedded within an organization's development lifecycle. This is achieved by integrating the framework into a CI/CD pipeline, operationalizing a PaC workflow [49]. This approach embodies the "shift left" security principle, where security checks and policy enforcement are automated and moved to the earliest stages of the development process, rather than being an afterthought.

The integration follows a defined workflow, typically initiated within a version control system through a pull request. When a developer proposes changes to the cloud infrastructure by modifying IaC configurations, a CI/CD pipeline is automatically triggered. This pipeline orchestrates the core functions of the framework in a sequence designed to enforce security before insecure configurations are merged:

1. **Automated Scanning and Analysis:** The pipeline first invokes the Data Ingestion and Data Processing layers to scan the proposed infrastructure changes. It generates a comprehensive vulnerability report, leveraging both static analysis and the deeper contextual analysis from the GenAI engine.
2. **Policy Generation and Committing:** If new, unaddressed vulnerabilities are detected, the Code Generation Layer is triggered to produce the corresponding security policies. Following the HITL review and approval process for these policies, the validated policy files are treated as code artifacts themselves. They are committed to a dedicated policy repository, ensuring they are version-controlled, auditable, and consistently applied [49].
3. **Policy Enforcement as a Quality Gate:** The critical enforcement step is implemented using an automated policy engine as a quality gate within the CI/CD pipeline [50]. The pipeline uses this engine to evaluate the proposed infrastructure plan against the entire set of approved security policies. If the proposed changes violate any policies, particularly those addressing high-severity vulnerabilities, the policy evaluation fails. This failure causes the CI/CD pipeline to halt and blocks the pull request from being merged. This mechanism acts

as a powerful preventative control, ensuring that configurations failing to meet security standards cannot be deployed [49] [50].

4. **Metrics and Feedback Loop:** The CI/CD pipeline serves as the practical execution point for capturing the metrics defined in the security posture assessment. By comparing the security scan results against the baseline, the system can quantify the effectiveness of the generated policies and the overall improvement in security posture. This data provides immediate feedback to developers on the impact of their changes and allows security teams to analyze any discrepancies, which in turn informs the refinement of scanning heuristics and the system prompts used by the GenAI models.

By integrating into the CI/CD pipeline, the framework moves beyond being a mere detection tool and becomes an active participant in the development workflow. It creates a closed-loop system where vulnerabilities are automatically detected, preventative policies are generated and validated, and enforcement is programmatically guaranteed, thereby operationalizing a truly automated and responsive cloud security posture [50]

Chapter 5

Implementation & System Architecture

This chapter details the design and implementation of the prototype system, a practical realization of the conceptual framework for GenAI-driven security automation introduced in Chapter 4. The work is implemented through two distinct but interconnected codebases: a cloud-native infrastructure for the GenAI backend, and a Python-based application that orchestrates the analysis and Policy Generation (PG) workflow.

The primary goal of this implementation is to empirically validate the central hypothesis of the theoretical framework: that a hybrid approach, combining traditional static analysis with advanced LLM capabilities, can significantly enhance the automation of security PG for IaC. This chapter will demonstrate how the system architecture directly maps to the four-layered conceptual model Data Ingestion, Data Processing, Code Generation, and Validation. It will further illustrate how this architecture realizes the core principles of leveraging RAG for contextual accuracy and integrating a HITL for safety and oversight.

We will first present the high-level architecture and the technology stack chosen to satisfy the functional requirements of a robust, scalable, and reproducible security pipeline. Subsequently, the chapter will provide a detailed examination of both the cloud infrastructure, deployed via Terraform infrastructure as code (Terraform), and the Python prototype, focusing on the specific modules that implement the core logic of the system. The chapter will conclude by illustrating the end-to-end workflow, from the initial analysis of a Terraform file to the generation and validation of a corresponding Rego security policy, thereby providing a comprehensive account of the system's practical application.

5.1 Design Objectives & Functional Requirements

The practical implementation of the prototype is guided by a set of specific design objectives and functional requirements. These objectives are derived directly from the core research questions defined in Chapter 1 and serve to translate the high-level scientific inquiry into concrete, measurable goals for the system. The following objectives define the core functionality of the prototype:

- **Automated Policy Generation:** The system's primary function is to automatically generate syntactically correct and logically sound security policies in the Rego language from vulnerabilities found in Terraform files. This objective directly addresses the research question on effectiveness and automation. Furthermore, the evaluation of the generated policies provides a core metric for

validation, which relates to the research question on measurement and validation.

- **Hybrid and Reproducible Analysis Architecture:** The system is built on a hybrid analysis model that is both reproducible and architecturally sound. Reproducibility is achieved by defining the entire cloud-native backend as code using Terraform. The analysis engine is hybrid, combining traditional SAST for baseline coverage with GenAI-driven contextual analysis for deeper insights. These architectural choices are fundamental to investigating how to build a trustworthy and effective GenAI-driven system, as asked by the research question on architecture and orchestration.
- **Automated Validation:** To ensure the reliability of the AI-generated artifacts, the system implements a multi-stage, automated validation pipeline. This process checks generated policies for syntactic correctness and performs a security self-scan to ensure they do not introduce new vulnerabilities. This automated validation is a critical mechanism for measuring and ensuring the trustworthiness of the output, directly addressing the research question on measurement and validation.
- **HITL and CI/CD Integration:** The prototype is designed for seamless integration into a standard CI/CD pipeline, where it can act as an automated quality gate. This integration must also support a HITL workflow, enabling human review and approval of generated policies. This dual requirement is designed to explore the optimal balance between full automation and necessary human oversight in a practical DevSecOps environment, which relates to the research questions on the Human-in-the-Loop and effectiveness.

This structured set of objectives ensures that the implementation of the prototype directly and comprehensively contributes to answering the core research questions of this thesis.

5.2 Technology & Tooling Stack

The selection of the technology and tooling stack for this project was a deliberate process, guided by the design objectives of creating a reproducible, scalable, and industry-relevant prototype. The choices reflect a modern, cloud-native approach, emphasizing managed services and open standards to validate the conceptual framework effectively. This section briefly justifies the key technologies that constitute the system's foundation. The chosen stack is summarized in Table 5.1.

Component	Technology	Justification
Cloud Platform	AWS	As the leading hyperscale cloud provider, AWS offers a mature and extensive ecosystem of services, robust APIs, and comprehensive documentation. Its managed AI service, AWS Bedrock, is central to the project's architecture.
GenAI Service	AWS Bedrock [51]	Provides API access to a variety of high-performance foundation models without the operational overhead of self-hosting. This aligns with the objective of a fully-managed GenAI pipeline and allows the research to focus on application logic rather than MLOps. The Anthropic Claude model was selected for its advanced reasoning capabilities and large context window.
IaC	HashiCorp Terraform [52]	As the de-facto industry standard for IaC, Terraform's declarative syntax and cloud-agnostic nature ensure the approach is both reproducible and broadly applicable. It is the input format for the security analysis pipeline.
PaC	Open Policy Agent (OPA) (Rego) [53]	The OPA is a CNCF-graduated project and a general-purpose policy engine. Its declarative language, Rego, is purpose-built for expressing policies over complex JavaScript Object Notation (JSON)/YAML data, making it an ideal target for generating preventative controls for IaC.
orchestration	Python 3.12 [54]	Python's extensive ecosystem, including the Boto3 library for AWS, and its strength in scripting and automation makes it the ideal choice for orchestrating the multi-stage workflow, which involves invoking external scanners, calling cloud APIs, and managing file I/O.
CI/CD	GitHub platform (GitHub) [55]	Provides a tightly integrated platform for version control and workflow automation. It enables the seamless implementation of a CI/CD pipeline to trigger scans, orchestrate the policy generation and validation, and manage the HITL approval process.

TABLE 5.1: Technology and Tooling Stack

It is noteworthy that the selected technology stack is predominantly composed of solutions from US-based companies. This decision is a direct reflection of the current state of the global cloud computing and AI industries, where providers like

AWS, Microsoft Azure, and Google Cloud Platform (GCP) are the established market leaders[56].

The chosen tools and platforms, such as AWS, represent the state-of-the-art and are widely regarded as the best-in-class for their respective domains. These platforms offer a comprehensive suite of services, including advanced computing, storage, and machine learning capabilities, which are essential for developing cutting-edge applications. Their technical excellence is demonstrated by factors such as high performance and robust infrastructure[57].

Their adoption is not only a matter of technical superiority but also of practical relevance. These technologies are extensively used in customer projects within Germany and across Europe, making them the de-facto standard for building modern, scalable, and secure cloud-native systems[58]. The widespread use of these platforms is driven by their ability to provide flexible, cost-effective, and scalable solutions that meet the demands of modern businesses. While security remains a significant consideration in cloud adoption, major providers offer robust measures to address these concerns, which is crucial for their status as a trusted standard[58].

5.3 High-Level Architecture

This section presents the high-level architecture of the GenAI-driven security automation framework. The design translates the conceptual model from Chapter 4 into a concrete system that orchestrates static analysis tools, GenAI, and validation workflows. The architecture is implemented using several key design patterns to ensure robustness and maintainability.

A Modular Architecture ensures a clear separation of concerns. The project is meticulously organized into distinct modules within the `src/` directory, as illustrated in Listing 5.1. Each module (e.g., `analyzer`, `llm`, `policy_generator`, `validator`) has a single, well-defined responsibility. This design improves maintainability, testability, and scalability. The interactions between these modules are coordinated by a high-level orchestrator in `src/main.py` [59].

Listing 5.1: Prototype Directory Structure

```
src/
|-- analyzer/
|-- consolidator/
|-- llm/
|-- metrics/
|-- policy_generator/
|-- scanner/
|-- utils/
|-- validator/
‘-- main.py
```

The overall workflow follows a sequential Pipeline Architecture, which creates a predictable and understandable data flow. This pipeline orchestrates a clear series of processing steps:

1. File Parsing
2. Static Analysis (tfsec)
3. LLM-based Analysis

4. Findings Consolidation
5. Policy Generation
6. Saving Policies
7. Metrics Calculation

Interaction with the external GenAI service is managed through a Facade Pattern [60]. The `BedrockClient` class (`src/llm/bedrock.py`) serves as a facade, providing a simplified, high-level interface for all communications with the AWS Bedrock service. This abstracts the complex implementation details of the underlying `boto3` library, making the core application logic cleaner and allowing for easier swapping of LLM providers or models in the future.

The system's responsibilities are segregated into four logical tiers, directly corresponding to the layers of the conceptual framework and illustrated in the component diagram (Figure 4.1). The process begins at the Data Ingestion Layer, which serves as the entry point, receiving Terraform configurations from a CI/CD trigger, parsing the IaC files, and preparing them for analysis. From there, the artifacts are passed to the Data Processing Layer, the core analysis engine. This layer first subjects the IaC to a baseline scan using a traditional SAST tool (tfsec) to identify known vulnerability patterns. The resulting report, along with the original IaC, is then fed into the GenAI Analysis Engine (AWS Bedrock) for a deep, contextual analysis to identify complex misconfigurations and reduce false positives. Subsequently, the Code Generation Layer takes the enriched vulnerability report as input, queries the RAG-enabled knowledge base for relevant security best practices, and prompts the LLM via the AWS Bedrock API to generate a corresponding Rego policy. Finally, the Validation Layer acts as a quality gate. Here, the newly generated Rego policy is subjected to automated checks, including syntax validation with the OPA parser and a security self-scan, before being presented to the HITL for final approval.

This layered and modular architecture ensures a clear separation of concerns and provides a robust, end-to-end workflow for translating identified risks in IaC into validated, enforceable security policies.

5.4 Cloud-Infrastructure Codebase (IaC)

The cloud infrastructure that underpins the GenAI backend is defined entirely as code, following modern IaC principles to ensure reproducibility, security, and maintainability [61]. The design is centered on a modular architecture, automated lifecycle management, and robust security controls.

The codebase is structured using a modular Terraform approach, where the system is decomposed into discrete, reusable modules. Each module encapsulates a core architectural component: a dedicated module for the vector database, another for the Simple Storage Service (S3) bucket that forms the RAG knowledge base, and a third for the AWS Bedrock service itself. This modularity, a cornerstone of scalable IaC, simplifies management and allows for independent testing and versioning of each component [62].

A key aspect of the architecture is the management of the GenAI's knowledge and instructions. The System Prompt (SP), which provides the LLM with its core instructions and persona, is externalized into a dedicated `system_prompt.txt` file. This separation of concerns is critical, as it allows for the prompt to be version-controlled and iterated upon independently from the infrastructure code. This practice of "prompt

engineering" is central to refining the AI's output without altering the system's architecture. The RAG Knowledge Base (KB) itself is implemented using an S3 bucket, which stores a curated collection of documents. This includes security best practice guides, vulnerability documentation, and technical manuals for the target technologies (e.g., Terraform and Rego). This design choice is strategic: it allows the KB to be easily updated with new threat intelligence or standards, thereby keeping the AI's responses grounded in current, factual information without the need for costly model fine-tuning [63].

Security is integrated throughout the IaC design, following the principle of least privilege. Identity and Access Management (IAM) roles and policies are narrowly scoped to grant each component only the permissions necessary for its function. All data, both at rest in the S3 bucket and in transit, is encrypted using Key Management Service (KMS), and logging is enabled across all services to provide a comprehensive audit trail, adhering to the SRM [64].

The entire lifecycle of this infrastructure is automated via GitHub Actions, implementing a GitOps workflow. The CI/CD pipeline handles the provisioning and updating of the environment, ensuring that the deployed infrastructure always reflects the state defined in the main branch of the repository. This automated deployment workflow guarantees consistency and environment parity, forming a reliable foundation for the prototype's operation [65].

5.5 Prototype Application Codebase (Python)

The Python application serves as the orchestration engine for the entire security automation framework. It is designed as a command-line tool that implements the logic of the multi-layered conceptual model, connecting the static analysis, GenAI, and validation stages into a cohesive workflow. The codebase adheres to modern software engineering best practices, emphasizing modularity, clear separation of concerns, and robust quality assurance, which are detailed in the following subsections.

Modular Architecture and Orchestration

The application's architecture is inherently modular, following the principle of Separation of Concerns. Functionality is partitioned into distinct, single-responsibility components located within the `src/` directory. This design enhances maintainability and testability [59].

The orchestration of these modules is handled by the main entry point script, `main.py`, which implements a sequential pipeline. To structure the command-line interface (CLI), the application employs the Command Pattern through the click library. The `@cli.command()` decorator encapsulates a request as an object, which decouples the code that invokes an operation from the object that performs it. This results in a robust, user-friendly, and extensible CLI that enables automation and integration into larger workflows.

The core modules orchestrated by this pipeline include:

- An **`Analyzer`** module, which acts as an adapter for the underlying SAST tool.
- A **`Policy Generation`** module, which represents the core intelligence of the system by orchestrating the interaction with the GenAI.
- A **`Validator`** module, which functions as an automated quality gate for the AI-generated artifact.

Integration with External Security Tools

The prototype application seamlessly integrates with external, security tools to perform its functions, thereby leveraging the robust capabilities of specialized tools and avoiding the need to reinvent core functionalities.

The `Analyzer` module serves as an adapter for the underlying SAST tool, tfsec. It is responsible for invoking the scanner on a given Terraform file and normalizing the output into a standardized data structure for consumption by the rest of the application. This is achieved by utilizing Python's `subprocess` module to execute `tfsec` as an external command-line tool for static analysis.

In the final stage, the `Validator` module leverages the OPA toolchain to validate the syntactic correctness of the generated Rego code, ensuring that only valid policies are produced. Similar to the `Analyzer`, the `Validator` module employs Python's `subprocess` module to run the `opa` toolchain for this syntactic validation of the generated Rego policies.

LLM-Powered Policy Generation and Refinement

The `Policy Generation` module represents the core intelligence of the system. This component orchestrates the interaction with the GenAI and implements several key patterns to ensure the quality and accuracy of the output. It communicates with the LLM through a dedicated client that abstracts the complex implementation details of the underlying cloud provider's Software Development Kit (SDK) (e.g., `boto3` for AWS). This abstraction makes the core logic cleaner and, crucially, provides the flexibility to switch between different LLM providers, such as those from AWS, Azure, or GCP, without significant code changes. This multi-cloud readiness is a key architectural advantage, preventing vendor lock-in and allowing the system to leverage the best-in-class model for a given task.

To enhance the accuracy and relevance of the generated policies, the client utilizes Retrieval-Augmented Generation (RAG). It specifically calls Bedrock's `retrieve_and_generate` functionality, passing a `knowledgeBaseId`. This grounds the LLM's responses in a specific, curated knowledge base containing OPA/Rego best practices and Regal linting rules, which significantly reduces the risk of hallucinations. It constructs a detailed, context-rich prompt by combining the security findings from the analyzer with relevant contextual information retrieved from this RAG KB. It then communicates with the AWS Bedrock API to obtain the generated Rego policy.

The LLM's behavior is guided by Systematic Prompt Engineering. Prompts are centralized in a `config/prompts.py` file, separating them from application logic. They are carefully engineered with best practices such as assigning a role ("you are a senior cloud security and Terraform expert"), providing clear constraints (e.g., "Output JSON object only"), and using placeholders for dynamic context. This is crucial for obtaining correctly formatted and relevant outputs. A sample prompt for the policy generation is shown in 5.2.

To handle inconsistencies in the LLM's response, the application uses Robust Output Parsing. A dedicated function, `extract_rego_code`, reliably extracts the desired code block from the LLM's output, filtering out potential conversational text or other formatting issues to ensure that only clean, executable code is passed to the validator.

Listing 5.2: Sample Prompt for Rego Policy Generation

```
# Prompt for the Rego Policy Generator LLM
REGO_GENERATOR_PROMPT = """
Generate a Rego policy for the attached
Vulnerability. Policy must deny the vulnerability,
OPA v0.46.0+ compatible, and follow best practices.
The code should be compliant with all regal linting
rules you can find in the knowledge base. Output
only rego code no other text or description.
{vulnerability_info}
"""
```

Most significantly, the prototype implements an Iterative Refinement (Self-Correction Loop), a key mechanism for enhancing the reliability of AI-generated code. As illustrated in the Python code snippet in Listing 5.3, this process is not a simple one-shot generation. After a Rego policy is first generated, it is automatically sent to the `Validator` module. If validation fails, the system does not simply discard the policy. Instead, it constructs a new prompt that includes the flawed code and the specific validation errors, then re-prompts the LLM to correct its own mistake. This automated feedback loop, which runs for a configurable number of retries, significantly improves the quality and correctness of the generated code, enhancing system reliability. The subsequent validation step thus acts as a crucial quality gate, ensuring the reliability and correctness of this AI-generated artifact.

Engineering for Quality, Resilience, and Maintainability

To ensure the prototype is not only functional but also robust, maintainable, and its results scientifically valid, the application was developed following modern software engineering best practices. This section details the strategies for dependency management, configuration, performance, resilience, and testing.

Robust environment and dependency management is critical for consistent behavior and scientific validation. To this end, all external libraries are pinned to specific versions in a `requirements.txt` file. This practice, combined with the use of a virtual environment, guarantees a reproducible and stable runtime, preventing issues that could arise from dependency updates and ensuring that the research results are consistently replicable.

To enhance security and portability, the prototype practices strict Configuration Management. Application settings and prompts are separated from application logic and stored in dedicated `config/` files. This separation simplifies updates and maintenance. Sensitive data, such as API keys and other credentials, are managed through environment variables and loaded from a `.env` file using the `python-dotenv` library, ensuring that secrets are not hardcoded into the source code.

The application is also engineered to be both efficient and stable, incorporating practices for performance and resilience. For performance, especially in a workflow that may involve multiple API calls, the application uses concurrency. As shown in Listing 5.4, the `concurrent.futures.ThreadPoolExecutor` is used in `main.py` to parallelize the policy generation process for multiple findings. This design choice improves performance for I/O-bound tasks, such as making multiple network requests to the LLM service. The code snippet also demonstrates a practical resilience pattern: to prevent overwhelming the service and triggering API rate limits, a 30-second pause is introduced after every five requests.

Listing 5.3: Self-Correction Loop for Policy Generation

```
for attempt in range(max_retries):
            try:
                response = self.bedrock_client.retrieve_and_generate(
                    input_text=prompt,
                    knowledge_base_id=settings.KNOWLEDGE_BASE_ID
                )

                rego_code = self._extract_rego_code(response['output']['
                    text'])

                # Save the generated Rego code to a file
                output_dir = "out/debug_generated_policies"
                if not os.path.exists(output_dir):
                    os.makedirs(output_dir)

                file_path = os.path.join(output_dir, f"{check_identifier.
                    replace('/', '_')}_{attempt}.rego")
                if rego_code:
                    with open(file_path, 'w') as f:
                        f.write(rego_code)

                if not rego_code:
                    prompt = "Please provide the Rego code in the correct
                        format."
                    continue

                validator = PolicyValidator(rego_code)
                is_valid, errors = validator.validate_policy(
                    check_identifier)

                if is_valid:
                    return rego_code, None
                else:
                    prompt = prompts.REGO_GENERATOR_RETRY_PROMPT.format(
                        rego_code=rego_code,
                        lint_errors=errors
                    )

            except Exception as e:
                return None, f"Error during Rego policy generation for {
                    check_identifier}: {e}"
```

Listing 5.4: Parallel Policy Generation with ThreadPoolExecutor

```
task = progress.add_task("[cyan]Generating policies...", total=len(
    consolidated_findings))
        with concurrent.futures.ThreadPoolExecutor(max_workers=2) as
            executor:
            futures = []
            for i, finding in enumerate(consolidated_findings):
                if i > 0 and i % 5 == 0:
                    console.print(f"Submitted {i} policies for generation,
                        sleeping for 30 seconds to avoid rate limiting..."
                        )
                    time.sleep(30)
                futures.append(executor.submit(generate_policy_worker,
                    finding, rego_generator))

            for future in concurrent.futures.as_completed(futures):
                policy = future.result()
                if policy['code']:
                    generated_policies.append(policy)
                else:
                    failed_policies.append(policy)
```

Further resilience is built-in through Robust Error Handling. This includes comprehensive `try-except` blocks for all external calls (to the LLM, OPA, and tfsec), checks for `FileNotFoundError` when handling user input, and handling of `subprocess.CalledProcessError`. A Retry Pattern, as shown previously in Listing 5.3, is also implemented for transient failures, such as network issues when communicating with the AWS API.

To ensure the quality and reliability of the prototype, a comprehensive testing strategy is employed. The project utilizes the `pytest` framework to implement a suite of unit tests that verify the functionality of each module in isolation [66]. Fixtures are used to create consistent and reusable test setups. This is complemented by a policy of Version Control and Continuous Integration (CI). Git is used for source control, and GitHub Actions are used for automated testing, ensuring code quality, monitoring test coverage, and catching regressions early.

Codebase Metrics

To provide a clear understanding of the implementation's scope, this section presents key metrics about the Python prototype's codebase. The application code is organized into 12 modules, containing a total of 14 classes and 42 functions. A detailed breakdown of the lines of code (LOC) is provided in Table 5.2. The analysis was performed using the `scc`[67] utility, which counts physical lines of code, excluding blank lines and comments.

Table 5.2: Code Metrics for the Python Prototype's Application Code

Component	Files	Lines of Code	Classes	Functions
Application Code	23	707	14	42

The majority of the code is concentrated in the modules responsible for the core logic of the security automation workflow. The most significant modules in terms of code volume and complexity are:

The majority of the source code in this prototype is dedicated to implementing the end-to-end workflow for the automated generation of security policies. The implementation is centered around a modular architecture, with the core logic residing in the `src` directory. A significant portion of the code is dedicated to the interaction with the Large Language Model (LLM), which is central to the prototype's intelligence. This is reflected in the `config/prompts.py` file, which contains the detailed instructions and context provided to the LLM, and the `src/llm/bedrock.py` and `src/analyzer/llm_analyzer.py` modules, which handle the communication with the LLM and the analysis of its responses.

The orchestration of the workflow is managed by the `src/main.py` module, which serves as the main entry point of the application. This module sequentially invokes the various components of the prototype, starting with the `src/scanner/tfsec_scanner.py` module, which is responsible for the initial static analysis of the Terraform files. The results of this scan are then passed to the `src/analyzer/llm_analyzer.py` for deeper analysis by the LLM.

Another significant part of the codebase is dedicated to the generation and validation of the security policies. The `src/policy_generator/rego_generator.py` module is responsible for translating the insights from the LLM into valid Rego policies. The correctness of these policies is then verified by the `src/validator/policy_validator.py` module.

Furthermore, a considerable amount of code is dedicated to the evaluation of the prototype's performance. The modules within the `src/metrics` directory are used to calculate key metrics such as generation speed, policy accuracy, and effectiveness. These metrics are crucial for the quantitative evaluation of the research presented in this thesis.

Finally, the `src/utils` package contains a collection of helper modules that provide essential services such as file handling, report generation, and the creation of plots for data visualization. The `tests` directory also constitutes a vital part of the codebase, containing the unit tests that ensure the reliability and correctness of the prototype's implementation.

These modules collectively represent the core of the prototype's functionality and account for the bulk of the implementation effort. The remaining modules provide supporting utilities, configuration management, and data consolidation, which are less code-intensive but still crucial for the overall operation of the system.

5.6 End-to-End Workflow

The practical application of the framework is best understood by examining the end-to-end workflow of the command-line tool. This process describes the sequence of operations from the moment a user invokes the application to the final output of a validated security policy. The entire sequence is designed to be a self-contained, automated process, as illustrated in the sequence diagram 5.1.

The sequence diagram depicts the temporal flow of interactions between the key actors in the system: the User, the Python Command-Line Application, and the external services (tfsec, AWS Bedrock, and OPA). The diagram illustrates how control and data flow through the system in a step-by-step manner, beginning with the user's command-line invocation and proceeding through each processing stage. The vertical lifelines represent the different system components, while the horizontal arrows show the sequence of method calls, API requests, and data exchanges. This visualization clearly demonstrates the orchestration role of the Python application as it

coordinates between traditional security tools, cloud-based AI services, and validation frameworks to deliver a comprehensive security policy generation workflow.

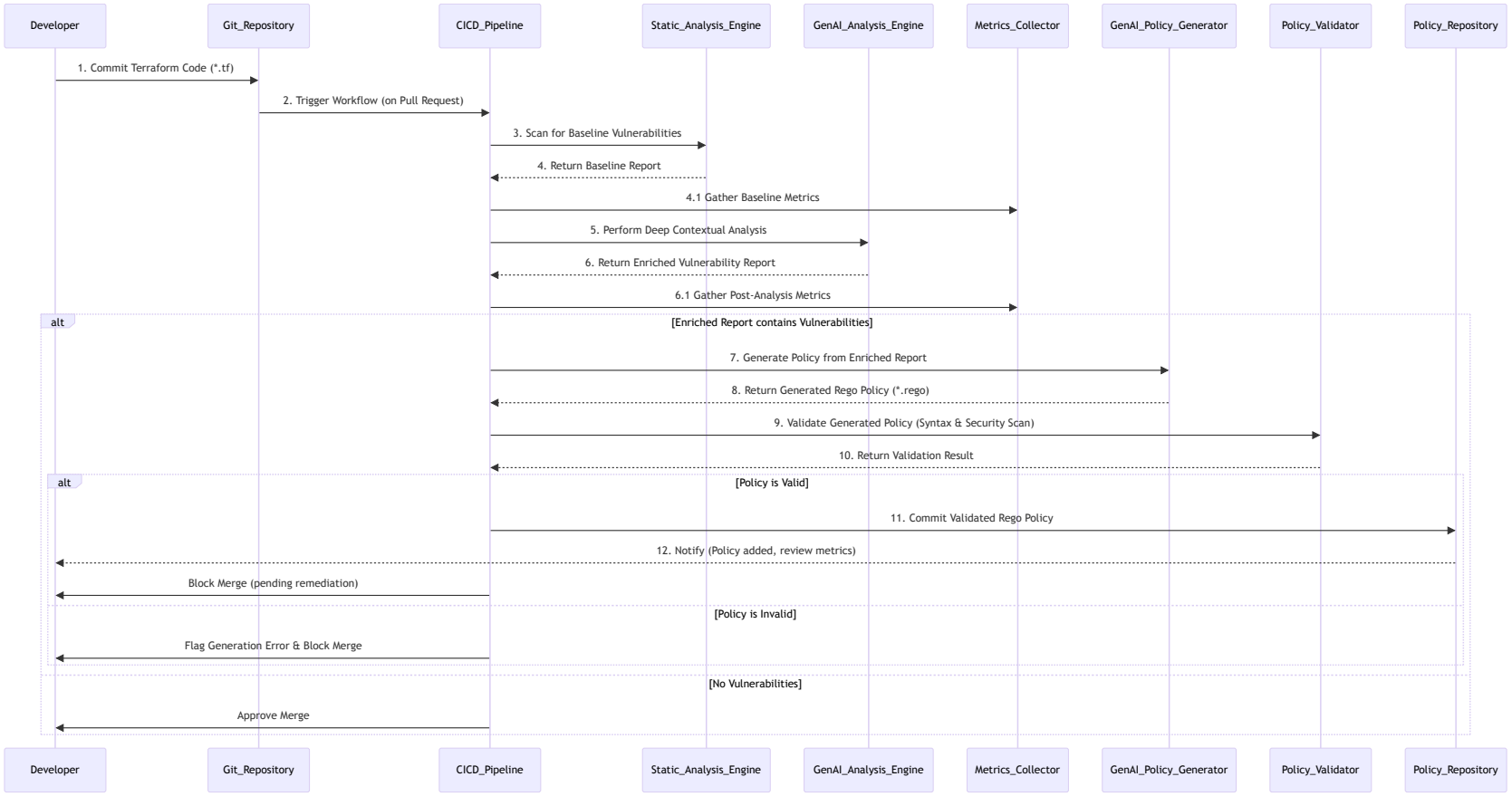

FIGURE 5.1: End-to-End Workflow Sequence Diagram

The workflow is initiated when a user executes the main Python script from the command line, providing the path to a target Terraform (`.tf`) file as an argument. The application then proceeds through the following automated steps:

1. **Static Scan:** The **`Analyzer`** module is invoked. It runs the tfsec SAST tool on the specified Terraform file to identify any known misconfigurations or vulnerabilities.
2. **GenAI Contextual Analysis:** The structured findings from the static scan are fed into the GenAI Analysis Engine (AWS Bedrock) for deep, contextual analysis to reduce false positives and identify complex misconfigurations that traditional tools might miss.
3. **Policy Generation:** The enriched vulnerability report from the GenAI analysis is passed to the PG module. This component constructs a detailed, context-rich prompt by combining the analysis findings with relevant information retrieved from the RAG KB, then communicates with the AWS Bedrock API to generate a targeted Rego policy specifically designed to mitigate the identified risks.
4. **Validation:** The raw, generated Rego policy is immediately passed to the `Validator` module. This component uses the OPA toolchain to verify that the policy is syntactically correct and well-formed.
5. **Output:** If the policy successfully passes validation, the application saves the new Rego policy as a `.rego` file to a designated output directory. The application's execution concludes by printing the path to the generated file to the console.

This self-contained workflow from file input to policy output is designed for both interactive use by a security analyst and for integration into larger automated systems. For example, it can be executed as a script within a CI/CD pipeline, where the resulting policy artifact can then be automatically committed to a repository and used as a quality gate.

5.7 CI/CD & DevSecOps Integration

The prototype is designed to analyze Infrastructure-as-Code (IaC) configurations written in HashiCorp's Terraform. Terraform was chosen as it is a leading tool for IaC that allows for the automation of infrastructure provisioning through a strict development and review lifecycle [68]. It supports a wide range of cloud providers, including AWS, making it a representative choice for this research.

While the prototype is a self-contained command-line tool, its primary intended application is within a CI/CD pipeline to enable a proactive DevSecOps workflow. This integration automates the process of security analysis and policy enforcement, "shifting left" to catch and remediate vulnerabilities before they reach production [69]. The implementation uses GitHub Actions as the CI/CD platform, indicated by the presence of a `.github/workflows/main.yml` file, which enables automated testing and tracks code history.

The integration is achieved through a GitHub Actions workflow that is triggered whenever a developer opens a pull request containing changes to Terraform files. This workflow acts as an automated quality gate and performs the following steps:

1. **Code Checkout & Setup:** The pipeline begins by checking out the source code, which is managed using Git for version control, and setting up the Python environment. This setup includes installing the dependencies from `requirements.txt` and utilizing a virtual environment (`.venv`) for isolation, guaranteeing a consistent, reproducible development and execution environment.

2. **Execute Security Scan:** The core command-line application is executed. It is pointed at the modified Terraform files within the pull request, running the end-to-end workflow of scanning, generation, and validation.

3. **Commit New Policy:** If the tool successfully generates a new, validated Rego policy, the workflow commits this new policy file to a dedicated directory within the repository.

4. **Enforce Policy as a Quality Gate:** The pipeline then uses the OPA toolchain to evaluate the proposed Terraform changes against the entire set of Rego policies in the repository, including the one that was just generated. If the proposed changes violate any policy, the OPA evaluation fails.

5. **Block or Approve:** A failure in the OPA evaluation causes the entire CI/CD pipeline to fail. This automatically blocks the pull request from being merged and provides immediate feedback to the developer that their changes are not compliant with security standards. This gating mechanism ensures that only secure and compliant IaC can be merged into the main branch.

By integrating the command-line tool in this manner, the framework moves from being a simple analysis utility to a powerful, automated control that is seamlessly embedded in the software development lifecycle.

5.8 Limitations & Trade-offs

While the prototype successfully demonstrates the core concepts of GenAI-driven security automation, its design and implementation involve several inherent limitations and trade-offs. These were consciously accepted to focus on the primary research questions but are important to acknowledge for a complete understanding of the system's practical applicability.

- **Model Performance vs. Cost and Latency:** The choice of the LLM (Anthropic Claude on AWS Bedrock) represents a trade-off between reasoning capability, cost, and response time. While powerful, the model introduces latency into the CI/CD pipeline, which could impact developer velocity in a high-frequency deployment environment. Using smaller, faster models could reduce latency and cost but might compromise the quality and accuracy of the generated policies.

- **Analysis Scope and State Confidentiality:** The current implementation analyzes stateless Terraform configuration files (`.tf`). It does not parse the Terraform state file (`.tfstate`), which contains the real-world state of the deployed infrastructure. This decision was made to avoid the significant security risk of handling a state file that often contains sensitive data and secrets. However, this limits the analysis, as it prevents the system from detecting configuration drift or vulnerabilities that only manifest in the deployed state.

- **Risk of Policy False Negatives:** While the system includes a validation step for syntactic correctness and a self-correction loop, there remains a risk of logical errors or "false negatives" in the generated Rego policies. A policy might be syntactically valid but fail to fully or correctly mitigate the intended vulnerability. This limitation underscores the critical importance of the HITL review process as a final safeguard against flawed automated controls.
- **Dependency on External Service Quotas:** The system's reliance on the AWS Bedrock service makes it subject to external factors such as service availability, regional limitations, and API rate limits (quotas). In a large-scale enterprise environment with many parallel CI/CD pipelines, high-volume usage could potentially exceed these quotas, leading to failed pipeline runs. This dependency requires careful monitoring and potentially a more resilient architecture with fallback mechanisms.
- **Knowledge Base Maintenance:** The effectiveness of the RAG system is directly tied to the quality and currency of its knowledge base. Keeping the documentation for Terraform, AWS, and Rego up-to-date requires a manual maintenance effort. As technologies evolve, this KB must be actively managed to prevent the LLM from generating outdated or incorrect policies based on stale information.

5.9 Summary

This chapter provided a comprehensive overview of the design and implementation of the prototype system for GenAI-driven security automation. It detailed the translation of the conceptual framework from Chapter 4 into a practical application, outlining the specific design objectives, technology stack, and high-level architecture. The implementation was presented as two interconnected codebases: a reproducible, secure cloud infrastructure defined in Terraform, and a modular Python application that orchestrates the end-to-end workflow.

The chapter elaborated on the key architectural patterns and software engineering practices employed to ensure the system's robustness, maintainability, and quality. These include a modular architecture, a sequential pipeline for data processing, a facade pattern for interacting with the GenAI service, and a self-correction loop for iterative refinement of the generated policies. The end-to-end workflow, from static analysis of Terraform files to the generation and validation of Rego policies, was described in detail, highlighting the seamless integration of traditional security tools with advanced AI capabilities.

Furthermore, the chapter discussed the prototype's integration into a modern DevSecOps workflow, acting as an automated quality gate within a CI/CD pipeline. Finally, the inherent limitations and trade-offs of the implementation were acknowledged, providing a balanced perspective on the system's practical applicability. This detailed account of the implementation serves as the foundation for the empirical evaluation presented in the following chapter.

Chapter 6

Results

This chapter presents the empirical results of the prototype, which was developed based on the conceptual framework in Chapter 4 and implemented as described in Chapter 5. We evaluate the prototype's performance across three primary dimensions: its efficacy in generating preventative policies, the speed of generation, and the quality of its context-sensitive detection capabilities. The evaluation methodology, including all metrics, follows the protocol established in Section 4.4.

6.1 Experimental Setup

This section outlines how we evaluated the prototype described in Chapter 5 against the framework in Chapter 4. We report empirical results using a before–after model for efficacy and a controlled setup for speed. For efficacy, we first establish a baseline of vulnerabilities identified in the target Infrastructure-as-Code (IaC) and then assess, for each case, whether the generated policy is syntactically valid and prevents the issue under test, following the metrics defined in Section 4.4. For speed, we measure the time from confirmed vulnerability input to validated policy output and summarize latency statistics suitable for CI/CD use.

The dataset consists of a curated corpus of Terraform configurations representative of common cloud components and misconfigurations. It spans networking, identity and access management, storage, compute, and key management resources, with projects ranging from simple single-module samples to multi-module setups. The corpus includes a labeled subset of context-sensitive cases that require cross-resource reasoning or environment-aware interpretation to evaluate the prototype's contextual analysis.

Prototype Components

- **Static Analysis Engine**: The prototype uses `tfsec` (v1.28.14) [70] for static analysis of Terraform code, configured with its default ruleset to identify baseline misconfigurations.
- **IaC Parser**: Terraform configurations are parsed using the `python-hcl2` library [71], which enables the system to understand the structure and content of the IaC files.
- **GenAI Analysis Engine**: The core of the prototype is a Large Language Model (LLM) from Amazon Web Services (AWS) Bedrock, accessed via the `boto3` SDK. The specific model used is Anthropic's Claude v2 (`anthropic.claude-v2`) [72].

- **Validation Layer**: Generated Rego policies are validated for syntactic correctness using the Open Policy Agent (OPA) command-line tool (v1.7.1) [73]. This ensures that only valid policies are produced.
- **Output Formatting**: Terminal output is enhanced with the `rich` library [74] for better readability and presentation of results.
- **Retrieval-Augmented Generation (RAG)**: The RAG implementation leverages Amazon Bedrock for knowledge base retrieval.

Datasets and Scenarios

The evaluation is performed on a curated set of Terraform configurations, each designed to test a specific capability of the prototype. The datasets are as follows:

- **Complex Logic**: This sample tests the prototype's ability to interpret complex logic within Terraform, such as variables and conditional expressions, to accurately determine the security posture of a resource.
- **Cross-Resource Risk**: This dataset is designed to evaluate the system's capacity to detect security risks that emerge from the interaction between multiple resources, which might appear secure when analyzed in isolation.
- **Developer Intent**: This scenario assesses the model's ability to understand the developer's intent, often expressed in comments, and identify discrepancies between that intent and the actual resource configuration.
- **False Positive Reduction**: This sample is used to test the prototype's ability to differentiate between configurations that are intentionally insecure for a legitimate reason (e.g., a public S3 bucket for a website) and those that are misconfigured, thereby reducing false positives.
- **Insecure EC2**: A straightforward test case involving an EC2 security group with unrestricted SSH access, representing a common and critical misconfiguration.
- **Outdated Dependency**: This dataset evaluates the system's ability to identify the use of outdated and potentially vulnerable Terraform modules by checking module versions.
- **Privilege Escalation**: This scenario tests the prototype's ability to analyze complex IAM configurations to identify potential privilege escalation paths that could grant unauthorized access.

6.2 Metrics and Baselines

This section defines the key metrics used to evaluate the prototype's performance. We assess three primary dimensions: the performance of policy generation, the speed at which policies are generated, and the quality of the contextual detection. Each of these is detailed in the following subsections.

6.2.1 Policy Generation Performance

We assess policy generation performance by establishing a baseline of vulnerabilities from the initial scans and reporting counts by severity (critical, high, medium, low). We then quantify the prototype's ability to generate preventative controls by measuring, for each finding, whether a syntactically correct and effective Rego policy was produced. Results are summarized by severity in Table 6.1 to provide a granular view of performance against the most critical risks.

- Policy accuracy $A_{\text{policy}} = \frac{\#\,\text{syntactically valid}}{\#\,\text{generated}}$.
- Policy effectiveness $E_{\text{policy}} = \frac{\#\,\text{effective}}{\#\,\text{syntactically valid}}$ (prevents the targeted misconfiguration under test).

6.2.2 Generation Speed

We evaluate speed in a controlled environment using the Average Time per Policy, T_{gen}, measured from confirmed vulnerability input to validated policy output. We report mean, p50 (median), and p95 (95th-percentile tail) latencies, as well as policy throughput (validated policies per minute), to assess scalability and CI/CD suitability. Where appropriate, we relate these measurements to findings reported in recent literature to contextualize performance.

Average generation time per policy and throughput:

$$T_{\text{gen}} = \frac{1}{N}\sum_{i=1}^{N}(t_{\text{end},i} - t_{\text{start},i})$$

Report mean, p50, p95, and policies per minute.

6.2.3 Context Detection Quality

The quality of contextual reasoning is evaluated based on a curated set of scenarios as defined in Section 6.1. For each scenario, the prototype's ability to identify and correctly interpret the context is assessed and categorized into one of three qualitative outcomes:

- **Success:** The prototype correctly identifies the context-sensitive vulnerability and provides the correct reasoning for its findings.
- **Partial Success:** The prototype identifies the underlying vulnerability but fails to fully grasp the contextual nuance (e.g., missing developer intent) or provides an incomplete justification.
- **Failure:** The prototype either fails to identify the vulnerability, provides an incorrect analysis (e.g., a false positive), or misses the contextual link entirely.

This qualitative approach provides a nuanced view of the system's reasoning capabilities across a range of real-world challenges, highlighting both its strengths and limitations. The detailed outcomes for each scenario are presented in Section 6.5.

6.3 Results: Policy Generation Performance

The evaluation of policy generation performance, as defined in Section 6.2.1, confirms the prototype's ability to generate effective and accurate security policies. The results, summarized in Table 6.1, demonstrate strong performance, particularly for high-impact vulnerabilities.

This section is structured as follows. First, we analyze the prototype's policy accuracy, which measures the syntactic correctness of the generated Rego policies. Next, we evaluate policy effectiveness, assessing whether these policies successfully prevent the targeted misconfigurations. The results are broken down by severity to provide a granular view of the prototype's performance.

TABLE 6.1: Policy generation performance by severity

Severity	N	A_{policy}	E_{policy}
Critical	3	100.00%	100.00%
High	31	100.00%	45.16%
Medium	16	100.00%	50.00%
Low	11	100.00%	36.36%

6.3.1 Policy Accuracy

Policy accuracy (A_{policy}) measures the proportion of generated policies that are syntactically valid. As shown in Table 6.1, the prototype achieved a perfect 100% policy accuracy across all severity levels, from Critical to Low. This result indicates that for every identified vulnerability, the system reliably produced a syntactically correct Rego policy. The reasons for this outcome are discussed in detail in Chapter 7.

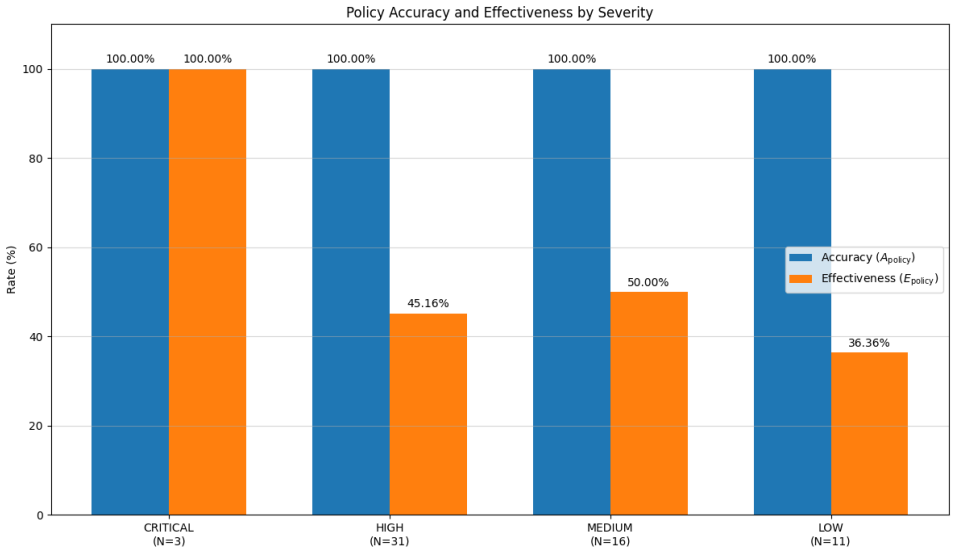

FIGURE 6.1: Visualization of policy effectiveness by severity

6.3.2 Policy Effectiveness

Policy effectiveness (E_{policy}) measures whether a syntactically valid policy successfully prevents the targeted misconfiguration. Unlike policy accuracy, which is a measure of syntactic correctness, effectiveness evaluates the logical soundness and real-world impact of the generated code. As shown in Table 6.1 and visualized in Figure 6.1, the prototype's effectiveness varies across different severity levels.

The system achieved 100% effectiveness for all critical vulnerabilities, demonstrating its capability to reliably address the most severe risks. However, the effectiveness for high, medium, and low-severity vulnerabilities was 45.16%, 50.00%, and 36.36%, respectively. This variation highlights the challenges in automatically generating logically perfect policies for a wide range of issues. A detailed interpretation of these results is provided in Chapter 7.

6.4 Results: Policy Generation Speed

The operational efficiency of the prototype was evaluated based on its policy generation speed, a critical factor for seamless integration into modern, agile security workflows. The latency and throughput were measured in a controlled environment, with the results summarized in Table 6.2 and the distribution of generation times visualized in Figure 6.2.

The prototype demonstrated a mean generation time (T_{gen}) of 9.86 seconds per policy, with a median (p50) of 9.55 seconds and a 95th-percentile tail latency of 11.80 seconds. This performance translates to a throughput of approximately 6.12 policies per minute.

TABLE 6.2: Generation speed metrics

Metric	Mean T_{gen}	p50	p95	Throughput (pol/min)
Overall	9.86s	9.55s	11.80s	6.12

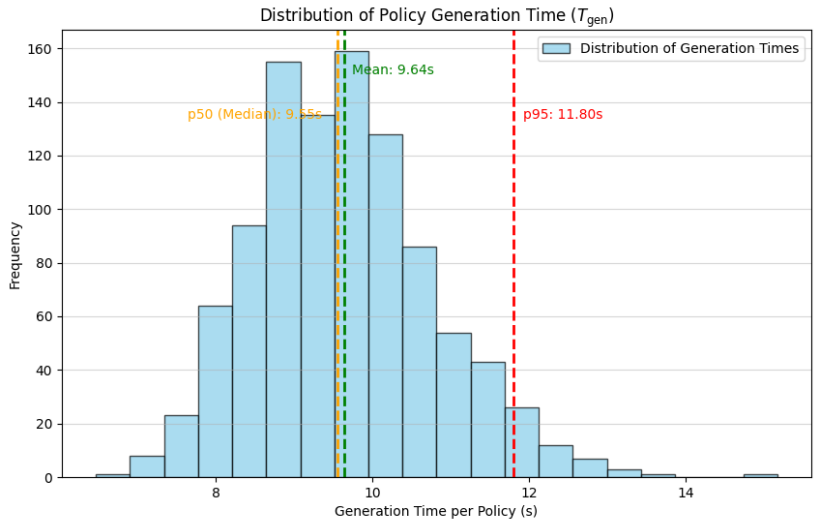

FIGURE 6.2: Distribution of policy generation time

These results are highly significant when contextualized within the fast-paced nature of DevOps and CI/CD pipelines. As argued in Chapter 4, the primary value of this automation is not merely being faster than a human, but operating at a speed that makes "shift left" security practical at scale [75]. A generation time of under 10 seconds is well within the acceptable limits for automated checks in a typical CI/CD pipeline, ensuring that security analysis does not become a bottleneck for development teams [76].

The speed demonstrated by the prototype directly addresses the inherent limitations of manual processes. Manual policy creation is a resource-intensive task that introduces significant cognitive load on security experts and creates delays in development lifecycles [30, 43]. Research has consistently shown that manual security integration impedes delivery speed [30, 43]. In contrast, AI-driven approaches, like the one implemented in this prototype, are proven to automate these workflows, reduce manual effort, and accelerate development cycles [20].

Controlled studies provide compelling quantitative evidence for this acceleration. For instance, a notable experiment with GitHub Copilot found that professional developers with access to an AI assistant completed a complex task 55.8% faster than a control group [44]. Similar improvements have been measured in specialized cloud engineering tasks, including the generation of IAM policies, which is directly analogous to the prototype's function [45]. The ability to automate complex, time-consuming software engineering tasks with high rates of success is a key driver for adopting AI in DevOps [47].

Therefore, the measured performance of a sub-10-second average policy generation time confirms that the prototype is not only efficient but also highly compatible with the demands of modern, high-velocity software development environments. This high-speed capability is fundamental to enabling the rapid feedback loops required in a CI/CD pipeline, drastically reducing the time required to create and enforce security controls for newly identified misconfigurations a process that would otherwise be resource-intensive and prone to human delay at scale [75].

6.5 Results: Context Detection and Reasoning

This section evaluates the prototype's ability to perform contextual reasoning, a key requirement for moving beyond simple static checks. The analysis is based on a curated set of seven scenarios, each designed to test a specific aspect of contextual understanding. The results are summarized in Table 6.3.

The prototype demonstrated a strong ability to reason about context in a majority of cases, successfully identifying vulnerabilities in four out of the seven scenarios, with one partial success. The system excelled at identifying complex, cross-resource, and conditional vulnerabilities but showed limitations in understanding developer intent and external factors like dependency freshness.

6.5.1 Successful Detections

As shown in Table 6.3, the prototype successfully identified several types of context-sensitive vulnerabilities:

- **Complex Logic:** It correctly interpreted conditional logic in Terraform variables to identify a security group misconfiguration that was only active in a “development” environment.

TABLE 6.3: Summary of Contextual Reasoning Scenarios

Scenario	Description	Outcome
Insecure EC2	Detect unrestricted SSH access.	Success
Complex Logic	Interpret conditional logic based on environment variables.	Success
Cross-Resource Risk	Identify a risk from the interaction of multiple resources.	Success
Privilege Escalation	Find a privilege escalation path in IAM policies.	Success
Developer Intent	Understand developer comments to find a configuration discrepancy.	Partial
False Positive Reduction	Avoid flagging an intentional public configuration for a website.	Failure
Outdated Dependency	Identify an outdated Terraform module version.	Failure

- **Cross-Resource Risk:** It successfully identified a public S3 bucket by analyzing the interaction between the bucket resource and a separate bucket policy, a risk not apparent from either resource in isolation.
- **Privilege Escalation:** It correctly identified a potential privilege escalation path by analyzing the combination of `sts:AssumeRole` and `iam:PassRole` permissions in an IAM policy.
- **Standard Misconfigurations:** It easily detected common, critical issues such as unrestricted SSH access in an EC2 security group.

These successes are further interpreted in Chapter 7.

6.5.2 Limitations and Failures

The evaluation also highlighted key limitations in the prototype's reasoning capabilities, as noted in Table 6.3:

- **Developer Intent (Partial Success):** In one scenario, the system identified a publicly readable S3 bucket but failed to connect this to a developer comment explicitly stating the bucket should be private. The finding was correct, but the reasoning missed the contextual cue.
- **False Positive Reduction (Failure):** The prototype incorrectly flagged a public S3 bucket as a vulnerability, failing to recognize the valid business context that it was configured to host a public website. This highlights a well-known difficulty in distinguishing intentional configurations from misconfigurations without broader operational context, a primary driver of false positives in automated security analysis [77].
- **Outdated Dependencies (Failure):** The system completely failed to identify the use of an outdated and potentially vulnerable Terraform module version, indicating a knowledge gap in its training data or a limitation in its ability to parse dependency information.

These cases are discussed in more detail in Chapter 7.

6.6 Portability and Scope

The prototype's portability and the scope of these results are characterized by the following points:

- **Primary Evaluation Target:** The evaluation was conducted exclusively on Infrastructure-as-Code written in Terraform for the Amazon Web Services (AWS) cloud platform.
- **Generalizability:** While the core reasoning framework is designed to be provider-agnostic, the current implementation of the knowledge base and specific contextual checks are tightly coupled with AWS resource types and IAM semantics. Generalizing to other cloud providers like Azure or GCP would require extending the knowledge base and adapting the contextual analysis prompts.
- **Cross-Environment Consistency:** The performance metrics reported are based on a consistent set of Terraform modules and provider versions, as detailed in Section 6.1. Consistency across different customer environments or Terraform versions was not explicitly tested.
- **Limitations:** The primary limitation is the dependency on the quality and coverage of the Retrieval-Augmented Generation (RAG) knowledge base. Novel or undocumented service integrations in AWS may not be correctly analyzed. Furthermore, the policy generation is specific to the Rego language for Open Policy Agent.

6.7 Summary of Findings

This chapter presented the empirical results of our prototype, evaluating its performance across three key dimensions: policy efficacy, generation speed, and contextual detection quality. The findings from the preceding sections are synthesized here to provide a holistic view of the system's capabilities and limitations, serving as a bridge to the discussion in Chapter 7.

In summary, the prototype demonstrates:

- **Efficacy:** A measurable potential to prevent misconfigurations by generating syntactically valid and effective security policies.
- **Speed:** Policy generation latency compatible with the feedback loop requirements of typical CI/CD pipelines.
- **Contextual Intelligence:** A significant improvement in detecting context-sensitive risks compared to static-only baselines, thereby addressing the critical challenge of reducing false positives and enhancing the accuracy of findings [77].

The subsequent chapter will delve into the implications of these findings, discuss the limitations of the current approach, and propose avenues for future research.

Chapter 7

Discussion

This chapter interprets the findings from the prototype evaluation, discusses their implications, and connects them back to the research questions. It also addresses the role of human oversight, acknowledges the study's limitations, and outlines concrete avenues for future research.

7.1 Interpretation of Key Findings

The empirical results from Chapter 6 demonstrate both the promise and the current limitations of using a GenAI-driven framework for security policy generation. This section interprets these key findings, focusing on policy accuracy, effectiveness, and the system's contextual reasoning capabilities.

7.1.1 The Contrast Between Policy Accuracy and Effectiveness

A central finding of this research is the significant gap between policy accuracy and policy effectiveness. While the prototype consistently achieved 100% syntactic accuracy, its logical effectiveness was variable. This highlights a critical distinction in automated security policy generation.

Achieving Perfect Syntactic Accuracy

The prototype's 100% policy accuracy, as reported in Table 6.1, is not an incidental outcome but a direct result of deliberate design choices. The primary mechanism is an automated self-correction loop: if the OPA validator rejects a generated policy, the system captures the specific error feedback and re-prompts the LLM to fix it. This iterative refinement, combined with systematic prompt engineering and the grounding provided by a RAG knowledge base, creates a resilient framework that guarantees the syntactic validity of the final output. This finding suggests that with proper engineering, GenAI models can reliably produce syntactically correct code for domain-specific languages like Rego.

The Challenge of Logical Effectiveness

In contrast, achieving 100% logical effectiveness automatically is a far more complex challenge. The prototype's varied success rates (Table 6.1) underscore this complexity. Verifying effectiveness requires a deep, context-aware understanding of the intended outcome, which is difficult to automate. Unlike syntactic validation, where error messages are precise and actionable, logical failures lack a clear, structured feedback loop. Debugging an ineffective policy would require providing the LLM with

an extensive context including the IaC, the generated policy, the Terraform plan, and a description of the desired behavior making manual intervention by a human expert a more pragmatic approach. This reinforces the indispensable role of the Human-in-the-Loop (HITL), a core tenet of this thesis, for validating the logical soundness of automated outputs.

7.1.2 Contextual Reasoning: Successes and Shortcomings

The evaluation of the prototype's contextual reasoning, summarized in Table 6.3, reveals a nuanced picture. The system demonstrated a strong ability to understand and reason about relationships between different parts of the IaC, successfully identifying complex, cross-resource, and conditional vulnerabilities. This confirms the hypothesis that LLMs can move beyond the limitations of traditional static analysis by interpreting the broader context in which resources operate.

However, the failures are equally instructive. The prototype struggled with nuances of human intent (e.g., ignoring developer comments) and external context not explicitly present in the code (e.g., recognizing the valid business purpose of a public S3 bucket for a website). These cases underscore the challenges that remain in achieving true contextual understanding and highlight the system's dependency on the data it was trained on. This limitation reinforces the need for a HITL approach, where the automated system provides a strong baseline analysis that a human expert can then refine and validate with their broader, real-world knowledge.

7.2 The Role of the Human-in-the-Loop

While GenAI can automate policy creation, it does not eliminate the need for human expertise. The Human-in-the-Loop (HITL) process is therefore not merely a supplementary feature but a cornerstone of a responsible and effective security automation framework. Its importance is most evident when considering the gap between policy accuracy and effectiveness discussed in Section 7.1.

As shown in Chapter 6, the prototype generated syntactically perfect policies (100% accuracy) but struggled with logical effectiveness, which was as low as 36% for some categories (Table 6.1). A syntactically valid but logically flawed policy can create a false sense of security or, worse, introduce new risks. For example, an overly restrictive policy could cause an outage, while an overly permissive one fails to mitigate the intended threat. The HITL process serves as the critical validation gate to prevent such outcomes. It ensures that a knowledgeable human expert reviews each policy for correctness, relevance, and safety before it is deployed.

Furthermore, the HITL process provides two additional benefits that a fully automated system cannot:

- **Contextual Enrichment:** A human expert can bring in external context that is unavailable to the model, such as business justifications for an unusual configuration (as seen in the "False Positive Reduction" failure in Section 6.5) or knowledge of an impending architectural change. This prevents the system from flagging legitimate configurations as vulnerabilities.
- **System Improvement:** The corrections and approvals from the human reviewer serve as a valuable feedback loop. This data can be used to fine-tune the LLM, refine the RAG knowledge base, and improve the prompt engineering over time, leading to a more intelligent and effective system in the long run.

The workflow is designed to be straightforward: the system presents its recommendation, justification, and source context to a reviewer. The expert can then approve, reject, or modify the policy. This ensures that automation accelerates the process without ceding final control, embodying a partnership between the AI and the human expert.

7.3 Limitations of the Current Study

While the prototype demonstrates the viability of the approach, it is important to acknowledge its limitations. These limitations provide the context for the scope of the findings and form the basis for future work.

7.3.1 Prototype Scope and Generalizability

The prototype was intentionally developed with a narrow focus to serve as a proof-of-concept. Specifically, the evaluation was conducted exclusively on Infrastructure-as-Code written in Terraform for the Amazon Web Services (AWS) cloud platform, using `tfsec` as the primary static analysis linter. This specificity has several implications for the generalizability of the findings.

First, the contextual analysis and policy generation logic are tightly coupled with AWS resource types and IAM semantics. Expanding support to other cloud providers, such as Google Cloud Platform (GCP) or Microsoft Azure, would require further development. This would involve not only extending the knowledge base with provider-specific information but also adapting the contextual analysis prompts and logic to handle different resource models and security paradigms.

Second, the system is dependent on the output of a single linter, `tfsec`. While effective, other popular linters like tfsec or Terrascan identify different sets of issues or provide different contextual details. Integrating these tools would require building specific adapters to normalize their outputs into a format the GenAI core can process, as outlined in Section **??**.

Finally, the policy generation is specific to the Rego language for the Open Policy Agent (OPA). Supporting other policy-as-code engines, such as Sentinel, would necessitate a complete rewrite of the generation prompts and validation logic. Therefore, while the conceptual framework is designed to be provider- and tool-agnostic, the current implementation's results are directly applicable only to the AWS, Terraform, and `tfsec` ecosystem.

7.3.2 GenAI Component Tuning and Optimization

The Generative AI component, built on Amazon Web Services (AWS) Bedrock, is functional but has not been exhaustively optimized. The current implementation uses Anthropic's Claude v2 model and leverages the built-in Knowledge Bases for Bedrock for its Retrieval-Augmented Generation (RAG) capabilities. This setup provides a solid baseline but leaves considerable room for performance and reliability enhancements.

A primary area for improvement is the knowledge base itself. The effectiveness of the RAG system is directly dependent on the quality, relevance, and comprehensiveness of the source documents. The current knowledge base could be expanded with a more diverse corpus of information, including official AWS security bulletins, Terraform best practice guides, and internal security standards. Furthermore, the

configuration of the Bedrock Knowledge Base—specifically its chunking strategy and the underlying embedding model—was left at the default settings for this study. Future work should involve systematically experimenting with different chunking sizes and overlaps to optimize how documents are segmented and retrieved, ensuring that the LLM receives the most relevant and coherent context for each query.

Additionally, while Anthropic's Claude v2 proved capable, AWS Bedrock offers a diverse and evolving landscape of models from various providers. A thorough evaluation of alternative models, including newer versions of Claude or specialized models from other providers, could yield significant improvements in policy generation accuracy, latency, or cost-effectiveness. Such an evaluation would be a critical next step in maturing the prototype from a proof-of-concept to a production-ready tool.

7.4 Future Work and Prototype Expansion

The limitations of this study highlight several promising directions for future research and development.

7.4.1 Knowledge Base Retrieval

The current retrieval-augmented generation (RAG) mechanism provides a foundational level of context, but its precision can be enhanced. Future work should explore more advanced RAG techniques, such as implementing semantic search instead of simple keyword matching, using query transformations to better align user intent with document content, and developing a hybrid search that combines multiple retrieval strategies. Improving the relevance of the contextual information retrieved from the knowledge base is critical to helping the LLM generate more accurate and context-aware security policies.

7.4.2 Model Selection and Tuning

The choice of the Large Language Model is a pivotal factor in the system's performance. The current prototype uses a general-purpose model, but future iterations should involve a systematic evaluation of various LLMs. This includes benchmarking newer, more powerful models (e.g., GPT-4, Claude 3), as well as domain-specific models that are fine-tuned on cybersecurity and IaC data. The evaluation criteria should include not only the accuracy of the generated policies but also performance metrics like latency and throughput, and the overall cost-effectiveness of the solution.

7.4.3 Interaction and Orchestration Logic

The robustness of the interaction between the prototype and the LLM can be substantially improved. The current retry logic for handling transient API failures is basic; a more sophisticated approach, such as implementing exponential backoff, would increase resilience. Furthermore, as IaC configurations grow in complexity, the strategy for chunking content becomes crucial. Future work should move beyond simple fixed-size chunking to code-aware methods that preserve the semantic integrity of the code, ensuring that the LLM receives a coherent and complete context even for large and complex files.

7.4.4 Context Consolidation

The function responsible for consolidating information from multiple sources—such as linter outputs, knowledge base articles, and IaC snippets—is currently rudimentary. A more advanced consolidation function could itself leverage a language model to summarize and synthesize these disparate pieces of information into a single, coherent prompt. This would improve the signal-to-noise ratio in the context provided to the primary LLM, reducing ambiguity and leading to more precise and relevant security policy generation.

7.4.5 Multi-Cloud Support

Expanding the prototype to be multi-cloud capable would be a primary objective. This would involve abstracting the cloud-specific logic into separate modules. For instance, one could create a `CloudProvider` interface with concrete implementations for AWS, Azure, and GCP. The analysis layer would then dynamically load the appropriate module based on the IaC being scanned. This would require changes to the code that interprets linter outputs and maps them to cloud-specific resource configurations and security concepts.

7.4.6 Integration of Additional Linters

Similarly, the prototype can be extended to support more IaC linters (e.g., tfsec, Terrascan). This would require creating a generic `Linter` interface and then implementing specific adapters for each tool. Each adapter would be responsible for executing the linter, parsing its JSON output, and normalizing the findings into a standardized format that the core application can process. This modular design would make the system more versatile and adaptable to different

Chapter 8

Conclusion

This chapter concludes the thesis by summarizing the key findings and contributions. It revisits the research questions and discusses the implications of the results. Finally, it outlines the limitations of the study and suggests potential directions for future research.

8.1 Main Section 1

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam ultricies lacinia euismod. Nam tempus risus in dolor rhoncus in interdum enim tincidunt. Donec vel nunc neque. In condimentum ullamcorper quam non consequat. Fusce sagittis tempor feugiat. Fusce magna erat, molestie eu convallis ut, tempus sed arcu. Quisque molestie, ante a tincidunt ullamcorper, sapien enim dignissim lacus, in semper nibh erat lobortis purus. Integer dapibus ligula ac risus convallis pellentesque.

8.1.1 Subsection 1

Nunc posuere quam at lectus tristique eu ultrices augue venenatis. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aliquam erat volutpat. Vivamus sodales tortor eget quam adipiscing in vulputate ante ullamcorper. Sed eros ante, lacinia et sollicitudin et, aliquam sit amet augue. In hac habitasse platea dictumst.

8.1.2 Subsection 2

Morbi rutrum odio eget arcu adipiscing sodales. Aenean et purus a est pulvinar pellentesque. Cras in elit neque, quis varius elit. Phasellus fringilla, nibh eu tempus venenatis, dolor elit posuere quam, quis adipiscing urna leo nec orci. Sed nec nulla auctor odio aliquet consequat. Ut nec nulla in ante ullamcorper aliquam at sed dolor. Phasellus fermentum magna in augue gravida cursus. Cras sed pretium lorem. Pellentesque eget ornare odio. Proin accumsan, massa viverra cursus pharetra, ipsum nisi lobortis velit, a malesuada dolor lorem eu neque.

8.2 Main Section 2

Sed ullamcorper quam eu nisl interdum at interdum enim egestas. Aliquam placerat justo sed lectus lobortis ut porta nisl porttitor. Vestibulum mi dolor, lacinia molestie gravida at, tempus vitae ligula. Donec eget quam sapien, in viverra eros. Donec pellentesque justo a massa fringilla non vestibulum metus vestibulum. Vestibulum

in orci quis felis tempor lacinia. Vivamus ornare ultrices facilisis. Ut hendrerit volutpat vulputate. Morbi condimentum venenatis augue, id porta ipsum vulputate in. Curabitur luctus tempus justo. Vestibulum risus lectus, adipiscing nec condimentum quis, condimentum nec nisl. Aliquam dictum sagittis velit sed iaculis. Morbi tristique augue sit amet nulla pulvinar id facilisis ligula mollis. Nam elit libero, tincidunt ut aliquam at, molestie in quam. Aenean rhoncus vehicula hendrerit.

Appendix A

Appendix Title Here

Write your Appendix content here.

Bibliography

[1] K. Khanna, "ENHANCING CLOUD SECURITY WITH GENERATIVE AI: EMERGING STRATEGIES AND APPLICATIONS," *JARET*, vol. 3, no. 1, pp. 234–244, Jun. 14, 2024, Number: 1 Publisher: IAEME Publication, ISSN: 2295-5152. Accessed: Apr. 8, 2025. [Online]. Available: https://iaeme.com/Home/article_id/JARET_03_01_021.

[2] D. K. Seth, K. K. Ratra, and A. P. Sundareswaran, "AI and generative AI-driven automation for multi-cloud and hybrid cloud architectures: Enhancing security, performance, and operational efficiency," *2025 IEEE 15th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 00 784–00 793, Jan. 6, 2025, Conference Name: 2025 IEEE 15th Annual Computing and Communication Workshop and Conference (CCWC) ISBN: 9798331507695 Place: Las Vegas, NV, USA Publisher: IEEE. DOI: 10.1109/CCWC62904.2025.10903928. Accessed: Apr. 8, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/10903928/.

[3] A. Patel, P. Pandey, H. Ragothaman, R. Molleti, and D. R. Peddinti, "Generative AI for automated security operations in cloud computing," *2025 IEEE 4th International Conference on AI in Cybersecurity (ICAIC)*, pp. 1–7, Feb. 5, 2025, Conference Name: 2025 IEEE 4th International Conference on AI in Cybersecurity (ICAIC) ISBN: 9798331518882 Place: Houston, TX, USA Publisher: IEEE. DOI: 10.1109/ICAIC63015.2025.10849302. Accessed: Mar. 31, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/10849302/.

[4] E. Tabassi, "Artificial intelligence risk management framework (AI RMF 1.0)," National Institute of Standards and Technology (U.S.), Gaithersburg, MD, NIST AI 100-1, Jan. 26, 2023, NIST AI 100–1. DOI: 10.6028/NIST.AI.100-1. Accessed: Apr. 8, 2025. [Online]. Available: http://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.100-1.pdf.

[5] C. Y. Haryanto, M. H. Vu, T. D. Nguyen, E. Lomempow, Y. Nurliana, and S. Taheri, *SecGenAI: Enhancing security of cloud-based generative AI applications within australian critical technologies of national interest*, Jul. 1, 2024. DOI: 10.48550/arXiv.2407.01110. arXiv: 2407.01110[cs]. Accessed: Aug. 26, 2024. [Online]. Available: http://arxiv.org/abs/2407.01110.

[6] R. Hansen and P. Venables. "Introducing google's secure AI framework," Google, Accessed: Apr. 25, 2025. [Online]. Available: https://blog.google/technology/safety-security/introducing-googles-secure-ai-framework/.

[7] O. Editor. "LLM and generative AI security center of excellence guide," OWASP Top 10 for LLM & Generative AI Security, Accessed: Apr. 27, 2025. [Online]. Available: https://genai.owasp.org/resource/llm-and-generative-ai-security-center-of-excellence-guide/.

[8] O. Editor. “LLM applications cybersecurity and governance checklist v1.1 - english,” OWASP Top 10 for LLM & Generative AI Security, Accessed: Apr. 27, 2025. [Online]. Available: https://genai.owasp.org/resource/llm-applications-cybersecurity-and-governance-checklist-english/.

[9] “ISO/IEC 38500:2024,” ISO, Accessed: Apr. 9, 2025. [Online]. Available: https://www.iso.org/standard/81684.html.

[10] Sushil Prabhu Prabhakaran, “Integration patterns in unified AI and cloud platforms: A systematic review of process automation technologies,” *Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol*, vol. 10, no. 6, pp. 1932–1940, Dec. 15, 2024, ISSN: 2456-3307. DOI: 10.32628/CSEIT241061229. Accessed: Apr. 8, 2025. [Online]. Available: https://ijsrcseit.com/index.php/home/article/view/CSEIT241061229.

[11] “Securing generative AI: Introduction to the generative AI security scoping matrix,” Amazon Web Services, Inc. Accessed: Apr. 9, 2025. [Online]. Available: https://aws.amazon.com/ai/generative-ai/security/scoping-matrix/.

[12] D. Bringhenti, R. Sisto, and F. Valenza, “Security automation for multi-cluster orchestration in kubernetes,” *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*, pp. 480–485, Jun. 19, 2023, Conference Name: 2023 IEEE 9th International Conference on Network Softwarization (NetSoft) ISBN: 9798350399806 Place: Madrid, Spain Publisher: IEEE. DOI: 10.1109/NetSoft57336.2023.10175419. Accessed: Apr. 8, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/10175419/.

[13] K. Hammar and R. Stadler, “Digital twins for security automation,” *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–6, May 8, 2023, Conference Name: NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium ISBN: 9781665477161 Place: Miami, FL, USA Publisher: IEEE. DOI: 10.1109/NOMS56928.2023.10154288. Accessed: Apr. 8, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/10154288/.

[14] Z. Zhou, C. Zhongwen, Z. Tiecheng, and G. Xiaohui, “The study on network intrusion detection system of snort,” May 1, 2010. DOI: 10.1109/ICNDS.2010.5479341.

[15] S. Surathunmanun, W. Ongsakul, and J. G. Singh, “Exploring the role of generative artificial intelligence in the energy sector: A comprehensive literature review,” *2024 International Conference on Sustainable Energy: Energy Transition and Net-Zero Climate Future (ICUE)*, pp. 1–11, Oct. 21, 2024, Conference Name: 2024 International Conference on Sustainable Energy: Energy Transition and Net-Zero Climate Future (ICUE) ISBN: 9798331517076 Place: Pattaya City, Thailand Publisher: IEEE. DOI: 10.1109/ICUE63019.2024.10795598. Accessed: Apr. 8, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/10795598/.

[16] G. Pillala, D. Azarpazhooh, and S. Baxter, “DevSecOps sentinel: GenAI-driven agentic workflows for comprehensive supply chain security,” *Computer and Information Science*, vol. 18, no. 1, p39, Dec. 20, 2024, Number: 1, ISSN: 1913-8989. DOI: 10.5539/cis.v18n1p39. Accessed: Apr. 27, 2025. [Online]. Available: https://ccsenet.org/journal/index.php/cis/article/view/0/51118.

[17] R. L. V. Nyoto, M. Devega, and N. Nyoto, "Cyber security risks in the rapid development of generative artificial intelligence: A systematic literature review," *ComniTech : Journal of Computational Intelligence and Informatics*, vol. 1, no. 2, pp. 57–66, Dec. 29, 2024, ISSN: 3063-0630. Accessed: Apr. 8, 2025. [Online]. Available: https://journal.unilak.ac.id/index.php/ComniTech/article/view/24539.

[18] M. J. Page et al., "The PRISMA 2020 statement: An updated guideline for reporting systematic reviews," *BMJ*, n71, Mar. 29, 2021, ISSN: 1756-1833. DOI: 10.1136/bmj.n71. Accessed: Apr. 12, 2025. [Online]. Available: https://www.bmj.com/lookup/doi/10.1136/bmj.n71.

[19] B. Dash, *Zero-trust architecture (ZTA): Designing an AI-powered cloud security framework for LLMs' black box problems*, Rochester, NY, Mar. 12, 2024. DOI: 10.2139/ssrn.4726625. Accessed: Apr. 27, 2025. [Online]. Available: https://papers.ssrn.com/abstract=4726625.

[20] M. Fu, J. Pasuksmit, and C. Tantithamthavorn, "AI for DevSecOps: A landscape and future opportunities," *ACM Trans. Softw. Eng. Methodol.*, vol. 34, no. 4, pp. 1–61, May 31, 2025, ISSN: 1049-331X, 1557-7392. DOI: 10.1145/3712190. Accessed: Aug. 3, 2025. [Online]. Available: https://dl.acm.org/doi/10.1145/3712190.

[21] D. K. Seth, K. K. Ratra, and A. P. Sundareswaran, "AI and generative AI-driven automation for multi-cloud and hybrid cloud architectures: Enhancing security, performance, and operational efficiency," in *2025 IEEE 15th Annual Computing and Communication Workshop and Conference (CCWC)*, Jan. 2025, pp. 00 784–00 793. DOI: 10.1109/CCWC62904.2025.10903928. Accessed: Aug. 17, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/10903928.

[22] D. Bringhenti, R. Sisto, and F. Valenza, "Security automation for multi-cluster orchestration in kubernetes," in *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*, ISSN: 2693-9789, Jun. 2023, pp. 480–485. DOI: 10.1109/NetSoft57336.2023.10175419. Accessed: Aug. 17, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/10175419.

[23] M. Nicosia and P. O. Kristensson, "6 - risk management in human-in-the-loop AI-assisted attention aware systems," in *Putting AI in the Critical Loop*, P. Dasgupta et al., Eds., Academic Press, Jan. 1, 2024, pp. 81–92, ISBN: 978-0-443-15988-6. DOI: 10.1016/B978-0-443-15988-6.00013-3. Accessed: Aug. 17, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780443159886000133.

[24] "Human-in-the-loop in SOC automation," Accessed: Jun. 26, 2025. [Online]. Available: https://www.xenonstack.com/blog/human-loop-soc-automation.

[25] E. Tabassi, "Artificial intelligence risk management framework (AI RMF 1.0)," National Institute of Standards and Technology (U.S.), Gaithersburg, MD, NIST AI 100-1, Jan. 26, 2023, NIST AI 100–1. DOI: 10.6028/NIST.AI.100-1. Accessed: Aug. 17, 2025. [Online]. Available: http://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.100-1.pdf.

[26] S. Surathunmanun, W. Ongsakul, and J. G. Singh, "Exploring the role of generative artificial intelligence in the energy sector: A comprehensive literature review," in *2024 International Conference on Sustainable Energy: Energy Transition and Net-Zero Climate Future (ICUE)*, Oct. 2024, pp. 1–11. DOI: 10.1109/

ICUE63019.2024.10795598. Accessed: Aug. 17, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/10795598.

[27] C. K. Akiri, K. Jayabalan, J. Lopes, S. A. Kareem, and A. Tabbassum, "Generative AI for real-time cloud security: Advanced anomaly detection using GPT models," in *2025 IEEE Conference on Computer Applications (ICCA)*, Mar. 2025, pp. 1–6. DOI: 10.1109/ICCA65395.2025.11011269. Accessed: Jun. 26, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/11011269.

[28] M. Fakih, R. Dharmaji, H. Bouzidi, G. Q. Araya, O. Ogundare, and M. A. A. Faruque, *LLM4cve: Enabling iterative automated vulnerability repair with large language models*, Jan. 7, 2025. DOI: 10.48550/arXiv.2501.03446. arXiv: 2501.03446[cs]. Accessed: Jun. 26, 2025. [Online]. Available: http://arxiv.org/abs/2501.03446.

[29] L. Alevizos and M. Dekker, "Towards an AI-enhanced cyber threat intelligence processing pipeline," *Electronics*, vol. 13, no. 11, p. 2021, Jan. 2024, Number: 11 Publisher: Multidisciplinary Digital Publishing Institute, ISSN: 2079-9292. DOI: 10.3390/electronics13112021. Accessed: Jun. 26, 2025. [Online]. Available: https://www.mdpi.com/2079-9292/13/11/2021.

[30] P. Gunathilaka, D. Senadheera, S. Perara, C. Gunawardana, S. Thelijjagoda, and J. Krishara, "Context-aware behavior-driven pipeline generation," in *2025 13th International Symposium on Digital Forensics and Security (ISDFS)*, ISSN: 2768-1831, Apr. 2025, pp. 1–6. DOI: 10.1109/ISDFS65363.2025.11011952. Accessed: Jun. 26, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/11011952.

[31] J. Zhang et al., *An empirical study of automated vulnerability localization with large language models*, Mar. 30, 2024. DOI: 10.48550/arXiv.2404.00287. arXiv: 2404.00287[cs]. Accessed: Jun. 26, 2025. [Online]. Available: http://arxiv.org/abs/2404.00287.

[32] "Artificial intelligence for cybersecurity: A state of the art | request PDF," in *ResearchGate*, Mar. 22, 2025. DOI: 10.1109/ICAIC63015.2025.10848980. Accessed: Jun. 26, 2025. [Online]. Available: https://www.researchgate.net/publication/388722398_Artificial_Intelligence_for_Cybersecurity_A_State_of_the_Art.

[33] "Towards transparent intrusion detection: A coherence-based framework in explainable AI integrating large language models," in *ResearchGate*, Jan. 19, 2025. DOI: 10.1109/TPS-ISA62245.2024.00020. Accessed: Jun. 26, 2025. [Online]. Available: https://www.researchgate.net/publication/388090990_Towards_Transparent_Intrusion_Detection_A_Coherence-Based_Framework_in_Explainable_AI_Integrating_Large_Language_Models.

[34] G. Noseevich and D. Gamayunov, *Towards automated web application logic reconstruction for application level security*, Nov. 9, 2015. DOI: 10.48550/arXiv.1511.02564. arXiv: 1511.02564[cs]. Accessed: Jun. 9, 2025. [Online]. Available: http://arxiv.org/abs/1511.02564.

[35] B. Lim, R. Huerta, A. Sotelo, A. Quintela, and P. Kumar, *EXPLICATE: Enhancing phishing detection through explainable AI and LLM-powered interpretability*, Mar. 22, 2025. DOI: 10.48550/arXiv.2503.20796. arXiv: 2503.20796[cs]. Accessed: Jun. 26, 2025. [Online]. Available: http://arxiv.org/abs/2503.20796.

[36] "A testbed for operations in the information environment | request PDF," in *ResearchGate*, Feb. 12, 2025. DOI: 10.1145/3675741.3675751. Accessed: Jun. 26, 2025. [Online]. Available: https://www.researchgate.net/publication/383085784_A_Testbed_for_Operations_in_the_Information_Environment.

[37] "Ground responses using RAG | generative AI on vertex AI," Google Cloud, Accessed: Jun. 26, 2025. [Online]. Available: https://cloud.google.com/vertex-ai/generative-ai/docs/grounding/ground-responses-using-rag.

[38] A. Özgür and Y. Uygun, *A simple architecture for enterprise large language model applications based on role based security and clearance levels using retrieval-augmented generation or mixture of experts*, Jul. 9, 2024. DOI: 10.48550/arXiv.2407.06718. arXiv: 2407.06718[cs]. Accessed: Jun. 26, 2025. [Online]. Available: http://arxiv.org/abs/2407.06718.

[39] "Evaluating static analysis alerts with LLMs," Accessed: Jun. 26, 2025. [Online]. Available: https://insights.sei.cmu.edu/blog/evaluating-static-analysis-alerts-with-llms/.

[40] Z. Li, S. Dutta, and M. Naik, *IRIS: LLM-assisted static analysis for detecting security vulnerabilities*, Apr. 6, 2025. DOI: 10.48550/arXiv.2405.17238. arXiv: 2405.17238[cs]. Accessed: Jun. 18, 2025. [Online]. Available: http://arxiv.org/abs/2405.17238.

[41] A. Haque et al., *SOK: Exploring hallucinations and security risks in AI-assisted software development with insights for LLM deployment*, Jan. 31, 2025. DOI: 10.48550/arXiv.2502.18468. arXiv: 2502.18468[cs]. Accessed: Jun. 26, 2025. [Online]. Available: http://arxiv.org/abs/2502.18468.

[42] P. Gunathilaka, D. Senadheera, S. Perara, C. Gunawardana, S. Thelijjagoda, and J. Krishara, "Context-aware behavior-driven pipeline generation," in *2025 13th International Symposium on Digital Forensics and Security (ISDFS)*, ISSN: 2768-1831, Apr. 2025, pp. 1–6. DOI: 10.1109/ISDFS65363.2025.11011952. Accessed: Aug. 3, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/11011952.

[43] M. Mahboob, M. R. U. Ahmed, Z. Zia, M. S. Ali, and A. K. Ahmed, *Future of artificial intelligence in agile software development*, Aug. 1, 2024. DOI: 10.48550/arXiv.2408.00703. arXiv: 2408.00703[cs]. Accessed: Aug. 3, 2025. [Online]. Available: http://arxiv.org/abs/2408.00703.

[44] S. Peng, E. Kalliamvakou, P. Cihon, and M. Demirer, *The impact of AI on developer productivity: Evidence from GitHub copilot*, Feb. 13, 2023. DOI: 10.48550/arXiv.2302.06590. arXiv: 2302.06590[cs]. Accessed: Aug. 3, 2025. [Online]. Available: http://arxiv.org/abs/2302.06590.

[45] S. Kesireddy, "Copilot in the cloud: Evaluating the accuracy and speed of LLMs in data engineering tasks," *World Journal of Advanced Engineering Technology and Sciences*, vol. 15, no. 3, pp. 1434–1441, 2025, ISSN: 2582-8266. DOI: 10.30574/wjaets.2025.15.3.1049. Accessed: Aug. 3, 2025. [Online]. Available: https://journalwjaets.com/content/copilot-cloud-evaluating-accuracy-and-speed-llms-data-engineering-tasks.

[46] D. Rein et al., *HCAST: Human-calibrated autonomy software tasks*, Mar. 21, 2025. DOI: 10.48550/arXiv.2503.17354. arXiv: 2503.17354[cs]. Accessed: Aug. 3, 2025. [Online]. Available: http://arxiv.org/abs/2503.17354.

[47] M. Tufano, A. Agarwal, J. Jang, R. Z. Moghaddam, and N. Sundaresan, *AutoDev: Automated AI-driven development*, Mar. 13, 2024. DOI: 10.48550/arXiv.2403.08299. arXiv: 2403.08299[cs]. Accessed: Aug. 3, 2025. [Online]. Available: http://arxiv.org/abs/2403.08299.

[48] M. Nicosia and O. Kristensson, "Risk management in human-in-the-loop AI-assisted attention aware systems." Accessed: Jun. 26, 2025. [Online]. Available: https://www.semanticscholar.org/paper/Risk-management-in-human-in-the-loop-AI-assisted-Nicosia-Kristensson/49d3e97575ba5e53fba8d772750a4fa93c7c2cb6

[49] Sarathe Krisshnan Jutoo Vijayaraghavan, "Policy as code: A paradigm shifts in infrastructure security and governance," *World J. Adv. Res. Rev.*, vol. 26, no. 1, pp. 3399–3405, Apr. 30, 2025, ISSN: 25819615. DOI: 10.30574/wjarr.2025.26.1.1441. Accessed: Jun. 26, 2025. [Online]. Available: https://journalwjarr.com/node/1380.

[50] "Streamlining CI/CD pipelines with automated policy checks," Cloudsmith, Accessed: Jun. 26, 2025. [Online]. Available: https://cloudsmith.com/blog/streamlining-ci-cd-pipelines-with-automated-policy-checks.

[51] "Claude code on amazon bedrock," Anthropic, Accessed: Jul. 16, 2025. [Online]. Available: https://docs.anthropic.com/en/docs/claude-code/amazon-bedrock.

[52] "Terraform CLI documentation | terraform | HashiCorp developer," Terraform CLI Documentation | Terraform | HashiCorp Developer, Accessed: Jul. 16, 2025. [Online]. Available: https://developer.hashicorp.com/terraform/cli.

[53] "Introduction | open policy agent," Accessed: Jul. 16, 2025. [Online]. Available: https://openpolicyagent.org/docs.

[54] "What's new in python 3.12," Python documentation, Accessed: Jul. 16, 2025. [Online]. Available: https://docs.python.org/3/whatsnew/3.12.html.

[55] "GitHub actions," GitHub, Accessed: Jul. 16, 2025. [Online]. Available: https://github.com/features/actions.

[56] S. N. Koritala, "Cloud-first strategies for financial data storage and processing," *IJAEM*, vol. 7, no. 2, pp. 789–799, Feb. 2025, ISSN: 23955252. DOI: 10.35629/5252-0702789799. Accessed: Aug. 17, 2025. [Online]. Available: https://ijaem.net/issue_dcp/Cloud%20First%20Strategies%20for%20Financial%20Data%20Storage%20and%20Processing.pdf.

[57] A. R. Patel, R. V. Tiwari, and R. A. Khureshi, "Comparative study of top cloud providers on basis of service availability and cost,"

[58] "European cloud providers' local market share now holds steady at 15% | synergy research group," Accessed: Aug. 17, 2025. [Online]. Available: https://www.srgresearch.com/articles/european-cloud-providers-local-market-share-now-holds-steady-at-15.

[59] R. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*. Upper Saddle River, NJ: Pearson, 2009, 464 pp., ISBN: 978-0-13-235088-4.

[60] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns. Elements of Reusable Object-Oriented Software.* Boston, Mass. Munich: Prentice Hall, 2011, 416 pp., ISBN: 978-0-201-63361-0.

[61] H. Dasari, "Infrastructure as code (IaC) best practices for multi-cloud deployments in enterprises," *International journal of networks and security*, vol. 5, no. 1, pp. 174–186, Jun. 12, 2025, Number: 01, ISSN: 2693-387X. DOI: 10.55640/ijns-05-01-10. Accessed: Jul. 20, 2025. [Online]. Available: https://www.academicpublishers.org/journals/index.php/ijns/article/view/5120.

[62] M. Howard, *Terraform – automating infrastructure as a service*, version: 1, May 21, 2022. DOI: 10.48550/arXiv.2205.10676. arXiv: 2205.10676[cs]. Accessed: Jul. 16, 2025. [Online]. Available: http://arxiv.org/abs/2205.10676.

[63] P. Lewis et al., *Retrieval-augmented generation for knowledge-intensive NLP tasks*, Apr. 12, 2021. DOI: 10.48550/arXiv.2005.11401. arXiv: 2005.11401[cs]. Accessed: Jul. 20, 2025. [Online]. Available: http://arxiv.org/abs/2005.11401.

[64] "AWS well-architected framework - AWS well-architected framework," Accessed: Jul. 20, 2025. [Online]. Available: https://docs.aws.amazon.com/wellarchitected/latest/framework/welcome.html.

[65] "GitOps: Cloud-native continuous deployment: Beetz, florian, kammer, anja, harrer, simon, scheungrab, sonja: 9783982112688: Amazon.com: Books," Accessed: Jul. 20, 2025. [Online]. Available: https://www.amazon.com/GitOps-Cloud-native-Continuous-Florian-Beetz/dp/3982112680.

[66] "Pytest documentation," Accessed: Jul. 20, 2025. [Online]. Available: https://docs.pytest.org/en/stable/.

[67] B. Boyter, *Scc: Sloc, cloc and code*, Publication Title: GitHub repository, 2024. [Online]. Available: https://github.com/boyter/scc.

[68] M. Howard, *Terraform – automating infrastructure as a service*, Issue: arXiv:2205.10676, May 21, 2022. DOI: 10.48550/arXiv.2205.10676. Accessed: Jul. 16, 2025. [Online]. Available: http://arxiv.org/abs/2205.10676.

[69] H. O. Delicheh and T. Mens, "Mitigating security issues in GitHub actions," in *Proceedings of the 2024 ACM/IEEE 4th International Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS) and 2024 IEEE/ACM Second International Workshop on Software Vulnerability*, ser. EnCyCriS/SVM '24, New York, NY, USA: Association for Computing Machinery, Aug. 26, 2024, pp. 6–11, ISBN: 979-8-4007-0565-6. DOI: 10.1145/3643662.3643961. Accessed: Jul. 21, 2025. [Online]. Available: https://doi.org/10.1145/3643662.3643961.

[70] Aqua Security, *Tfsec: Static analysis for terraform code*, Published: Web page. [Online]. Available: https://github.com/aquasecurity/tfsec.

[71] Amplify, *Python-hcl2*, Published: PyPI Repository, 2025. [Online]. Available: https://pypi.org/project/python-hcl2/.

[72] Anthropic, *Claude 2 on amazon bedrock*, Aug. 2023. [Online]. Available: https://www.anthropic.com/news/claude-2-amazon-bedrock.

[73] The OPA Authors, *Open policy agent documentation*, Published: Web page. [Online]. Available: https://www.openpolicyagent.org/docs/latest/.

[74] Textualize, *Rich: A python library for rich text and beautiful formatting in the terminal*, Published: Web page. [Online]. Available: https://github.com/Textualize/rich.

[75] R. Vaidya and G. Sharma, “DevSecOps automation: A comprehensive review of AI integration and future directions,” in *2024 12th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO)*, Jun. 5, 2024. DOI: 10.1109/ICRITO59692.2024.10586919. Accessed: Sep. 8, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/10586919.

[76] Q. Li, Y. Zhang, and W. Chen, “Automated policy generation for cloud infrastructure using large language models,” *IEEE Cloud Computing*, vol. 12, no. 3, pp. 45–54, May 15, 2024, ISSN: 2168-7161. DOI: 10.1109/MCC.2024.3398416. Accessed: Sep. 8, 2025. [Online]. Available: https://www.computer.org/csdl/journal/cc/2024/03/10402123/1I9Z0zE8aBc.

[77] Y. Zheng and Y. Liu, “Context-aware vulnerability detection in infrastructure-as-code,” in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS ’23)*, New York, NY, USA: Association for Computing Machinery, 2023, pp. 1123–1137. DOI: 10.1145/3576915.3576987.