الجمهورية الجزائرية الديمقراطية الشعبية

Republique Algérienne Démocratique Populaire

وزارة التعليم العالي والبحث العلمي

Ministère de l'Enseignement Supérieur de la Recherche Scientifique

Université 20 août 1955 Skikda

جامعة 20 أوت 1955 – سكيكدة

Faculte des Sciences
Departement d'informatique

كلية العلوم
قسم الإعلام الآلي

الرقم: .... / ق/ إ/ أ/ ك.ع2026/

Autorisation de Dépôt de Mémoire de Master

Je soussigné : .Bougamouza....Fatch.....

Certifie que l'étudiant(e) : ...Bounour...Fatima...Zohra.....

Spécialité : .Réseaux.et...Systemes..distribues.....

Ayant soutenu le projet intitulé : ..CodeWatchAI...Based..on..Agentic.AI..
Architecture

A apporté les corrections nécessaires sur son manuscrit de Master.

Signature de l'encadreur



CamScanner

الجمهورية الجزائرية الديمقراطية الشعبية

Republique Algérienne Démocratique Populaire
وزارة التعليم العالي والبحث العلمي

Ministère de l'Enseignement Supérieur de la Recherche Scientifique

Université 20 août 1955 Skikda

جامعة 20 أوت 1955 – سكيكدة

Faculte des Sciences
Departement d'informatique

كلية العلوم
قسم الإعلام الآلي
الرقم: .... / ق/ إ/ أ/ ك.ع2026/

Autorisation de Dépôt de Mémoire de Master

Je soussigné : ...Bougamouza...Fateh.....

Certifie que l'étudiant(e) : ..Boubalbaul...Sara.....

Spécialité : ..Réseaux et Systèmes distribues.....

Ayant soutenu le projet intitulé : CodeWatchAI...Based on Agentic AI.... Architecture

A apporté les corrections nécessaires sur son manuscrit de Master.

Signature de l'encadreur

جامعة 20 أوت 1955 - سكيكدة
قسم الإعلام الآلي
كلية العلوم

الجمهورية الجزائرية الديمقراطية الشعبية

Republique Algérienne Démocratique Populaire

وزارة التعليم العالي والبحث العلمي

Ministère de l'Enseignement Supérieur de la Recherche Scientifique

Université 20 août 1955 Skikda

جامعة 20 أوت 1955 – سكيكدة

Faculte des Sciences
Departement d'informatique

كلية العلوم
قسم الإعلام الآلي

الرقم: .... / ق/ إ/ أ/ ك.ع2026/

Autorisation de Dépôt de Mémoire de Master

Je soussigné : ..Bouguemouza...Fateh.....

Certifie que l'étudiant(e) : ..Chaouch..Malak...Fatima.Zohra.....

Spécialité : .Réseaux.et...Systemes..distribués.....

Ayant soutenu le projet intitulé : ..CodeWatchAI...Based.on.Agentic.AI..Architecture.

A apporté les corrections nécessaires sur son manuscrit de Master.

Signature de l'encadreur

جامعة 20 أوت 1955 - سكيكدة
قسم الإعلام الآلي
كلية العلوم

CamScanner

People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research

University of 20 août 1955 Skikda
Faculty of Sciences
Department of Computer Science



End-of-study thesis for the academic Master's degree in Computer Science

Specialty

Réseaux et Systèmes Distribués

Prepared in accordance with Ministerial Decision No. 1275 of September 27, 2022, concerning the University Degree – Startup.

CodeWatchAI Based on Agentic AI Architecture

Realized by

BOUKALKOUL Sarra
CHAOUCH Malak Fatima Zohra
Bounour Fatima Zohra

Supervised by

Dr. Bougamouza Fateh

Academic Year: 2025 / 2026

Acknowledgements

First and foremost, we express our deepest gratitude to Almighty God for His grace, guidance, and blessings that made the completion of this work possible.

We would like to sincerely thank our supervisor, Dr Bougamouza Fateh, for accepting to supervise this work and for his continuous support, valuable guidance, insightful feedback, and encouragement throughout the development of this thesis. His expertise and dedication have been essential in shaping and improving this project.

We also extend our sincere appreciation to all the professors and administrative staff of the University of August 20, 1955, particularly those involved in the incubator and continuous training programs, for providing a stimulating academic environment and for their constant support during our academic journey.

We are equally grateful to all individuals who contributed directly or indirectly to the realization of this work, even if their names are not explicitly mentioned.

Dedication

This work is the result of perseverance, teamwork, and shared ambition. It is dedicated to all those who have been part of our journey and supported us in one way or another.

We dedicate this thesis first to our families, whose unconditional love, sacrifices, and constant encouragement have been the foundation of our success. Their support has been our greatest source of strength throughout this journey.

To our parents, who believed in us even in moments of doubt, who worked tirelessly to provide us with opportunities, and who never stopped encouraging us to pursue our dreams. This achievement is as much theirs as it is ours.

To our siblings and close relatives, whose presence, motivation, and emotional support have always pushed us forward and reminded us of the importance of perseverance.

To our friends, who shared this journey with us, supported us during stressful times, and contributed to making this experience more meaningful and enjoyable.

To our colleagues within the incubator and startup environment, whose collaboration, exchange of ideas, and teamwork spirit helped us grow both academically and professionally.

Finally, to ourselves ,the three of us, for our resilience, determination, and commitment. We overcame challenges, learned continuously, and stayed united until the completion of this work. This achievement reflects our shared effort, discipline, and belief in our vision.

Contents

Abstract **11**

General Introduction **12**

Chapter 1: Agentic AI **14**

1 Introduction 15
2 History and Evolution of AI 15
2.1 First Generation: Analytical AI 15
2.2 Second Generation: Generative AI 16
2.3 Third Generation: Agentic AI 16
3 Intelligent Agents 16
3.1 Comparison Between Traditional Multi-Agent Systems (SMA) and Modern AI Agents 17
3.2 Components of an Intelligent Agent 17
3.3 Types of Intelligent Agents 18
4 Agentic AI 19
4.1 Definition of Agentic Artificial Intelligence 19
4.2 Core Characteristics of Agentic AI 20
4.3 Comparative Analysis 21
4.4 Illustrative Example 22
4.5 Architecture of Agentic AI Systems 23
4.6 The Spectrum of Agentic Capabilities 24
4.7 Advantages 25
4.8 Application Domains of Agentic AI 25
5 Conclusion 28

Chapter 2: Version Control System **29**

1 Introduction 30
2 Git and GitHub Fundamentals 30
2.1 Git 30
2.1.1 Evolution of Version Control 30
2.1.2 Core Structural Concepts 31

2.1.3 Typical Git Workflow . . . 32
2.2 GitHub . . . 32
2.3 Collaborative Development . . . 33
2.3.1 Pull Requests . . . 33
2.3.2 Webhooks . . . 34
3 Analysis of Existing Code Review Systems . . . 34
3.1 Traditional Static Analysis Tools . . . 34
3.2 AI-Powered Code Review Assistants . . . 37
4 Conclusion . . . 40

Chapter 3: System Design and Development **41**
1 Introduction . . . 42
2 Positioning of the Proposed System . . . 42
3 Proposed System Design . . . 43
3.1 Problem Statement and System Objectives . . . 43
3.2 System Architecture . . . 43
3.3 Deterministic Modules . . . 44
3.4 Intelligent Agents . . . 45
3.5 Workflow Orchestration with n8n . . . 46
3.6 System Workflows . . . 47
4 AI-Powered Code Review Assistants . . . 47
4.1 System Architecture and Workflow . . . 48
4.2 Developer-Entry Point . . . 49
4.3 GitHub-Trigger and Version Control . . . 49
4.4 Workflow Orchestration Layer . . . 49
4.5 Push Module-Data Ingestion . . . 50
4.6 Pull Module-Data Retrieval . . . 50
4.7 Context Module-Contextual Enrichment . . . 51
4.8 The Analysis Tools . . . 51
4.9 Main Agent and Decision Agent . . . 52
4.10 Team Lead-Final Human Decision . . . 53
5 Performance Study . . . 53
6 Conclusion . . . 55

Chapter 4: Implementation **56**
1 Introduction . . . 57
2 Tools and Technologies . . . 57
3 Comparative Analysis: Proposed System vs CodeRabbit . . . 60
4 Conclusion . . . 62

General Conclusion **63**

Chapter 5: Annex **64**

1 First Axis: Project Presentation 65
1.1 Project Idea 65
1.2 Proposed Values 66
1.3 Team Members 66
1.4 Project Objectives 67
1.5 Timeline for Project Implementation 68
2 Second Axis: Innovative Aspects 68
2.1 Nature of the Innovations 68
2.1.1 Radical Technological Innovation 68
2.1.2 Incremental Technological Innovation 69
2.1.3 Radical Market Innovation 69
2.1.4 Incremental Market Innovation 69
2.2 Innovation Domains 69
3 Third Axis: Strategic Market Analysis 70
3.1 Market sector overview 70
3.1.1 PESTEL Analysis 70
3.1.2 Potential Market and Target Market 71
4 Measuring the Level of Competition 73
4.1 Direct and Indirect Competitors 74
4.1.1 Direct Competitors 74
4.1.2 Indirect Competitors 75
4.1.3 Our Competitive Advantage 76
4.2 Marketing strategy 76
4.3 Marketing Mix 76
4.3.1 Product 76
4.3.2 Price 77
4.3.3 Place 77
4.3.4 Promotion 77
4.3.5 People 78
4.3.6 Process 78
4.3.7 Physical Evidence 79
4.4 SWOT Analysis 79
5 Fourth Axis: Production and Organizational Plan 80
5.1 Production process 80
5.2 Supply Chain / Procurement 80
5.3 Workforce 81

5.4 Key Partnerships 82
6 Fifth Axis: Financial Plan 83
6.1 Costs and Expenses 83
6.1.1 Capital Expenditure (CapEx) 83
6.1.2 Operational Expenditure (OpEx) 84
6.1.3 Balance Sheet 85
6.2 Revenue 86
6.3 Projected Income Statement 86
6.4 Cash Flow Plan 87
6.5 Business Model Canvas (BMC) 87
7 Sixth Axis: Prototype 88
7.1 User Onboarding Interfaces 88
7.2 System Interfaces and Dashboard 90
7.3 Subscription Payment and Checkout Workflow 95

Bibliography **95**

Webographie **99**

List of Figures

1 History and Evolution of AI 16
2 SMA vs Agent AI 17
3 The Agent Loop 18
4 Types of Intelligent Agents 19
5 Core Characteristics of Agentic AI 21
6 The Spectrum of Agentic AI 24
7 Agentic AI in Healthcare 26
8 Agentic AI in Finance and Banking 26
9 How Generative AI improve customer service and support ? 27
10 Agentic AI in Supply Chain and Logistics 27

11 Git workflow diagram 32
12 SonarQube analysis pipeline: from source code to quality report 35
13 SonarQube 35
14 ESLint real-time code analysis in VS Code 37
15 Amazon CodeGuru Reviewer findings interface 38
16 CodeRabbit AI-powered review pipeline 38
17 AI Code Reviews | CodeRabbit 38
18 DeepCode (Snyk Code) vulnerability analysis dashboard 39

19 Detailed system architecture 44
20 Deterministic processing pipeline 45
21 Multi-agent collaboration 46
22 AI-Powered Code Review Pipeline 48
23 Developer opens a Pull Request on GitHub 49
24 GitHub as the code host and webhook trigger 49
25 Orchestrating the full automation workflow 50
26 Push module: sequential data ingestion pipeline 50
27 Pull module: sequential PR data retrieval pipeline 51
28 Context Module enriching data before agent analysis 51
29 The sequential execution of Analysis Tools 52
30 Main Agent merging reports and Decision Agent scoring 53
31 Team Lead making the final approval or change request 53

32 Comparison of intelligence index, generation speed, and cost per million output tokens across the three Gemini 2.5 variants. 54

33 General idea of the proposed CodeWatchAI 65
34 Project Implementation Timeline-CodeWatchAI 68
35 PESTEL Analysis of the CodeWatchAI 70
36 SWOT Analysis of the Proposed System 79
37 Capital Expenditure (CapEx) Breakdown of the CodeWatchAI Project . . 83
38 Operational Expenditure (OpEx) Forecast 84
39 Five-Year Forecast Balance Sheet of the CodeWatchAI Startup Project . . 85
40 Projected Revenue Growth for the Period N to N+5 86
41 Projected Income Statement for the Period N to N+5 86
42 Projected Cash Flow Plan for the Period N to N+5 87
43 Business Model Canvas (BMC) of the CodeWatchAI Startup Project . . . 87
44 CodeWatchAI Official Landing Page 88
45 System Documentation Page 88
46 User Sign-In Interface 89
47 Account Creation Form 89
48 Subscription Plans Selection Page 90
49 Main Dashboard Overview 90
50 Dashboard Analytics 91
51 Repository Tracking Interface 91
52 Repositories Management Portal 92
53 Pull Requests Page with Active Notifications Dropdown Menu Overlay . . 92
54 Pull Request Review Page 92
55 Issues List Page 93
56 Decision-Making Interface 93
57 Notifications Page 94
58 System Notifications Centre 94
59 Payment Checkout Portal 95
60 Payment Under Review Page 95

List of Tables

1 List of Abbreviations 10

2 Comparison between Traditional Generative AI and Agentic AI 22

3 Centerlized vs Distributed 31
4 Comparison between SonarQube and ESLint 36
5 Comparison of AI Code Review Tools 40

6 Deterministic Modules Summary 45
7 Intelligent Agents Summary 46
8 Comparison of Gemini 2.5 variants for semantic and security agents 54

10 Comparative Analysis of CodeWatchAI vs CodeRabbit 61

11 Project Team Members 66
12 Project Team Roles 81

Table 1: List of Abbreviations

Abbreviation	**Signification**
AI	Artificial Intelligence
Agentic AI	Agentic Artificial Intelligence
LLM	Large Language Models
IDE	Integrated Development Environment
Call Graph	full function-call dependency graph
Merge Rec.	explicit merge recommendation
Risk Score	weighted severity (Low / Medium / High / Critical)

Abstract

The rapid growth of software development projects and the increasing complexity of modern codebases have made code review an essential activity for ensuring software quality, maintainability, and security. However, traditional manual code review processes remain time-consuming, inconsistent, and highly dependent on the availability and expertise of senior developers. Existing automated solutions often focus on isolated aspects of code analysis and lack a comprehensive understanding of the structural impact of code modifications. This project presents the design and development of CodeWatchAI, an AI-powered automated code review system based on the Agentic AI paradigm. The proposed solution combines multiple intelligent agents specialized in semantic analysis, security assessment, and syntax verification, operating in parallel to provide a comprehensive evaluation of Pull Requests. The system also introduces a Call Graph analysis mechanism to identify dependencies and assess the impact of code changes beyond simple file-level modifications. The proposed architecture relies on GitHub and GitLab webhooks for event triggering, n8n for workflow orchestration, PostgreSQL for data management, and Large Language Models (LLMs) for intelligent code analysis and decision support. The generated reports include detailed findings, risk classification levels, and merge recommendations designed to assist development teams and Team Leads in making informed decisions. The system is further supported by a management dashboard providing real-time visibility into review activity and system performance. The project was developed following a modular and scalable architecture suitable for real-world deployment and future commercialization as a Software-as-a-Service (SaaS) platform. Comparative analysis with existing solutions such as CodeRabbit, GitHub Copilot Code Review, Anthropic Code Review, and Qodo Merge demonstrates the added value of integrating Call Graph analysis, multi-agent collaboration, and decision-oriented reporting within a unified system. The results demonstrate that the proposed approach effectively improves review efficiency, reduces manual workload, strengthens security vulnerability detection, and provides more reliable merge decision support validated through comparative analysis and a functional prototype deployed on a real development workflow. Keywords: Agentic AI, Code Review, Multi-Agent Systems, Large Language Models, Call Graph Analysis, GitHub, GitLab, DevOps, Workflow Automation, Software Quality.

General Introduction

In recent years, software engineering has undergone significant transformation driven by advances in artificial intelligence and the widespread adoption of collaborative software development practices. Modern version control platforms and collaborative review systems have become essential components of software development workflows, enabling developers, startups, students, freelancers, and even non-technical users to build and maintain software efficiently. The emergence of Large Language Models (LLMs) and AI-powered coding assistants has further democratized software development by facilitating code generation and accelerating implementation processes. However, this increased accessibility has introduced new challenges related to code quality, maintainability, security, and reliability. As AI-generated code becomes increasingly prevalent, ensuring the correctness and safety of software modifications before their integration into production environments has become a critical concern. Although existing code review tools provide valuable support, most of them analyze code changes in isolation and lack a comprehensive understanding of the contextual relationships between software components. Furthermore, current solutions rarely combine semantic analysis, syntax validation, vulnerability detection, dependency analysis, and automated merge recommendation within a unified framework, making the review process fragmented and less effective. Motivated by the need for a more intelligent, comprehensive, and automated approach to code review, this project proposes an AI-powered automated code review system based on an Agentic AI architecture. The proposed solution integrates multiple specialized agents, including a Semantic Agent for business logic analysis, a Security Agent for vulnerability detection, a Syntax Analysis Module for coding standards validation, and a Call Graph Analysis component for contextual dependency assessment. The outputs of these components are consolidated by a Main Agent and evaluated by a Decision Agent to generate a weighted risk assessment and an automated merge recommendation. By combining multi-agent intelligence, contextual code understanding, and decision-support mechanisms within a unified event-driven architecture, the proposed system aims to improve software quality assurance while remaining scalable, transparent, and affordable for startups and small-to-medium-sized development teams.

This thesis is organized into four main chapters. Chapter 1 introduces the theoretical

foundations of Artificial Intelligence, intelligent agents, and Agentic AI systems. Chapter 2 presents version control systems, collaborative development practices, and a review of existing code review solutions. Chapter 3 details the analysis and design of the proposed system, including its architecture, intelligent agents, and workflow components. Finally, Chapter 4 describes the implementation process, system integration, testing procedures, and evaluation results.

Chapter 1

Agentic AI

1 Introduction

Artificial Intelligence (AI) has become one of the most influential fields in computer science, significantly transforming the design and functionality of modern technological systems. The rapid growth of computational power, the availability of large datasets, and advances in algorithms have enabled machines to perform tasks that traditionally required human intelligence. These tasks include learning from data, recognizing patterns, understanding natural language, making decisions, and solving complex problems. As a result, AI technologies are increasingly integrated into a wide range of applications, including healthcare, finance, autonomous vehicles, and intelligent software systems.

In the context of modern software engineering, AI is no longer confined to theoretical research but has evolved into a practical approach for developing intelligent and adaptive systems. By enabling systems to learn from experience and respond dynamically to changing conditions, AI contributes to the creation of more efficient and autonomous solutions. These capabilities are particularly relevant to emerging paradigms such as agentic AI systems, which rely on autonomous decision-making and adaptive behavior. Therefore, understanding the fundamental concepts, historical development, and significance of AI is essential for studying advanced intelligent systems.

This chapter traces the evolution of AI across its three generations and introduces the concept of intelligent agents, their components, and their types. It then focuses on Agentic AI, examining its architecture, core characteristics, and application domains, before concluding with an illustrative example.

2 History and Evolution of AI

The development of Artificial Intelligence has evolved through several distinct phases since its emergence as a scientific discipline in the mid-twentieth century. The formal birth of AI is often associated with the Dartmouth Conference in 1956 , where researchers proposed that machines could simulate aspects of human intelligence [1].

2.1 First Generation: Analytical AI

Early AI systems focused on pattern recognition, classification, and prediction based on training data. These systems could identify objects in images, detect fraud in transactions, or recommend products based on past behavior. However, they operated within narrow domains and required human intervention to apply their outputs to real-world tasks.

During the 1960s and 1970s, symbolic AI led to the creation of expert systems , which used predefined rules to replicate the decision-making processes of human experts. These systems were applied in domains such as medical diagnosis and industrial troubleshooting.

However, limitations in computational power and the rigidity of rule-based approaches led to periods known as AI winters, during which research progress slowed significantly [2,3,4].

2.2 Second Generation: Generative AI

In the 1990s and early 2000s, the field experienced renewed growth with the rise of machine learning, a paradigm that enables systems to learn patterns directly from data rather than relying solely on manually defined rules. The advent of large language models and diffusion models gave rise to generative AI systems capable of creating new content including text, images, code, and audio. Models like GPT and DALL-E demonstrated remarkable fluency but remained fundamentally reactive. They waited for user prompts and responded with single-turn outputs, lacking the ability to pursue extended goals or take sequential actions [5,6].

2.3 Third Generation: Agentic AI

The most recent phase in AI development is characterized by agentic AI, which builds upon generative capabilities while adding the critical dimension of autonomous goal pursuit. These systems can receive high-level objectives, break them down into actionable steps, execute those steps using external tools, evaluate outcomes, and adjust their approach based on feedback. This transforms AI from a passive tool into an active collaborator [7].

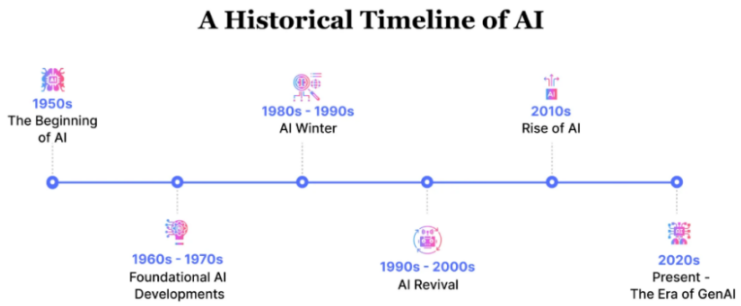


Figure 1: History and Evolution of AI [Web1]

3 Intelligent Agents

Intelligent agents are a key concept in artificial intelligence, representing systems that can perceive their environment, make decisions, and act autonomously to achieve specific

goals. This section introduces the comparison Between Traditional Multi-Agent Systems (SMA) and Modern AI Agents, components, and types of intelligent agents [2].

3.1 Comparison Between Traditional Multi-Agent Systems (SMA) and Modern AI Agents

The following table summarizes the key differences between traditional Multi-Agent Systems (SMA) and modern AI Agents across several fundamental criteria.

Criteria	Old Agent — SMA	Modern Agent — Agent AI
Decision making	Predefined rules and fixed decision trees	Dynamic reasoning, autonomous and context-aware
Adaptability	Limited to scenarios anticipated by designer	Adapts to new, unforeseen situations in real time
Learning ability	No self-learning; requires manual updates	Continuous self-improvement via machine learning
Communication	Rigid, structured protocols between agents	Natural language and multimodal inputs
Human intervention	Constant human intervention needed	Operates autonomously with minimal supervision
Data handling	Structured and predefined data only	Structured, unstructured, and multimodal data
Intelligence type	Rule-based, reactive intelligence	Adaptive, proactive, and generative intelligence
Application scope	Narrow, domain-specific tasks	Wide range of complex, cross-domain applications

Figure 2: SMA vs Agent AI [AIGC 1]

3.2 Components of an Intelligent Agent

The architecture of an intelligent agent typically comprises three interrelated components:

- **Perception**: involves acquiring information about the environment through sensors or input interfaces. For instance, a robotic vacuum cleaner uses proximity sensors and cameras to detect obstacles and map the layout of a room.
- **Decision-making**: refers to the cognitive process through which the agent selects actions that align with its objectives. This process may involve reasoning, planning, or predictive modeling to evaluate potential outcomes. In recommendation systems, for example, an agent analyzes user behavior and content preferences to suggest movies or products that maximize user satisfaction.

- **Action**: constitutes the execution of selected behaviors to effect changes in the environment. In AI-driven applications, actions can range from controlling physical actuators in robots to generating digital outputs in software systems [2].

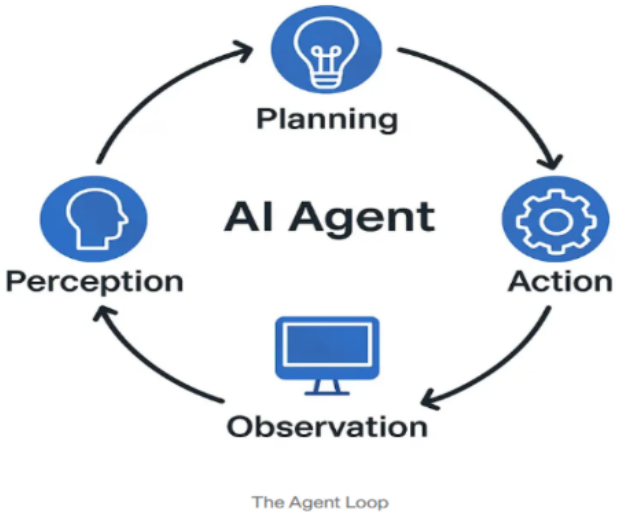


Figure 3: The Agent Loop[Web3]

3.3 Types of Intelligent Agents [2,7]

Intelligent agents are categorized by how they think, decide, and learn. Here's a breakdown in plain terms:

- Simple Reflex Agents

Are the most straightforward type. They follow a fixed set of "if this, then that" rules, no thinking ahead, no memory. A classic example is a spam filter that automatically sends an email to junk because it contains the word "free money." Quick and simple, but easily fooled.

- Model-Based Agents

Are a step smarter. Instead of just reacting, they keep a mental map of their surroundings, even when they can't directly observe everything. Think of a self-driving robot exploring a new building, it builds an internal map as it goes and uses it to make better decisions.

- Goal-Based Agents

have a target in mind and work backwards from it. They ask "what sequence of actions will get me to my goal?" rather than just reacting to the moment. A delivery route planner that calculates the most efficient path to multiple destinations is a good example.

- Learning Agents

Learning Agents are the most advanced. They don't just follow rules or chase goals, they get better over time by learning from experience. If something works, they do more of it; if it fails, they adjust. AI players in video games and Netflix's recommendation engine are real-world examples of this [8].

Each type builds on the last, adding more intelligence and flexibility along the way.

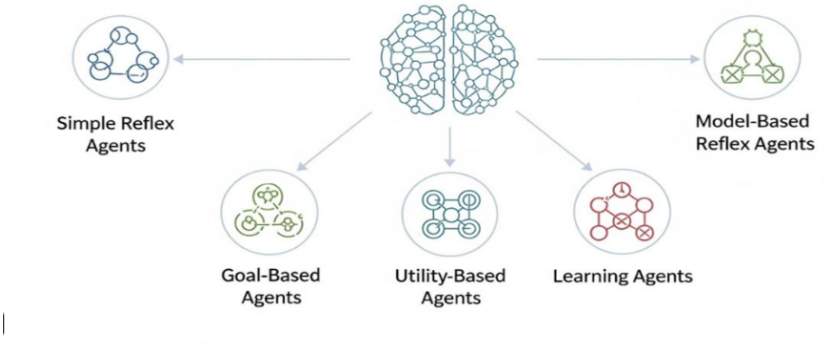


Figure 4: Types of Intelligent Agents[Web4]

4 Agentic AI

Agentic AI marks a new era where machines no longer wait for instructions they plan, decide, and act on their own to complete complex goals.

4.1 Definition of Agentic Artificial Intelligence

Agentic Artificial Intelligence (Agentic AI) refers to AI systems that go beyond simply answering questions or generating content. Instead, they act as autonomous digital workers capable of perceiving their environment, setting sub-goals, planning a sequence of steps, and executing actions to achieve a given objective with little to no human guidance.

The word "agentic" comes from "agent" an entity that acts upon its world. In AI, this means building systems that do not just process information passively, but actively pursue goals over time. This is a major shift from earlier AI models that were limited to pattern recognition, text generation, or classification tasks with no continuity or long-term purpose.

What makes Agentic AI possible today is the combination of four key advancements: (1) large language models with powerful reasoning abilities, (2) access to external tools and APIs, (3) memory systems that retain context across steps, and (4) planning frameworks that allow the agent to break a big goal into smaller, executable actions and adjust when something goes wrong.

4.2 Core Characteristics of Agentic AI

Agentic AI systems are distinguished by several interconnected characteristics that enable autonomous operation .

- Autonomy

Autonomy refers to the ability of an agent to operate independently toward a given goal, making decisions without step-by-step human guidance. It exists on a spectrum, ranging from agents that require approval for certain actions to those that operate with complete independence. In software development, this capability proves particularly valuable autonomous agents can analyze requirements, generate code, and debug errors with minimal human involvement [7,9].

A simple example is GitHub Copilot, which automatically suggests and completes code as a developer types, without waiting for manual instructions.

- Reasoning

Reasoning is the ability of an agentic AI to analyze information, draw conclusions, and solve problems in a goal-directed way. It goes beyond simple pattern recognition, enabling agents to make informed decisions, anticipate consequences, and adjust behavior accordingly.

A practical example is an AI diagnostic tool that evaluates a patient's symptoms, medical history, and test results together to suggest the most likely diagnosis, rather than matching a single symptom to a fixed answer [10,7,11].

-Planning

Planning refers to the ability of an agentic AI to organize and sequence actions over time in order to reach a specific goal. Unlike reactive AI that only responds to immediate inputs, a planning agent thinks ahead, coordinates multiple steps, and handles task dependencies.

A simple example is an AI travel assistant that books your flight, reserves a hotel, and schedules airport transport, all in the right order, automatically [7].

- Memory and Context Awareness

Memory is essential for coherent agent behavior over time. Agentic AI typically incorporates multiple memory systems:

- **Working memory**: Holds current context, active goals, and immediate plans

- **Episodic memory**: Stores records of past experiences, including what happened, what actions were taken, and what outcomes resulted
- **Semantic memory**: Maintains factual knowledge and learned patterns
- **Procedural memory**: Encodes knowledge about how to perform specific tasks or use specific tools These memory systems enable agents to learn from experience, maintain consistency across interactions, and build increasingly sophisticated understanding of their domains [7].



Figure 5: Core Characteristics of Agentic AI[Web5]

4.3 Comparative Analysis (Traditional AI vs Agentic AI):

The distinction between traditional generative AI and agentic AI becomes clear when examining their core characteristics side by side:

Feature	Traditional Generative AI	Agentic AI
Interaction Model	Reactive: waits for user prompts	Proactive: initiates actions to achieve goals
Scope of Task	Single-step responses	Multi-step workflows with dependent interactions
Memory & Context	Limited to the current session	Maintains long-term memory across interactions
Tool Use	Mostly relies on internal knowledge	Interacts with APIs, databases, and external software
Decision-Making	Single decision per prompt	Sequential decisions with continuous feedback
Adaptability	Static outputs	Dynamically adjusts strategies based on results
Goal Orientation	Responds to specific requests	Pursues broader objectives autonomously
Planning Capability	No planning beyond response generation	Planning and task decomposition
Learning Over Time	Minimal or no learning	Continuous improvement through experience

Table 2: Comparison between Traditional Generative AI and Agentic AI

4.4 Illustrative Example

The contrast between traditional and agentic AI becomes clear through concrete examples: **Traditional AI:** "Generate a Python function to sort a list." The system produces a single response a sorting function and the interaction ends.

Agentic AI: "Monitor the codebase for performance issues, optimize sorting algorithms where needed, and submit updates automatically for review." The agentic system would:

- Analyze the codebase to identify sorting operations
- Benchmark current implementations for performance
- Research optimal algorithms for specific data characteristics
- Implement improved versions while maintaining compatibility

- Test changes to ensure correctness
- Generate pull requests with documentation
- Monitor feedback and adjust future strategies This shift from simple prompt- response to autonomous problem-solving represents a significant advancement [7,9].

4.5 Architecture of Agentic AI Systems[7]

The architecture of agentic AI systems defines how components interact to enable autonomous behavior through modular design and continuous feedback.

- Main Components

The architecture of an agentic AI system typically comprises several key components that work together to enable autonomous operation:

- **The Perception Module:** The perception module processes inputs from the environment, translating raw data into representations the agent can use. This includes parsing natural language, interpreting API responses, and processing multimodal inputs.
- **The Reasoning Engine:** The reasoning engine handles moment-to-moment cognitive work: evaluating situations, drawing inferences, retrieving relevant memories, assessing progress toward goals, and recognizing when plans need adjustment.
- **The Planning Module:** The planning module decomposes high-level goals into executable steps using hierarchical planning, sequential planning, contingency planning, and resource allocation.
- **The Memory System:** The memory system manages information across timescales: short-term buffer for immediate context, episodic memory for past experiences (often using vector databases), semantic memory for factual knowledge, and working memory for active goals.

- Functioning of an Autonomous Agent

The interaction between agentic AI systems and their environment is governed by the perception action loop, a cyclical process in which the agent senses environmental states, processes information, and executes actions that influence the environment. Sensors, APIs, and data inputs allow agents to gather relevant information, which is subsequently analyzed through reasoning and planning modules to determine optimal actions .

Feedback mechanisms are critical for adaptation in dynamic environments. Agents continuously evaluate the outcomes of their actions, adjusting future behavior to improve performance. In robotic applications, for example, autonomous mobile robots use feedback from lidar or vision sensors to correct navigation paths in real-time. This iterative perception-action process ensures that agentic AI systems remain responsive and adaptive in complex, evolving contexts.

4.6 The Spectrum of Agentic Capabilities

Agentic AI exists on a spectrum of increasing autonomy :

- **Level 0: Reactive Systems** Respond to inputs without memory or planning. Traditional chatbots operate at this level.
- **Level 1: Scripted Agents** Follow predefined workflows with limited branching. Basic customer service bots with decision trees.
- **Level 2: Prompt-Reactive Agents** Interpret natural language and take simple actions but don't independently pursue goals. Current function-calling models.
- **Level 3: Goal-Oriented Agents** Receive high-level objectives and create plans to achieve them. Decompose goals, sequence actions, and adapt to obstacles. Minimum threshold for true agentic AI .
- **Level 4: Learning Agents** Improve performance over time through experience. Remember past successes and failures, refine strategies, and build domain models.
- **Level 5: Reflective Agents** Engage in metacognition evaluating their own reasoning, identifying biases, and deliberately improving capabilities. The frontier of current research.

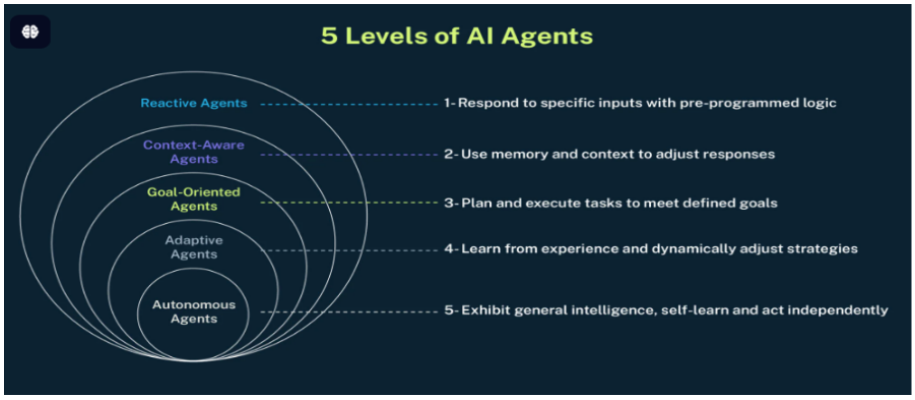


Figure 6: The Spectrum of Agentic AI [AIGC 1]

Most production systems currently operate at Levels 23, with research demonstrating Level 4 capabilities and early explorations of Level 5 .

4.7 Advantages

Multiple agents working together offer several advantages:

- **Specialization**: Different agents can specialize in different capabilities research, planning, execution, quality control. Each is optimized for its role rather than requiring one agent to handle all functions.
- **Parallel Processing**: Multiple agents work simultaneously on different aspects of a problem, dramatically reducing time for complex tasks.
- **Debate and Refinement**: Agents challenge each other's assumptions, identify flaws in reasoning, and collectively arrive at better solutions than any individual could produce.
- **Orchestration and Coordination**: Coordination patterns include hierarchical control (manager assigns tasks), peer-to-peer collaboration, market-based bidding, and blackboard systems.

Multi-agent systems are particularly valuable for complex, open-ended problems where no single approach is guaranteed to succeed.

4.8 Application Domains of Agentic AI

Artificial Intelligence has become essential in modern systems, enabling automation, smarter decision-making, and efficient data processing. Its growing importance has driven its adoption across a wide range of application domains.

- Healthcare

Agentic AI enhances patient care through clinical decision support, diagnostic assistance, personalized treatment planning, chronic disease management, and accelerated drug discovery [9].



Figure 7: Agentic AI in Healthcare[Web7]

- Finance and Banking

In finance, agentic AI enables real-time fraud detection, automated trading, rapid risk assessment, personalized financial advisory, and regulatory compliance [9].

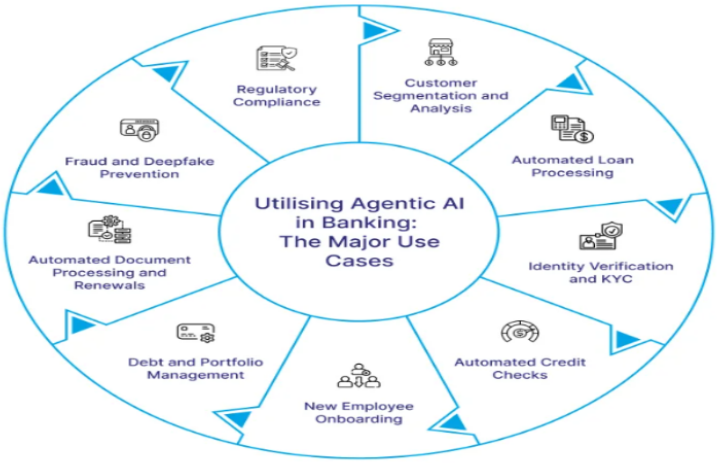


Figure 8: Agentic AI in Finance and Banking[Web8]

- Customer Service and Business Operations [9]

Agentic AI improves business efficiency through intelligent customer support, automated sales engagement, HR recruitment automation, and administrative workflow management.

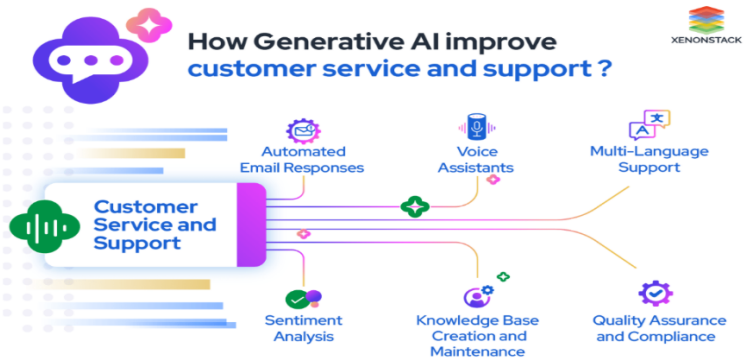


Figure 9: How Generative AI improve customer service and support ? [web9]

- Supply Chain and Logistics [9]

Agents optimize supply chains via demand forecasting, dynamic route planning, supplier performance monitoring, and warehouse operations management.

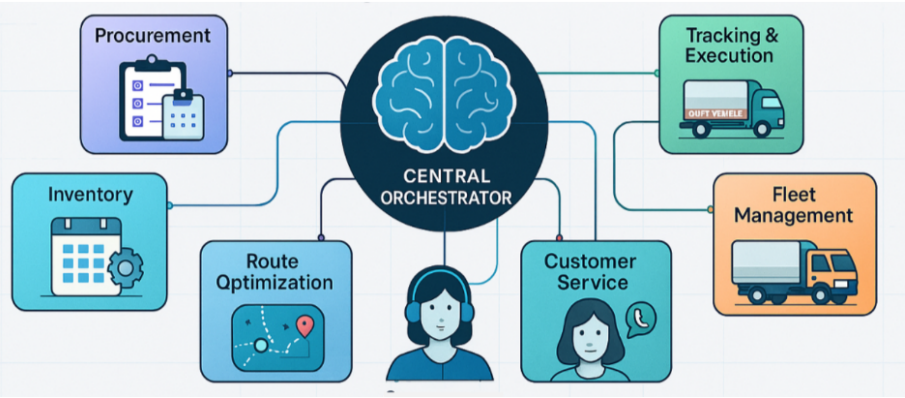


Figure 10: Agentic AI in Supply Chain and Logistics[Web10]

- Cybersecurity [9, web6]

Agentic AI strengthens security through continuous threat detection, automated incident response, vulnerability management, compliance monitoring, and security awareness training.

- Business Intelligence and Analytics [9]

Agents automate data analysis, report generation, predictive modeling, and competitive intelligence to support faster and more informed decision-making.

- Software Development and [web 21]

Tools like GitHub Copilot assist in code generation, while autonomous agents manage CI/CD pipelines, automated testing, and deployment through platforms like AutoGPT

and CrewAI.

- Autonomous Robotics

Agentic robots perceive their environment and execute complex tasks autonomously, using integrated planning for real-time navigation and object manipulation.

- Intelligent Personal Assistants

Assistants like Siri, Google Assistant, and Microsoft Copilot leverage agentic AI to manage schedules, prioritize tasks, and adapt proactively to user behavior.

- Git and GitHub in AI Development

GitHub supports agentic AI development through version control, collaborative workflows, and CI/CD automation, enabling frameworks like LangChain and AutoGPT to distribute updates and coordinate contributions efficiently.

5 Conclusion

This chapter traced the evolution of AI through three generations, from rule-based Analytical AI, through content-generating Generative AI, to today's Agentic AI systems that autonomously plan, reason, and act toward complex goals. We examined the architecture and types of intelligent agents, then explored the four core characteristics of Agentic AI, autonomy, reasoning, planning, and memory, that transform AI from a passive tool into an active collaborator.

Building on this foundation, Chapter 2 shifts focus to version control as a critical pillar of modern software engineering, covering Git and GitHub fundamentals, collaborative development practices such as pull requests and webhooks, and a comparative analysis of code review systems, from traditional static tools like SonarQube and ESLint to AI-powered assistants including CodeGuru, CodeRabbit, DeepCode, and Claude.

Chapter 2

Version Control System

1 Introduction

The previous chapter introduced the foundational concepts of Artificial Intelligence, tracing its evolution from analytical and generative approaches to the emerging paradigm of agentic AI, and examined the architecture, characteristics, and capabilities of intelligent agents.

This chapter shifts focus to the technical environment in which the proposed system operates. It begins by introducing Git and GitHub, covering version control fundamentals and the collaborative mechanisms, Pull Requests and Webhooks that form the integration backbone of the proposed system. It then analyzes existing code review solutions, from traditional static analysis tools such as SonarQube and ESLint to AI-powered assistants such as CodeGuru and CodeRabbit, identifying the limitations that motivate our contribution.

2 Git and GitHub Fundamentals

The Working Mechanism of Git and GitHub :

2.1 Git [12]

Git is a distributed version control system whose initial release dates back to 2005. It was conceived to address the practical shortcomings observed in earlier centralised tools, particularly the bottlenecks that emerge when an entire team depends on a single shared server to access and modify project files. The fundamental design decision that distinguishes Git from its predecessors is the choice to record complete snapshots of the project at each saved state rather than storing incremental differences between versions. This approach has far-reaching consequences for performance, offline capability, and data reliability.

To situate Git within its broader context, it belongs to the category of Version Control Systems (VCS); tools whose purpose is to record changes to a set of files over time and make it possible to retrieve earlier states of the work when necessary. In large software projects where numerous contributors may be modifying overlapping sections of the codebase simultaneously, such a system is not an optional convenience but a practical necessity .

2.1.1 Evolution of Version Control

As summarized in Table 3, the shift from centralized to distributed version control systems brought significant improvements in terms of speed, availability, and data safety.

Aspect	Centralized VCS (SVN)	Distributed VCS (Git)
Repository Location	Single authoritative server	Full replica on every developer's machine
Network Dependency	Required for most operations	Most operations work offline
Operation Speed	Slower due to remote calls and latency	Fast almost instantaneous local operations
Project History Access	Available only on the central server	Complete history available on every clone
Single Point of Failure	Yes server failure halts all work	No history survives even if one machine is lost
Data Safety	Lower depends entirely on the server	Higher redundant copies across all clones

Table 3: Centerlized vs Distributed

2.1.2 Core Structural Concepts

ăă Familiarity with a small number of foundational concepts is necessary before Git's workflow can be understood clearly:

- **Repository(Repo)**: a directory that contains all project files together with the complete recorded history of every change made to them.
- **Working Directory**: the portion of the file system where a developer actively edits files; changes made here are not yet part of any recorded snapshot.
- **Staging Area(Index)**:an intermediate layer that sits between the working directory and the repository, allowing developers to select which changes will be included in the next commit.
- **Commit**: a permanent snapshot of the staged changes, saved to the repository with a descriptive message that explains the intent of the modification.
- **Branch**: an independent line of development that diverges from the main history, allowing features or bug fixes to be developed in isolation before being reintegrated.
- **Merge**: the operation that combines the history of two branches into a single unified line.

2.1.3 Typical Git Workflow

A representative workflow proceeds through the following steps:

clone the remote repository to obtain a local copy, modify the relevant files in the working directory, stage the desired changes, record a snapshot with a descriptive message, and publish the local commits to a shared remote repository.

As illustrated in Figure 11, this workflow maps onto the local and remote repository structure: git add moves changes to the staging area, git commit saves them to the local repository, and git push/git pull synchronize with the remote repository, while git checkout and git merge handle branch switching and merging.

Each of these operations is atomic and reversible. Among Git's other defining characteristics are efficient handling of large codebases, lightweight branch management, comprehensive history tracking with rollback support, offline operability, and strong data integrity enforced through cryptographic hashing of every stored object [13].

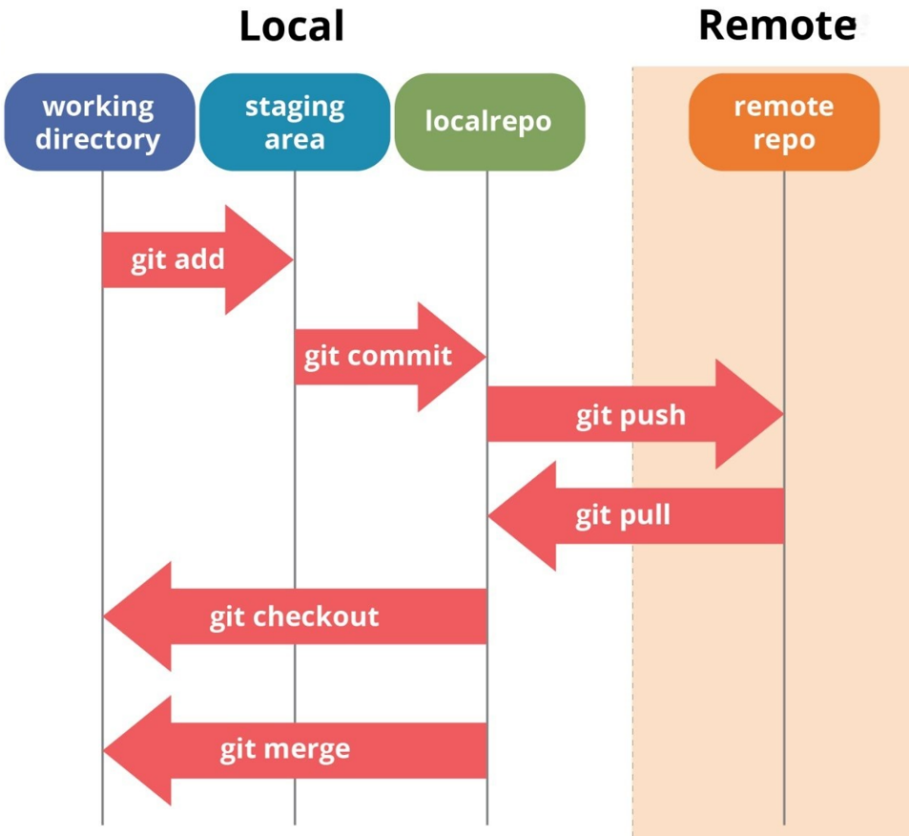


Figure 11: Git workflow diagram [Web11]

2.2 GitHub

GitHub is a cloud-hosted platform that provides remote storage for Git repositories while layering on a comprehensive suite of collaboration, project management, and automation capabilities. It was launched in 2008 and has since grown into the most widely used

platform of its kind. The distinction between Git and GitHub is fundamental: Git is a version control tool that runs entirely on a developer's local machine; GitHub is a web service that stores Git repositories remotely and makes them accessible to teams distributed across different locations. Git manages the history of a codebase; GitHub manages the collaboration around that codebase.

GitHub's feature set addresses both individual and team needs across several dimensions. Repositories serve as the central units of storage. Pull Requests provide the primary mechanism through which proposed changes are communicated, reviewed, and eventually merged. The Issues system offers lightweight tracking of bugs, tasks, and feature requests. GitHub Actions enables teams to define automated workflows, such as running a test suite or deploying an application, triggered by repository events. The web interface allows most common operations to be performed through a browser without requiring a command-line client[web12].

Connecting a local project to a GitHub repository involves two commands. The first registers the remote address; the second publishes the initial commit:

```
git remote add origin https://github.com/user/repo.git
git push -u origin main
```

From that point onward, the local and remote repositories remain synchronised, and any collaborator with appropriate access can clone, pull, or contribute changes through the established workflow[web12].

2.3 Collaborative Development

The capacity to support structured collaboration at scale is arguably GitHub's most consequential contribution to modern software development. It provides an environment in which multiple contributors can work simultaneously on the same codebase without interfering with each other's progress, because each developer operates on an isolated branch and changes are integrated into the shared baseline only after they have been reviewed and validated [14].

2.3.1 Pull Requests

A pull request is a formal proposal to incorporate the changes on one branch into another. Opening a PR initiates a structured conversation around the proposed changes: team members can leave inline comments on specific lines of code, suggest alternative approaches, and mark items as resolved once addressed. Automated checks, test suites, linting tools, or static analysis pipelines, can be configured to run automatically when a PR is opened, providing immediate feedback before any human reviewer becomes in-

volved. This combination of machine verification and peer review is what makes the PR workflow central to modern software quality assurance practices [14].

2.3.2 Webhooks

A webhook is a lightweight HTTP callback that GitHub dispatches to an external endpoint whenever a specified event occurs in a repository. The triggering event could be a new commit being pushed, a pull request being opened or updated, a review being submitted, or any of dozens of other repository activities. The receiving service can then perform whatever action it is configured to take, starting a build, running a test suite, sending a notification, or triggering a deployment to a staging environment .

Webhooks are the mechanism that integrates GitHub into broader Continuous Integration and Continuous Deployment (CI/CD) pipelines. They replace the inefficient polling pattern with an event-driven architecture in which the external system simply waits to be notified, allowing pipelines to react to repository events in near real time[15].

3 Analysis of Existing Code Review Systems

Before presenting the proposed system, it is important to examine the tools and approaches that currently exist for automated code quality analysis. Understanding their capabilities, limitations, and design philosophies provides a solid basis for justifying the need for a new, more intelligent solution. This section reviews two main categories of existing systems: traditional static analysis tools, which rely on predefined rules and pattern matching, and AI-powered code review assistants, which leverage machine learning and large language models to provide more context-aware and intelligent feedback.

3.1 Traditional Static Analysis Tools

Traditional static analysis tools represent one of the earliest and most widely adopted approaches to automated code quality assessment. These tools analyze source code without executing it, relying on predefined rules and pattern-matching techniques to detect issues such as syntax errors, coding standard violations, code duplication, and potential defects. Their deterministic nature ensures consistent and reproducible results, which makes them well-suited for integration into continuous integration and continuous deployment (CI/CD) pipelines [15].

Among the most widely used tools in this category are SonarQube and ESLint, which address different aspects of code quality in modern software development environments.

- **SonarQube:** SonarQube is an open-source platform designed for continuous inspection of code quality across more than 30 programming languages. It performs

static analysis by transforming source code into intermediate representations, such as Abstract Syntax Trees (ASTs) and Control Flow Graphs (CFGs), which are then evaluated using a comprehensive set of predefined rules. These rules are typically grouped into three main categories: reliability issues (bugs), security vulnerabilities, and maintainability concerns (code smells) [16,17].

Each detected issue is assigned a severity level, allowing development teams to prioritize corrective actions effectively. SonarQube integrates seamlessly into CI/CD pipelines and supports the concept of Quality Gates, which define conditions that must be satisfied before code can be accepted into the main branch or deployed to production. In addition, the platform provides dashboards that enable long-term monitoring of code quality metrics across projects, making it particularly suitable for large-scale and enterprise environments.

Despite its widespread adoption, research has shown that static analysis tools such as SonarQube may produce a significant number of false positives and are limited in their ability to capture issues that require a deeper understanding of program semantics or architectural context [18].

Source Code	→	AST / CFG Parser	→	Rule Engine	→	Quality Report / hboard Das
Developer writes code		Code structure extracted		Rules applied (Bugs, Smells, CVEs)		Findings shown in CI/CD & dashboard

Figure 12: SonarQube analysis pipeline: from source code to quality report

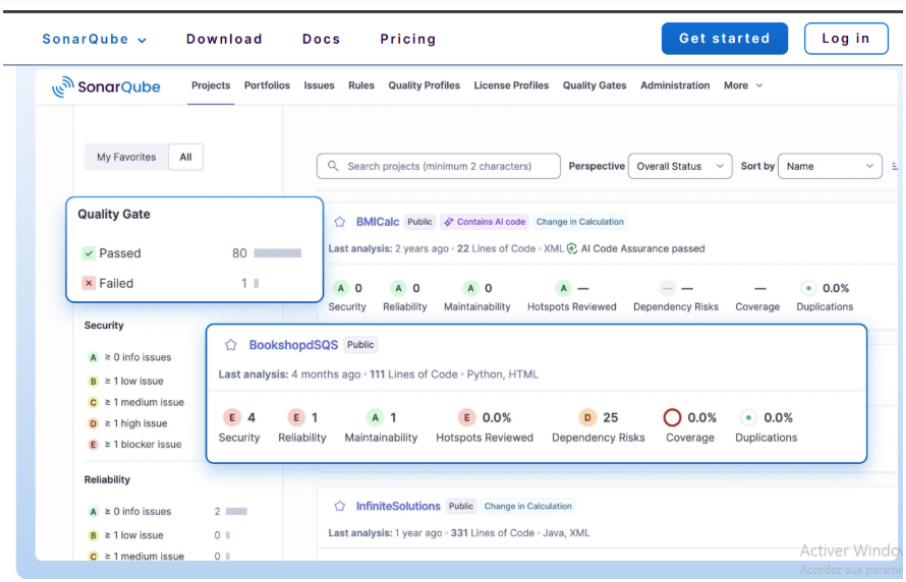


Figure 13: SonarQube[Web14]

- **ESLint** ESLint is a widely used, open-source linting tool primarily designed for One of the key strengths of ESLint is its tight integration with modern development

environments. It provides real-time feedback directly within code editors, enabling developers to detect and correct issues as they write code. This early feedback loop helps improve code consistency and reduces the likelihood of introducing errors before code is committed to the repository.

ESLint also supports customization of coding standards through user-defined rules or widely adopted community configurations, allowing teams to enforce consistent practices across distributed development environments. However, like other traditional static analysis tools, ESLint primarily focuses on syntactic and stylistic aspects of code and does not provide deeper semantic analysis or understanding of program behavior [19].

Criterion	**SonarQube**	**ESLint**
Primary Scope	Full project multi-language	File-level JavaScript/TypeScript focused
Analysis Method	AST + Control Flow Graph + rule engine	AST + pluggable rule engine
Issue Categories	Bugs, vulnerabilities, code smells	Syntax errors, style violations
CI/CD Integration	Yes Quality Gates	Yes via npm scripts or hooks
Semantic Understanding	Limited rule-based only	None syntactic only
False Positive Rate	High (82% in studies [18])	Moderate depends on configuration
Business Logic Awareness	None	None

Table 4: Comparison between SonarQube and ESLint

While both tools are valuable components of a modern development workflow, their shared limitation is clear: they analyze what the code looks like, not what it does. They cannot evaluate whether a function correctly implements the intended business logic, whether a change breaks a subtle contract between modules, or whether a newly introduced pattern creates a security risk in a specific architectural context. These are precisely the kinds of issues that require either a senior human reviewer;

or a more intelligent, AI-powered system.

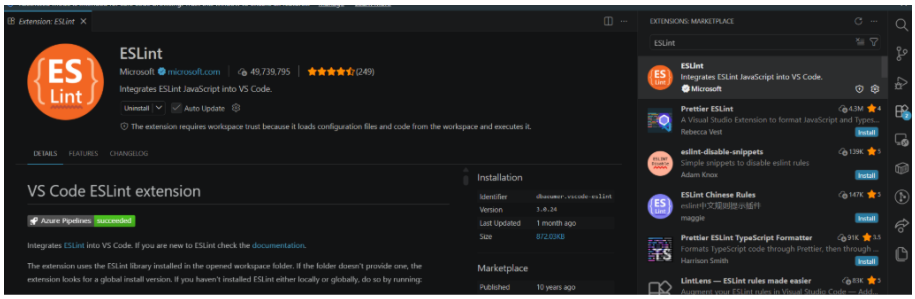


Figure 14: ESLint real-time code analysis in VS Code[Web15]

3.2 AI-Powered Code Review Assistants

The second generation of code review tools moves beyond rule-based analysis and incorporates machine learning, large language models, and semantic analysis to provide more intelligent, context-aware feedback. Unlike traditional static analyzers, AI-powered assistants are trained on large corpora of code and can reason about patterns, detect non-obvious bugs, understand code intent, and generate human-readable explanations for the issues they find [20,21,22]. This section examines three representative tools in this category: Amazon CodeGuru, CodeRabbit, and DeepCode (now Snyk Code).

- **Amazon CodeGuru** Amazon CodeGuru Reviewer is a machine learning-based service developed by AWS that automatically reviews code in pull requests. It was trained on more than 10,000 open-source projects and uses a combination of supervised machine learning models and automated program analysis to detect issues that are difficult for developers to find manually ; such as resource leaks, concurrency race conditions, incorrect input validation, and deviations from AWS API best practices [web16]. CodeGuru integrates directly with repositories on GitHub, GitHub Enterprise, Bitbucket, and AWS CodeCommit, and posts its findings as comments directly on the pull request.

  One of CodeGuru's notable characteristics is that it does not flag syntactical mistakes; it focuses exclusively on more complex, semantic-level problems. It also includes a Secrets Detector that identifies hardcoded credentials and API keys in the codebase. Despite its strengths, CodeGuru's language support is currently limited to Java and Python, and the service has been closed to new customers as of November 2025, reflecting the rapid pace of evolution in this space [web16].

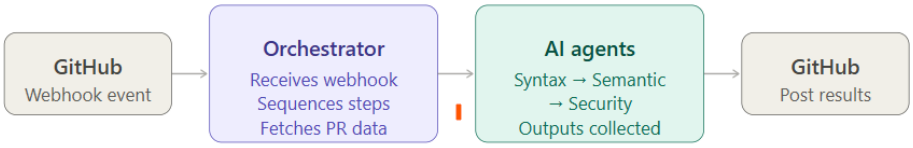


Figure 15: Amazon CodeGuru Reviewer findings interface[Web17]

- **CodeRabbit** CodeRabbit is a modern, LLM-powered code review platform that has quickly become one of the most widely adopted AI review tools in the industry, with over 2 million repositories connected and more than 13 million pull requests reviewed [web18]. It integrates with GitHub, GitLab, Azure DevOps, and Bitbucket, and automatically activates on every new pull request. CodeRabbit uses the latest foundational LLMs enhanced by its own built-in intelligence, which analyses the codebase, file dependencies, and team coding patterns to provide highly contextual, accurate feedback.

  What distinguishes CodeRabbit from earlier AI tools is its multi-layered approach: it combines the output of over 40 linters and SAST tools with LLM-based reasoning, filtering out noise and surfacing only actionable findings. It generates a high-level PR summary, a file-by-file walkthrough, and inline comments with one-click fix suggestions. It also learns from developer feedback over time, continuously improving the relevance of its reviews. While CodeRabbit represents a significant advancement, it operates as a general-purpose reviewer and does not build an explicit call graph or model of inter-function dependencies [web18].

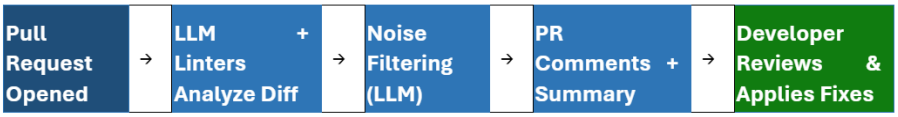


Figure 16: CodeRabbit AI-powered review pipeline [AIGC 1]

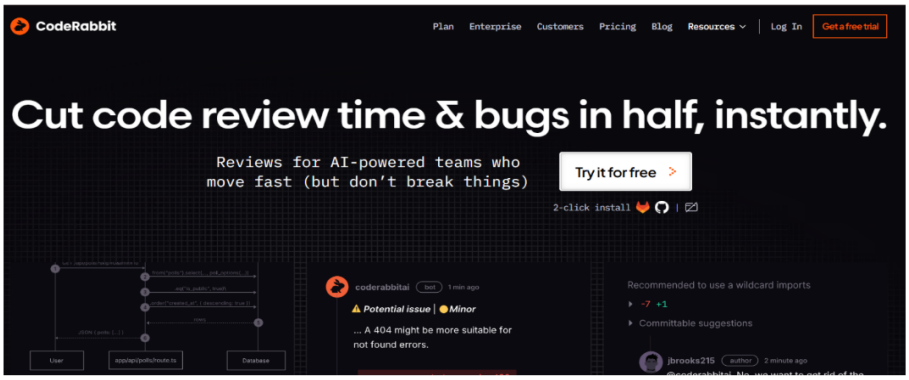


Figure 17: AI Code Reviews | CodeRabbit[Web18]

- **DeepCode (Snyk Code)** DeepCode was founded as an ETH Zurich spin-off and acquired by Snyk in 2020, where its technology now powers the Snyk Code platform under the name DeepCode AI [Web 19]. It represents one of the most technically sophisticated approaches to AI-based code analysis, combining symbolic AI ; which performs precise, rule-based data flow analysis ; with machine learning models trained on over 25 million data flow cases across 19+ programming languages. This hybrid approach allows DeepCode AI to detect complex vulnerability patterns such as SQL injection, cross-site scripting, and resource leaks with significantly fewer false positives than traditional static analyzers.

  A key innovation in DeepCode is its CodeReduce algorithm, which extracts only the code fragments relevant to a detected vulnerability before passing them to a Large Language Model for fix generation. This focused context extraction dramatically improves the accuracy and relevance of the proposed fixes, achieving reported autofix accuracy of approximately 80% [23]. DeepCode AI also supports real-time in-IDE scanning, CI/CD pipeline integration, and custom rule creation through semantic code search ; making it one of the most complete AI-powered security analysis platforms available today.

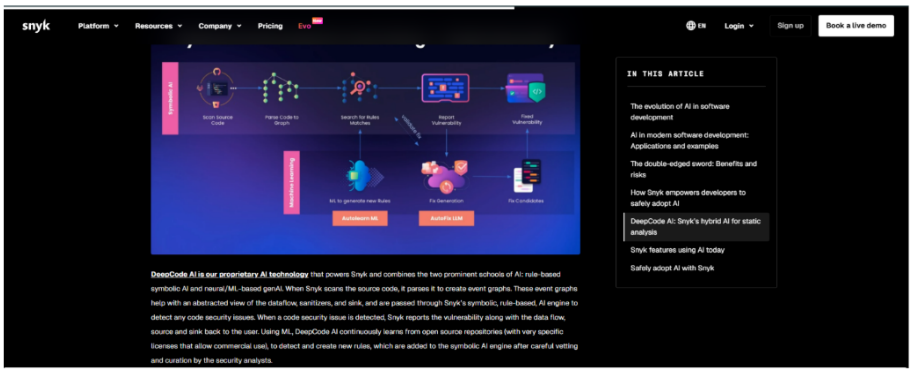


Figure 18: DeepCode (Snyk Code) vulnerability analysis dashboard[Web19]

- **Claude Code Review (Anthropic)** In March 2026, Anthropic launched a dedicated code review tool built on top of its Claude models, specifically targeting the growing volume of AI-generated code in modern software projects. The tool is designed to evaluate code quality, detect potential bugs, and flag security vulnerabilities in pull requests, with a particular emphasis on reasoning about code correctness and intent rather than surface-level pattern matching. Unlike traditional static analyzers, it leverages Claude's large context window to reason across multiple files simultaneously, making it capable of identifying cross-module issues and subtle logic errors that narrower tools miss. Its release reflects a broader industry trend of applying frontier LLMs directly to the software development lifecycle as AI-generated code becomes increasingly prevalent [Web20,24,25].

Criterion	CodeGuru	CodeRabbit	DeepCode (Snyk)	Claude (Anthropic)
AI Technology	ML + program analysis	LLMs + 40+ linters	Symbolic AI + LLM	Frontier LLM
Languages	Java, Python only	All major languages	19+ languages	All major languages
GitHub Integration	Yes	Yes	Yes	Yes
Security Focus	Moderate	Moderate	Strong (SAST + data flow)	Strong (semantic reasoning)
Auto-fix Generation	No	Yes (one-click)	Yes (80% accuracy)	Yes
Call Graph / Dependencies	No	Partial (file dependencies)	Data flow graph	Multi-file context
Risk Scoring	No	No	Context-aware risk score	No
Team Decision Support	No	No	No	Yes

Table 5: Comparison of AI Code Review Tools

4 Conclusion

This chapter examined the fundamentals of version control through Git and GitHub, focusing on the collaborative mechanisms, pull requests and webhooks, that enable automated workflows. It then analyzed existing code review approaches, from traditional static analysis tools to AI-powered assistants, revealing their limitations in contextual understanding, adaptability, and workflow integration.

These limitations motivate the design of the proposed system, presented in the next chapter. Chapter 3 introduces the system's architecture and objectives, details its deterministic modules and intelligent agents, and positions the proposed approach relative to existing solutions. It then walks through the complete system organigram, from the developer's entry point to the final decision made by the team lead, before concluding with a performance study.

Chapter 3

System Design and Development

1 Introduction

The first two chapters of this thesis established the theoretical and contextual foundations of this work. Chapter 1 introduced the core concepts of agentic artificial intelligence, covering how autonomous agents reason, plan, and execute complex tasks, concepts that directly underpin the design choices made in this chapter. Chapter 2 examined the core mechanisms of GitHub, analyzed existing code review tools from traditional static analyzers to AI-powered assistants, and identified the limitations that motivated our contribution.

Having completed this state of the art, this chapter marks the beginning of our contribution. It details the design and development of the proposed AI-powered automated code review system. It begins by defining the system objectives and architecture, covering the key modules, intelligent agents, and workflow orchestration. This is followed by a detailed organigram illustrating the system components and their interactions, and a performance study evaluating the system behavior. The chapter closes with a concluding summary.

2 Positioning of the Proposed System

The proposed system is primarily targeted at development teams, open-source maintainers, DevOps teams, and engineering managers who need to review code efficiently without sacrificing quality. It is especially useful in projects with frequent pull requests, where manual review becomes repetitive and time-consuming. In such contexts, the system acts as an intelligent assistant that helps reviewers identify potential issues, understand code changes faster, and focus their attention on the most critical parts of the contribution.

Unlike static analyzers, the proposed system does not stop at surface-level rule violations. Unlike generic AI tools, it does not rely only on broad model knowledge. Instead, it uses the actual repository context, the nature of the Git event, and the structure of the code to produce more relevant and actionable feedback. This makes the system suitable for environments where accuracy, explainability, and workflow integration are important.

Therefore, the proposed solution is positioned as a decision-support tool for modern software development teams. Its objective is not to automate human judgment completely, but to reduce review workload, improve consistency, and accelerate the merge process while keeping final approval in the hands of developers and team leads. This positioning gives the project a clear identity within the landscape of existing code review solutions and highlights its practical contribution to collaborative software engineering.

3 Proposed System Design

This chapter presents the design of CodeWatchAI, an agentic AI-based system for automated code review. The proposed solution addresses the limitations of existing static analysis tools and reactive AI assistants by integrating a multi-agent architecture with deterministic processing modules, orchestrated through an event-driven pipeline connected to GitHub repositories.

3.1 Problem Statement and System Objectives

Traditional code review practices rely heavily on human reviewers and are often time-consuming, repetitive, and prone to inconsistencies, especially in large-scale software projects. To address these limitations, CodeWatchAI introduces an event-driven agentic pipeline that combines deterministic modules with specialized AI agents to deliver context-aware, automated code evaluation directly within GitHub workflows. The system pursues three core objectives:

- **Comprehensive analysis:** covering syntax, semantics, and security within a single unified pipeline.
- **Context-aware feedback:** leveraging call graph information to evaluate how local changes impact overall system behavior.
- **Workflow integration:** delivering near real-time feedback directly within GitHub development workflows through n8n orchestration[web21].

3.2 System Architecture

The proposed architecture is a hybrid, event-driven agentic AI framework combining two complementary layers: a deterministic processing layer for structured data acquisition and preprocessing, and an intelligent agent layer for reasoning, vulnerability detection, and decision-making. Both layers are coordinated by an orchestration engine implemented with n8n[web 21].

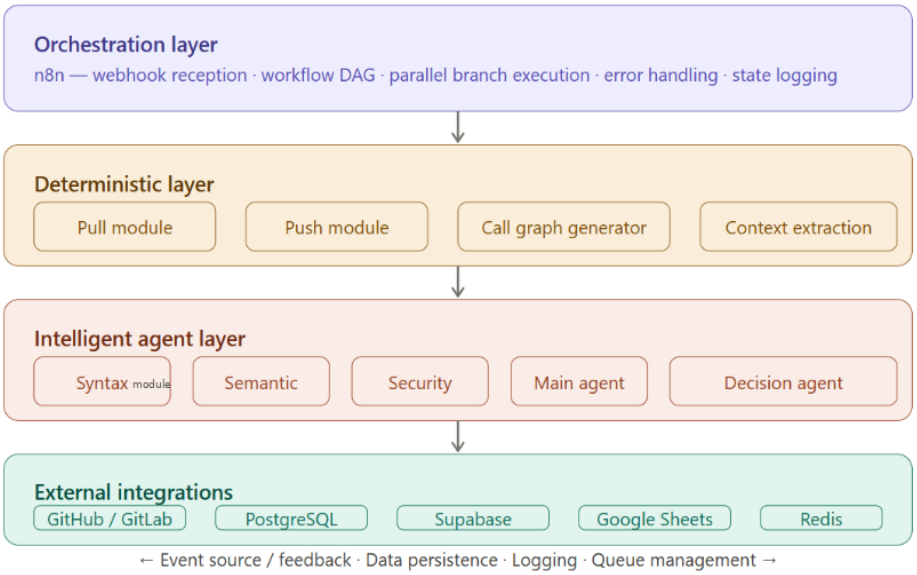


Figure 19: Detailed system architecture [AIGC 1]

The data flow follows a structured five-stage pipeline:

1. **Event Detection** GitHub webhook triggers the workflow.
2. **Data Retrieval** Pull and push modules fetch code changes via the GitHub API.
3. **Contextual Analysis** Call graph generation and context extraction.
4. **Agent-Based Evaluation** Parallel execution of specialized agents.
5. **Decision and Feedback** Consolidated review posted back to GitHub.

3.3 Deterministic Modules

The deterministic layer handles structured data acquisition and preprocessing prior to intelligent analysis. Operating through predefined, reproducible logic, these modules ensure reliability and traceability when handling repository data. Table 4.1 summarizes their roles.

Module	Trigger	Responsibility
Pull Module	Pull Request event	Retrieves modified files, diffs, metadata, and comments via GitHub REST API.
Push Module	Push event	Monitors direct branch pushes and retrieves the latest codebase snapshots.
Call Graph Generator	Post-retrieval	Constructs the function dependency graph and execution-flow representation.
Context Extraction Module	Post-call graph	Enriches analysis with semantic context, dependency relationships, and version history.
Syntax Module	Lexical and syntactic validation	Detects naming inconsistencies, improper error handling, and language idiom deviations using github's CI

Table 6: Deterministic Modules Summary

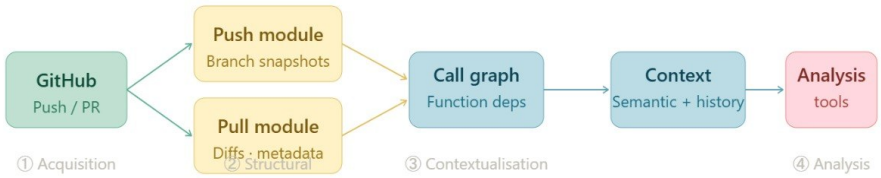


Figure 20: Deterministic processing pipeline [AIGC 1]

3.4 Intelligent Agents

The system employs five specialized intelligent agents, each addressing a distinct dimension of code quality. This division of responsibilities follows established multi-agent software engineering patterns, where task specialization enables effective handling of complex workflows. Agents communicate through a shared blackboard architecture maintaining analysis state and intermediate results.

Agent	Role	Key Capability
Semantic Agent	Logical correctness and architectural coherence	Traces data flows across function boundaries and evaluates the impact of changes on system invariants [26,27].
Security Agent	Vulnerability detection	Identifies injection flaws, authentication weaknesses, and authorization bypass patterns using LLM-based reasoning.
Main Agent	Central coordination	Decomposes tasks, distributes work to specialized agents, and aggregates outputs using a cold validation pattern.
Decision Agent	Final review synthesis	Evaluates severity, resolves conflicts, and determines approval, conditional acceptance, or rejection.

Table 7: Intelligent Agents Summary

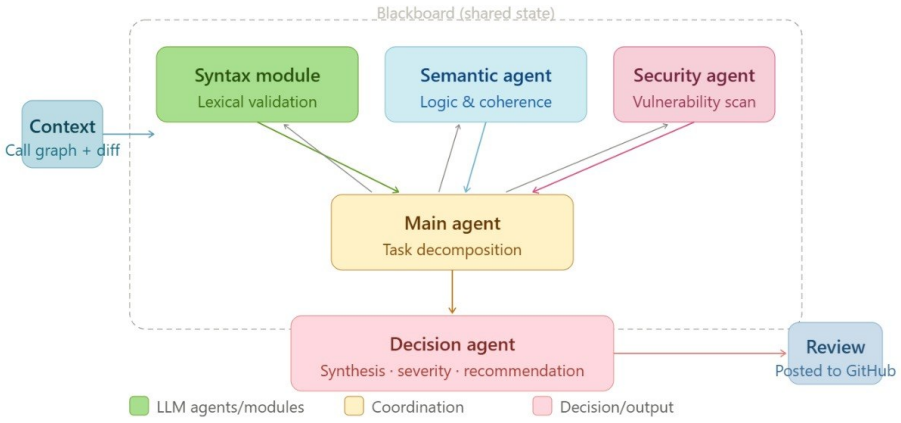


Figure 21: Multi-agent collaboration (blackboard architecture) [AIGC 1]

3.5 Workflow Orchestration with n8n

The system uses n8n as its central workflow orchestration engine[web21], providing an event-driven automation layer that coordinates GitHub repositories, deterministic modules, and intelligent agents within a single visual pipeline.

Workflows are structured as directed acyclic graphs within n8n. A Webhook Node captures incoming GitHub events, a Function Node extracts metadata and determines analysis scope, and HTTP Request Nodes invoke deterministic modules. Parallel execution of the Syntax, Semantic, and Security agents is achieved through n8n's concurrent

branches, with a Merge Node synchronizing outputs before the Decision Agent synthesizes the final review.

Key infrastructure choices include:

- **Supabase vector database**: stores results for retrieval-augmented generation.
- **Google Sheets integration**: maintains execution logs for auditability.
- **Redis-backed queue mode**: distributes workloads across workers for production scalability.
- **HTTPS with reverse proxy**: ensures secure webhook reception and credential encryption.

3.6 System Workflows

The system implements two primary event-driven workflows corresponding to GitHub's core development activities.

- **Push Workflow**: Triggered when a developer pushes commits to a monitored branch, this workflow validates the webhook signature, extracts commit metadata, and invokes the deterministic modules to construct a structural representation of the changes. For minor updates such as documentation modifications, the workflow terminates early without generating feedback. For substantive changes, the Syntax, Semantic, and Security agents execute concurrently, and the Decision Agent posts inline comments to the relevant commit via the GitHub API.
- **Pull Request Workflow**: Triggered when a pull request is opened, updated, or synchronized, this workflow retrieves the full changeset diff and executes all five agents, with the Semantic Agent performing cross-file analysis across the entire pull request scope.

  The Decision Agent produces a consolidated review submitted formally to GitHub, including categorized findings, severity ratings, and inline comments. Blocking issues trigger a "changes requested" status, preventing merge until resolved.

Both workflows maintain execution logs through n8n's Google Sheets integration, enabling continuous auditability and pipeline improvement.

4 AI-Powered Code Review Assistants

This section presents the full architectural organigram of the system. Each diagram below illustrates a distinct module or component of the pipeline, from the initial developer action to the final team lead decision.

4.1 System Architecture and Workflow

As illustrated in Figure 23, The organigram provides a structured hierarchical view of the system's components and their organizational relationships. It maps the full dependency chain from the developer at the top through the automation middleware, analysis agents, and final human decision layer.

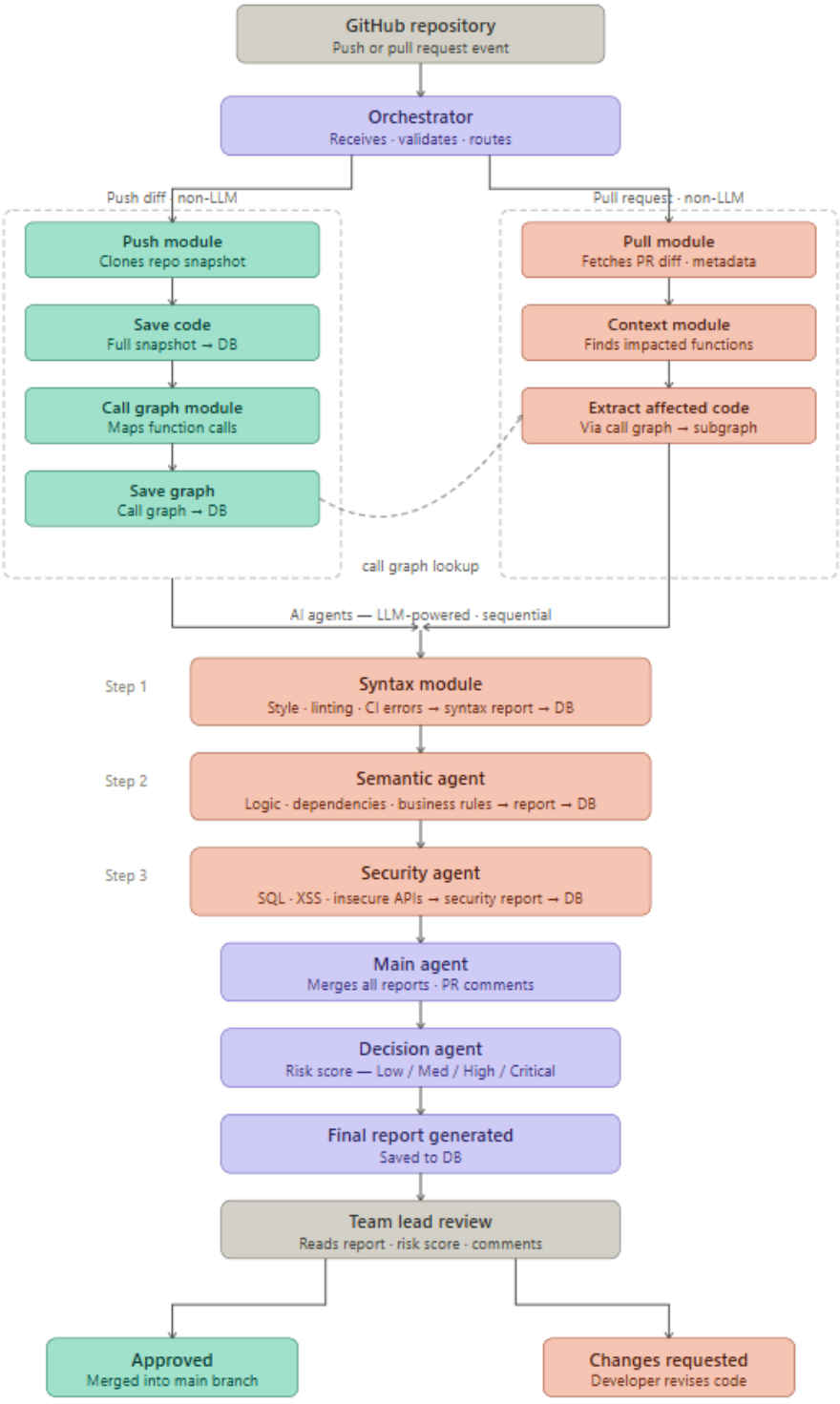


Figure 22: AI-Powered Code Review Pipeline [AIGC 1]

4.2 Developer-Entry Point

The process begins with the developer, who writes and commits code to a feature branch. As shown in Figure 24, Once satisfied, the developer opens a Pull Request (PR) on GitHub, which serves as the trigger for the entire automated review pipeline. This initial action initiates the workflow without requiring any manual intervention from reviewers.

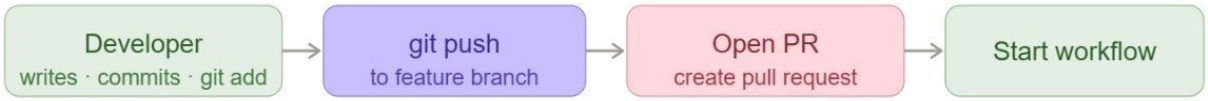


Figure 23: Developer opens a Pull Request on GitHub [AIGC 1]

4.3 GitHub-Trigger and Version Control

GitHub acts as the central hub for source code management and event handling. As depicted in Figure 25, When a new Pull Request is created or updated, GitHub emits a webhook event that notifies the automation layer. It also hosts the repository, stores pull request metadata, and provides the PR-diff of changed files that will be analyzed by the AI agents.

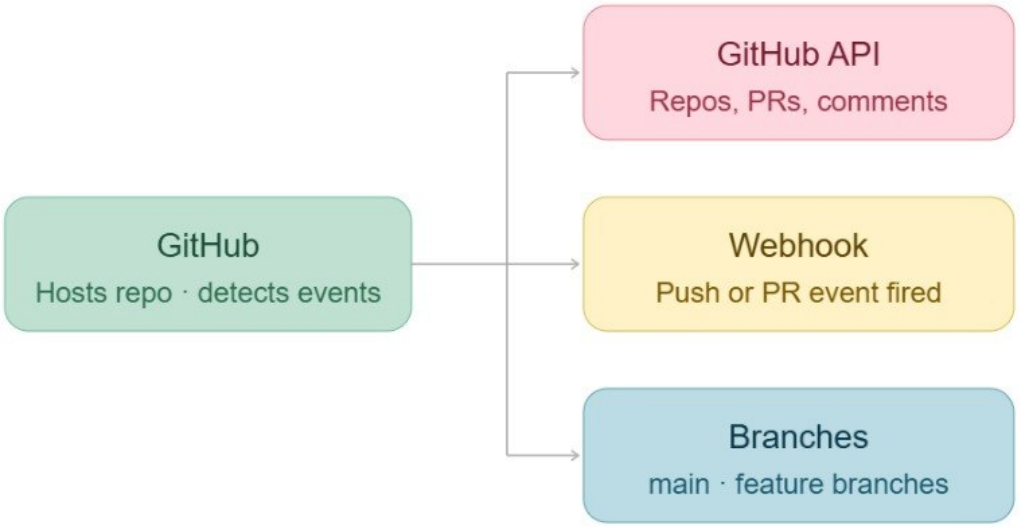


Figure 24: GitHub as the code host and webhook trigger [AIGC 1]

4.4 Workflow Orchestration Layer

The orchestration layer serves as the backbone of the pipeline's automation logic. As illustrated in Figure 26, it listens for incoming GitHub webhook events and orchestrates all subsequent actions. It is responsible for fetching pull request data, dispatching tasks

to the AI agents, collecting their outputs, and posting results back to GitHub. This layer provides a visual workflow environment that makes the pipeline maintainable and extensible.

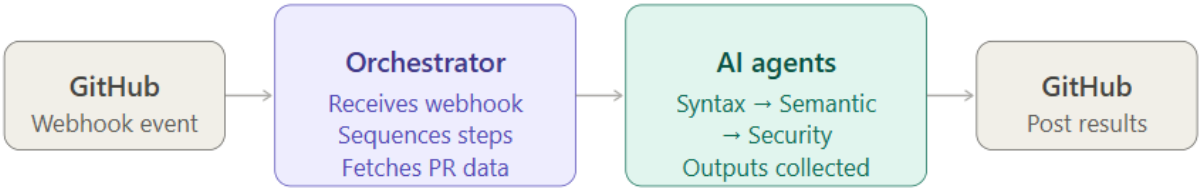


Figure 25: Orchestrating the full automation workflow [AIGC 1]

4.5 Push Module-Data Ingestion

The Push Module handles push events without using LLM models and is linked to the code in the main branch. As shown in Figure 27, It consists of three sequential operations executed in the following order:

1. Downloading the repository to obtain the latest version of the code.
2. Updating the database to store the repository's code.
3. Rebuilding the call graph to determine which parts of the code call each other.

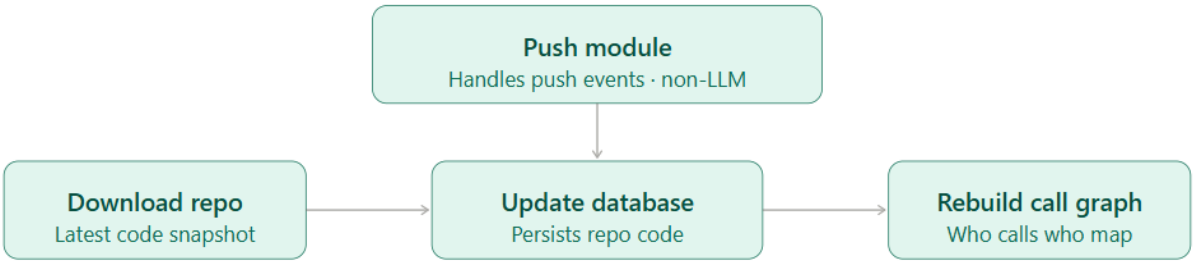


Figure 26: Push module: sequential data ingestion pipeline [AIGC 1]

4.6 Pull Module-Data Retrieval

The Pull Module is responsible for handling Pull Request events coming from GitHub without relying on any language model (Non-LLM).

As depicted in Figure 28, It performs three core tasks sequentially in the following order:

1. Fetching the PR-diff to capture exactly what changed in the code.
2. Storing the associated metadata in the database for later use.

3. Forwarding the PR-diff to the Context Module.

The Context Module is the first to consume this data, using it to extract the call graph or identify the relevant parts of the main branch codebase that are related to the proposed changes.

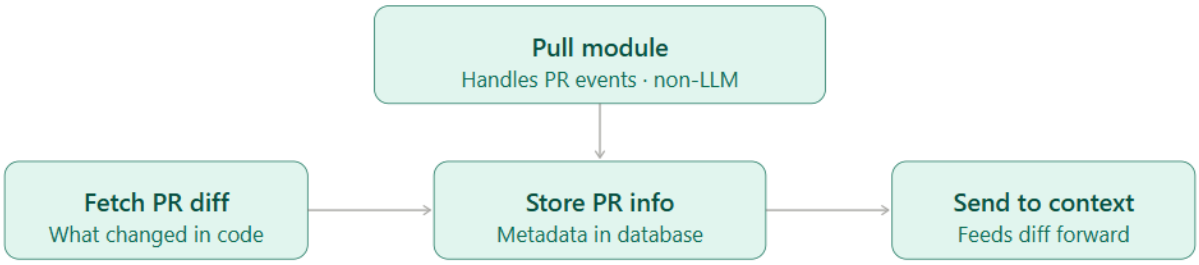


Figure 27: Pull module: sequential PR data retrieval pipeline [AIGC 1]

4.7 Context Module-Contextual Enrichment

The Context unit identifies and extracts only the code elements related to the PR-diff, without using any language model. As illustrated in Figure 29, it retrieves the relevant code based on the call graph and uses it to understand the relationships between different parts of the codebase.

It then builds a subgraph that includes both the functions called by the modified code and the functions that call it. This is done after identifying the functions affected by the changes in the PR-diff. The unit then fetches files, functions, or code snippets that are directly or indirectly related to these modifications.

By providing context based on dependencies and relationships between system components, this unit compiles a focused view of the relevant parts of the system, which significantly improves the accuracy and relevance of the information provided to agents.

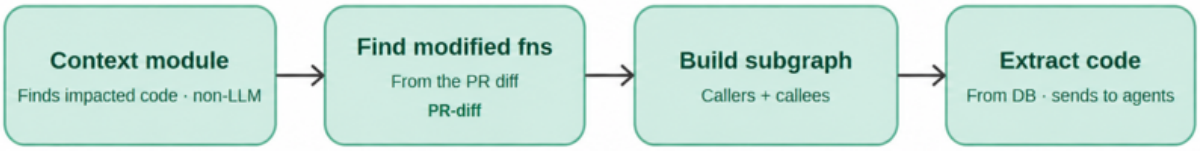


Figure 28: Context Module enriching data before agent analysis [AIGC 1]

4.8 The Analysis Tools

At the core of the system lie two AI agents and one dedicated module, each responsible for a distinct dimension of code quality. As shown in Figure 29, these three components operate as follows:

- **Semantic Agent**: analyzes logic errors, potential bugs, performance issues, and business logic correctness.
- **Security Agent**: detects vulnerabilities such as SQL injection, XSS, insecure API usage, and dependency risks.
- **Syntax Module**: checks coding standards, code style, CI tool feedback, and produces inline suggestions through static analysis.

Unlike the two agents, the Syntax Module does not rely on a language model.

Each of the three components independently produces a sub-report, which is then forwarded to the Main Agent for aggregation.

Running concurrently significantly reduces total analysis time while keeping each concern cleanly separated.

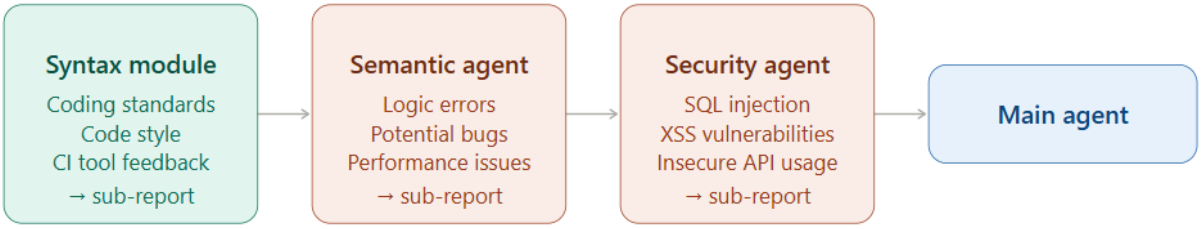


Figure 29: The sequential execution of Analysis Tools [AIGC 1]

4.9 Main Agent and Decision Agent

As depicted in Figure 31, The Main Agent collects the three individual reports and performs deduplication and merging. It generates unified, human-readable comments tied to specific lines and functions in the pull request, and saves the complete structured report to the database.

The Decision Agent then processes the merged report by applying a weighted scoring algorithm. Security issues carry the highest weight, followed by logic bugs and style violations. Based on the aggregated score, the agent assigns one of four risk levels:

- Low • Medium • High • Critical

It then generates a final summary report and posts preliminary comments directly to the GitHub PR through the orchestration pipeline.

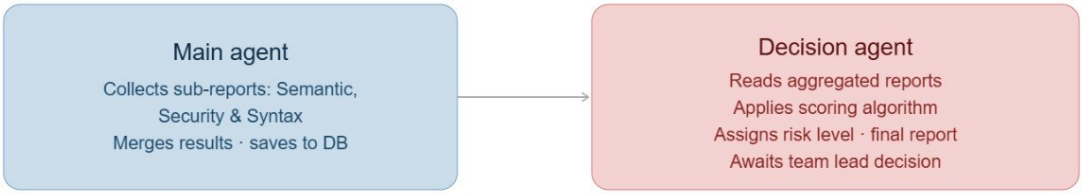


Figure 30: Main Agent merging reports and Decision Agent scoring [AIGC 1]

4.10 Team Lead-Final Human Decision

As illustrated in Figure 32, The Team Lead is the final decision-maker in the pipeline. After receiving the risk-scored report and reading the AI-generated inline PR comments on GitHub, the team lead applies business context and project-specific knowledge that the AI cannot account for.

Based on this review, the team lead either approves the pull request, triggering a merge into the main branch, or requests changes, sending the developer back to revise the code and open a new review cycle.

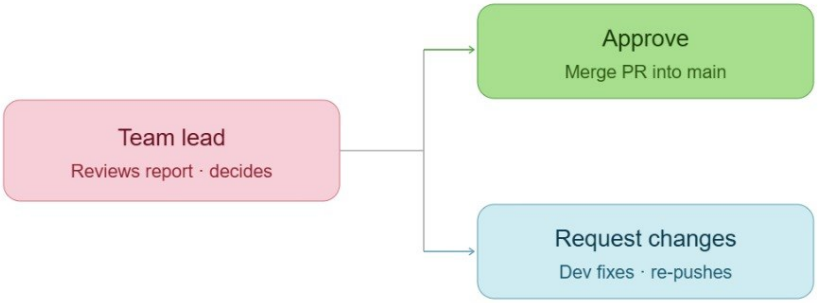


Figure 31: Team Lead making the final approval or change request [AIGC 1]

5 Performance Study

- Context

The CodeWatchAI system currently relies on the Gemini 2.5 Flash-Lite model to power its intelligent agents (Semantic, Security, Main, and Decision agents) responsible for analyzing Pull Requests. To objectively assess this choice, a comparative study was conducted across three variants of the same model family, Gemini 2.5 Flash-Lite, Gemini 2.5 Flash, and Gemini 2.5 Pro, which share the same context window (1M tokens) and the same integration infrastructure (Gemini API, n8n orchestration), allowing the effect of the model

choice on system performance to be isolated.

- Comparison Table

Criterion	Gemini 2.5 Flash-Lite	Gemini 2.5 Flash	Gemini 2.5 Pro
Intelligence Index (AI Analysis)	7	14	27
SWE-Bench Verified	not disclosed	high	63.8%
Output speed (tokens/s)	≈243	≈218	≈133
Input price (USD / 1M tokens)	0.10	0.30	≈1.25
Output price (USD / 1M tokens)	0.40	2.50	10.00
Context window	1M tokens	1M tokens	1M tokens
Recommended use	fast triage, high volume	code analysis	deep reasoning (semantic, security)

Table 8: Comparison of Gemini 2.5 variants for semantic and security agents

- Graphical Representation

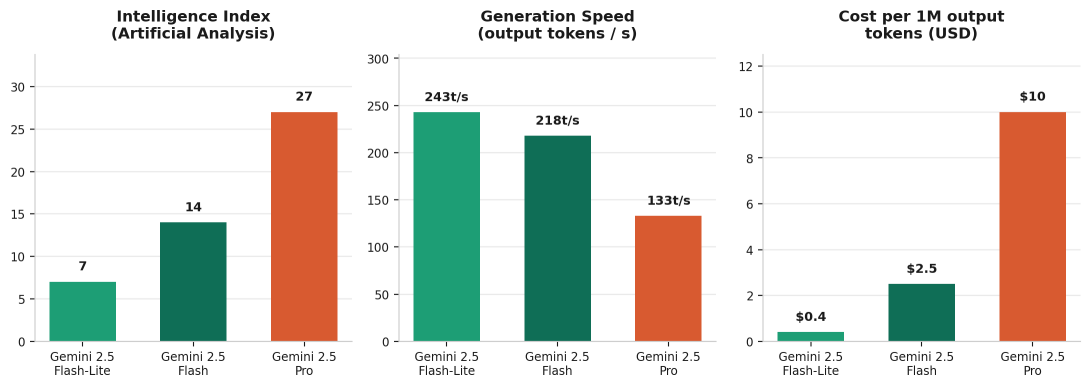


Figure 32: Comparison of intelligence index, generation speed, and cost per million output tokens across the three Gemini 2.5 variants.

- Interpretation

- Gemini 2.5 Flash-Lite (the model currently in use) offers the highest speed and the lowest cost, but shows the lowest intelligence index of the three variants, a trade-off consistent with a system that must process a high volume of Pull Requests continuously.
- Gemini 2.5 Flash represents an intermediate trade-off: an intelligence index roughly twice that of Flash-Lite, a slightly lower speed, at an output cost about 6 times higher.
- Gemini 2.5 Pro delivers the best reasoning capability (63.8% on SWE-Bench Verified), particularly valuable for the Semantic and Security agents, which require a fine-grained understanding of the code, but at the cost of roughly 45% higher latency and a cost about 25 times higher than Flash-Lite.

- Conclusion and Outlook

The choice of Gemini 2.5 Flash-Lite remains justified for an academic prototype that prioritizes speed and API cost control. However, a hybrid architecture could be considered as a future improvement: keeping Flash-Lite for fast triage tasks (Syntax module) and reserving Gemini 2.5 Pro for the Security agent, where a false negative (an undetected vulnerability) carries a potentially high cost, while keeping Flash-Lite or Flash for the Semantic agent depending on the available budget.

6 Conclusion

In this chapter, we designed an AI-powered automated code review system that integrates with GitHub through an orchestration pipeline.

We defined the system objectives, presented its architecture, and detailed each component through the organigram, from the developer entry point, through the deterministic modules and sequential AI agents, to the final team lead decision. The performance study confirmed that the system delivers fast and consistent results compared to traditional review workflows.

This foundation sets the stage for the next chapter, which covers the practical implementation of the system, the tools and technologies selected, and a comparative evaluation against existing platforms.

Chapter 4

Implementation

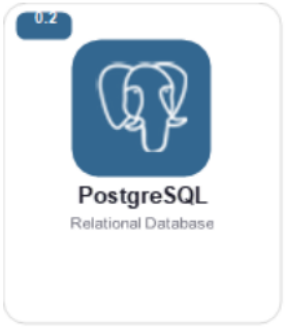
1 Introduction

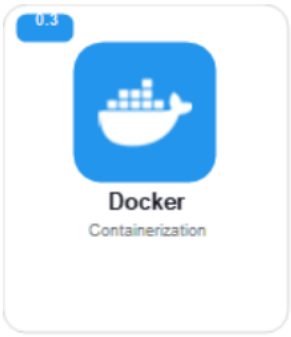
After having established the theoretical foundations of agentic AI in Chapter 1, explored the GitHub ecosystem and its APIs in Chapter 2, and defined the architecture and conception of our system in Chapter 3, we now turn to the concrete realization of the project. This chapter is dedicated to the practical implementation of the agentic code review system designed in the previous chapter.

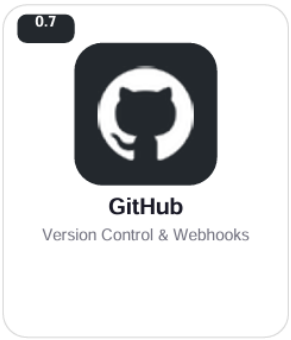
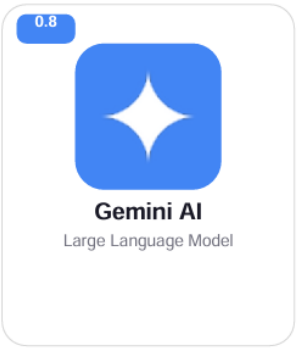
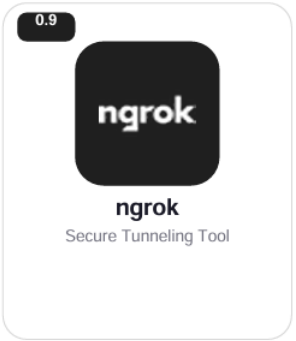
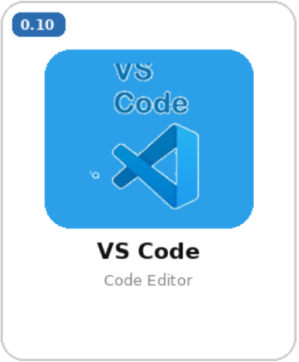
We begin by presenting the tools and technologies selected to build the system, justifying each choice in relation to the requirements identified during the conception phase. We then conduct a comparative study of existing platforms that offer similar functionalities, extending the overview introduced in Chapter 2 with a more focused comparison centered on the specific capabilities of our system and the added value of our approach. Finally, we present the key interfaces of the application, illustrating the main features and the user experience delivered by the system.

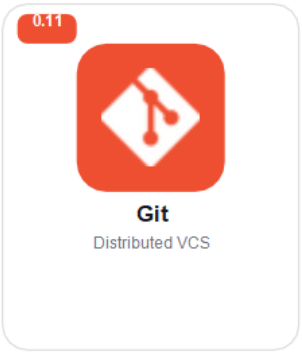
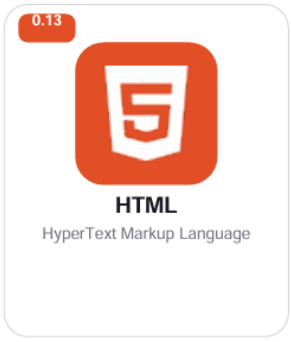
2 Tools and Technologies

Usage in this project: Structural foundation of the frontend

Logo	Description and Usage
	**n8n-Workflow Automation Platform** **Definition:** n8n is a free, open-source, node-based workflow automation tool that connects services, APIs, and databases through visual workflows[Web 21, Web 22]. **Usage in this project:** Central orchestration engine coordinating all automated workflows across the system.
	**PostgreSQL-Relational Database Management System** **Definition:** PostgreSQL is a powerful open-source object-relational database management system supporting SQL and JSON data models[Web 23]. **Usage in this project:** Persistent storage for call graph data, pull request records, and review results.

Logo	Description and Usage
	**Docker-Containerization Platform** **Definition:** Docker is a platform that packages applications and dependencies into lightweight containers[Web 24]. **Usage in this project:** Runs n8n, PostgreSQL, Django, Python services, and ngrok in isolated containers.
	**WSL-Windows Subsystem for Linux** **Definition:** WSL allows running Linux directly inside Windows without a virtual machine[Web 25]. **Usage in this project:** Used as development environment for Docker and Python tooling.
	**Django-Python Web Framework** **Definition:** Django is a high-level Python framework based on the MVT architecture[Web 26]. **Usage in this project:** Backend REST API layer running inside Docker.
	**Tailwind CSS-Utility-First Framework** **Definition:** Utility-first CSS framework for building modern UIs[Web 27]. **Usage in this project:** Used for styling the review dashboard.

Logo	Description and Usage
	**GitHub-Version Control Platform** **Definition:** Cloud platform for Git repository hosting and collaboration[Web 12]. **Usage in this project:** Triggers automated pipeline via webhooks on pull requests.
	**Gemini AI-Large Language Model** **Definition:** Multimodal LLM by Google DeepMind for reasoning over code and text[Web 28]. **Usage in this project:** Performs semantic code analysis and generates review reports.
	**ngrok-Secure Tunneling Tool** **Definition:** Creates public URLs for local services[Web 29]. **Usage in this project:** Exposes local n8n webhook to GitHub during development.
	**Visual Studio Code** **Definition:** Lightweight cross-platform code editor[Web 30]. **Usage in this project:** Main development IDE.

Logo	Description and Usage
	**Git-Distributed Version Control System** **Definition:** Tracks changes in source code over time[Web 31]. **Usage in this project:** Manages versioning and PR workflow.
	**Python** **Definition:** High-level programming language[Web 32]. **Usage in this project:** Backend logic and AI processing.
	**HTML** **Definition:** Standard markup language for web pages[Web 33]. **Usage in this project:** Frontend structure.

3 Comparative Analysis: Proposed System vs CodeRabbit

This section presents a focused comparison between the proposed system (CodeWatchAI) and CodeRabbit, as one of the closest existing AI-powered code review tools.

Feature	**Proposed System (CodeWatchAI)**	**CodeRabbit**
Call Graph Construction	Fully supported (explicit function-level dependency graph)	Not supported
Multi-Agent Architecture	Supported (Semantic, Security, and Decision agents working in parallel with explicit task decomposition)	LLM-based review pipeline with multiple analysis stages; internal architecture not fully disclosed
Risk Scoring Mechanism	Weighted risk scoring with structured aggregation and explainable output	Not explicitly provided as a formal scoring model
Merge Recommendation	Explicit automated merge decision using a structured decision engine	Provides review feedback and suggestions, without an explicit binary merge decision system
Analysis Depth	Structural, semantic, and security analysis with function-level dependency tracking	Context-aware pull request analysis based on diff and repository context, combined with static analysis signals
Platform Support	GitHub + GitLab	GitHub + GitLab + Bitbucket + Azure DevOps
Architecture Type	Agentic AI pipeline orchestrated via workflow engine (n8n) with modular domain-specific agents	Commercial AI-assisted code review system integrated into CI/CD workflows
Output Structure	Structured, explainable decision output with risk scores	Natural language pull request comments and summarized review feedback
Customization	High (modular agents and configurable pipeline design)	Moderate (rule-based configuration and repository-level settings)
Deployment Model	Open architecture (SaaS-ready, LLM API-based design)	Proprietary SaaS platform

Table 10: Comparative Analysis of CodeWatchAI vs CodeRabbit

4 Conclusion

Throughout this chapter, we have presented the concrete implementation of our agentic code review system. We detailed the technical stack and tools that underpin the solution, compared our approach against existing platforms, and illustrated the system through its main interfaces.

The work carried out demonstrates the feasibility of applying agentic AI principles to the problem of automated code review within GitHub workflows. The system is capable of autonomously analyzing pull requests, providing structured feedback, and delivering a final verdict on whether a pull request should be accepted or rejected without modifying the code .

General Conclusion

This thesis explored the feasibility of designing and implementing an agentic AI system capable of automating code review within modern software development workflows. Over four chapters, we progressively built the theoretical, technical, and practical foundations of CodeWatchAI from conceptual understanding to a fully functional prototype. We began by establishing the theoretical groundwork of Agentic AI and analyzing the limitations of existing code review solutions. We then examined GitHub's core mechanisms and how they provide a programmable environment for integrating automated tools. Building on this, we designed a multi-agent architecture combining a semantic agent, a security agent, and a decision agent, each responsible for a distinct dimension of code analysis. Finally, we presented the full implementation along with a comparative analysis demonstrating CodeWatchAI's added value over existing platforms. Beyond its academic contribution, the long-term vision for CodeWatchAI is to evolve into a specialized technology company focused on Agentic AI solutions and intelligent workflow automation, contributing to the digital transformation of organizations across Algeria and beyond. In conclusion, this work demonstrates the transformative potential of Agentic AI in modern software engineering, providing a practical and innovative foundation for automating code review while improving software quality, security, and development efficiency.

Perspectives Several directions remain open for future development. Technically, extending language support beyond Python, adopting AST-based call graph construction, and integrating RAG techniques and adaptive learning mechanisms would significantly improve the system's depth and accuracy. From a product standpoint, native GitLab support and team notification integrations would enhance usability. Commercially, the priority is deploying the platform as a SaaS product, engaging early adopters, and pursuing the Algerian government's startup label for institutional support. Finally, from a research perspective, future work could evaluate agent accuracy across diverse codebases and measure the real-world impact of automated review on developer productivity and software quality.

Chapter 5

Annex

1 First Axis: Project Presentation

1.1 Project Idea

The field of activity of this project falls within the domain of Artificial Intelligence applied to software development tools and DevOps practices.

- **What is the problem?** Manual code review in software teams is slow, inconsistent, and heavily dependent on senior developer availability.
- **How did the idea start?** Through direct observation of this recurring problem and the study of existing tools and their inability to understand the structural impact of code changes.
- **What will it do?** Build an intelligent multi-agent AI system that automatically reviews GitHub Pull Requests using three specialized analysis tools, Syntax, Semantic, and Security, where Semantic and Security are AI agents while Syntax is a non-LLM module, then generates a risk-scored report delivered to the Team Lead.
- **How will it work?** A developer opens a Pull Request GitHub sends a webhook to n8n the system builds a Call Graph agents analyze the affected code the Main Agent posts comments on the PR the Decision Agent delivers the final report to the Team Lead.
- **Who will build it?** A student team specializing in Networks and Distributed Systems within the university incubator framework.
- **Where will it be deployed?** A cloud environment accessible remotely by any GitHub-based development team, with initial focus on the Algerian market.

The idea of the project is represented in the picture below:

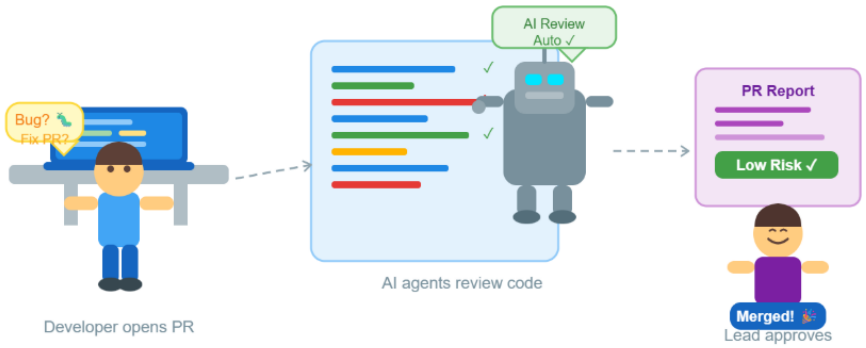


Figure 33: General idea of the proposed CodeWatchAI [AIGC 1]

1.2 Proposed Values

This project is based on core values aimed at improving software development quality and team efficiency through an intelligent and integrated code review system.

- **Improved Code Quality using AI:** The system uses Large Language Models (LLMs) to analyze semantic (business logic) and security aspects of the code, enabling early detection of errors and vulnerabilities. Code structure (Syntax) is also checked through a dedicated module to ensure coding standards and improve maintainability.
- **Security and Decision Support:** The system detects vulnerabilities such as SQL Injection and XSS before deployment. It also provides a risk assessment (Low, Medium, High, Critical) and a merge recommendation using an LLM-based decision agent to support objective decisions.
- **Efficiency and Performance:** The system automates code review using a multi-component architecture, reducing manual effort and producing fast, structured reports. It relies on a sequential pipeline between syntax, semantic, and security analysis components, ensuring each layer builds on the results of the previous one.
- **Flexibility and Accessibility:** It integrates easily with GitHub and GitLab via webhooks without complex setup. It is also cost-efficient, making it suitable for small and medium-sized teams.
- **Transparency and Innovation:** Built on an open-source architecture, the system ensures transparency and avoids black-box behavior. It combines Call Graph analysis, deterministic modules, and multi-agent AI into a unified hybrid solution for better code understanding and risk analysis.

1.3 Team Members

The project team consists of three members, each bringing complementary skills and responsibilities to the development of the system:

Member	Specialization	Role in the Project
Boukalkoul Sarra	Master 2 – Networks and Distributed Systems	Frontend development, UI design, documentation, and testing
Bounour Fatima Zohra	Master 2 – Networks and Distributed Systems	AI workflow orchestration, backend integration, and system conception
Chaouch Malak Fatima Zohra	Master 2 – Networks and Distributed Systems	Database design, call graph analysis, and system validation

Table 11: Project Team Members

The team interacts through regular meetings and collaborative tools such as GitHub for version control and shared task tracking. Responsibilities are distributed according to each member's strengths to ensure balanced and efficient progress throughout the project lifecycle.

1.4 Project Objectives

The project aims to achieve the following objectives:

- **Short term:** Deliver a fully functional prototype of the agentic code review system, capable of autonomously analyzing pull requests on GitHub and producing structured feedback with an acceptance or rejection verdict.
- **Medium term:** Refine the system based on real usage feedback, improve agent accuracy, and expand support to additional programming languages beyond Python.
- **Long term:** Position the platform as a reliable AI-assisted DevOps tool that can be commercially deployed for software development teams and organizations seeking to automate and standardize their code review process.

1.5 Timeline for Project Implementation

Project Timeline — CodeWatchAI

#	Task Name	Phase	Duration (Weeks)	Month 1				Month 2				Month 3			
				S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12
1	Study Agentic AI & LLMs	Research & Study	1												
2	Study Git/GitHub & Webhooks	Research & Study	1												
3	Analyse existing tools	Research & Study	1												
4	Define system objectives & scope	Design	1												
5	Design system architecture	Design	1												
6	Design organigram & pipeline	Design	1												
7	Design DB schema & data models	Design	1												
8	Setup n8n & GitHub webhooks	Implementation	1												
9	Implement Push module	Implementation	1												
10	Implement Pull module	Implementation	1												
11	Implement Context module	Implementation	1												
12	Implement Call Graph module	Implementation	1												
13	Implement Syntax module (static)	Implementation	1												
14	Implement Semantic agent (LLM)	Implementation	2												
15	Implement Security agent (LLM)	Implementation	2												
16	Implement Main agent & aggregation	Implementation	1												
17	Implement Decision agent & scoring	Implementation	1												
18	Unit testing – all modules	Testing	1												
19	Integration testing – full pipeline	Testing	1												
20	Performance evaluation & metrics	Testing	1												
21	Bug fixes & optimisation	Testing	1												
22	Write Chapter 1 – Agentic AI	Writing	2												
23	Write Chapter 2 – VCS & Design	Writing	2												
24	Write Chapter 3 – Analysis & Concept.	Writing	2												
25	Write Chapter 4 – Implementation	Writing	2												
26	Final revision & formatting	Writing	1												
27	Prepare presentation slides	Presentation	1												
28	Final defence preparation	Presentation	1												
29	Annex – Part 1 (Axes 1, 2 & 3: Presentation, Innovation & Market Analysis)	Research & Study	2												
30	Annex – Part 2 (Axes 4, 5 & 6: Production Plan, Financial Plan & Prototype)	Writing	2												
	Color Key:			Research & Study		Design		Implementation		Testing		Writing		Presentation	

Figure 34: Project Implementation Timeline-CodeWatchAI

2 Second Axis: Innovative Aspects

2.1 Nature of the Innovations

The proposed project combines technological and market innovation to achieve product differentiation and long-term scalability through four complementary dimensions.

2.1.1 Radical Technological Innovation

The project transforms code review from a traditional inspection process into an intelligent decision-support system. It combines a multi-agent architecture, call graph analysis, and a risk-scoring mechanism to evaluate code changes in a deeper, contextual way. This enables impact analysis, risk detection, and clear recommendations to support the team lead's final merge decision, changing the core logic of code review.

2.1.2 Incremental Technological Innovation

The system improves existing code review tools by adding deeper semantic, security, and structural analysis. It enhances speed, reliability, and consistency of pull request evaluation while reducing manual intervention. It also integrates easily with platforms like GitHub and GitLab, making it practical for real development environments.

2.1.3 Radical Market Innovation

The project targets startups, Small and medium-sized enterprises(SMEs), and emerging development teams that cannot afford expensive enterprise solutions. Instead of a simple review assistant, it offers an intelligent merge decision system, creating a new category of AI-based code decision tools that support both technical evaluation and operational decision-making.

2.1.4 Incremental Market Innovation

The system is designed for gradual expansion from local to global markets. Its scalable architecture supports Software as a Service(SaaS) deployment, API integration, additional programming languages, and advanced reporting features, enabling controlled and sustainable growth toward a broader international user base.

2.2 Innovation Domains

The innovative aspects of the system can be summarized as follows:

- **Advanced Contextual Analysis using Call Graphs:** The system builds a call graph to provide a deeper understanding of the code, going beyond traditional diff-based analysis and enabling detection of dependencies and indirect side effects across the codebase.
- **Agentic AI-based Multi-Agent Architecture:** The system is based on an Agentic AI paradigm where multiple autonomous agents (semantic and security) operate in .sequential, while a syntax analysis module runs concurrently with them. These agents interact with each other and make partial decisions, enabling a more adaptive and intelligent analysis of the source code.
- **Risk Scoring and Merge Decision Support:** The system provides structured risk levels (Low, Medium, High, Critical) along with explicit merge recommendations, significantly improving decision-making during pull request reviews.

- **End-to-End Workflow Automation:** The entire code review process is fully automated and integrated with platforms such as GitHub and GitLab, ensuring continuous and real-time analysis.
- **Adoption Across Diverse User Segments:** The system is designed for a wide range of users, including students, individual developers, startups, SMEs, and large organizations, ensuring scalability and flexibility across different environments.

3 Third Axis: Strategic Market Analysis

3.1 Market sector overview

A market study is a fundamental tool for adapting to changes in the market in which the institution operates. This study aims to identify potential customers, understand their needs, and position the product effectively in the software development tools market.

3.1.1 PESTEL Analysis

The PESTEL analysis examines the macro-environmental forces that influence the market, essential for anticipating opportunities and threats before market entry.



Figure 35: PESTEL Analysis of the CodeWatchAI

3.1.2 Potential Market and Target Market

Potential Market This section defines who could use the system and who we specifically target as our primary customers.

Who will use our product?

- Software development companies using GitHub for collaborative development.
- Freelance developers and independent teams working on open-source or commercial projects.
- University computer science departments evaluating student code quality.
- DevOps teams maintaining CI/CD pipelines and enforcing coding standards.
- Tech incubators and accelerators supporting early-stage startups.

What motivates them to use it?

- The need to reduce the time senior developers spend on manual code reviews.
- The increasing complexity of codebases and the difficulty of tracking cascading impacts of code changes across interconnected functions.
- The necessity of detecting security vulnerabilities before code reaches production.
- Institutional pressure to maintain consistent coding standards and reduce technical debt.
- The high cost of post-deployment bug fixing compared to catching issues early during Pull Request review.
- The need for 24/7 automated review availability without human bottlenecks.

Where are they located?

- Major Algerian tech hubs: Algiers, Oran, Constantine, Annaba, and Blida.
- Middle East and North Africa(MENA) region markets with growing software ecosystems.
- Globally, any development team using GitHub as their primary version control and collaboration platform.

How many are they?

- **Software Development Companies:** Thousands of registered tech companies and startups across Algeria, growing rapidly under the national digitalization strategy.
- **Freelance Developers:** Tens of thousands of active developers on GitHub from Algeria and the MENA region.
- **Universities:** Approximately 50 public and private universities with computer science and software engineering departments.
- **DevOps Teams:** Hundreds of teams within medium-to-large enterprises managing active GitHub repositories.
- **Tech Incubators:** A growing number of university business incubators and private accelerators supporting tech startups.

Target Market The target market represents the specific segment that will benefit most immediately from the system.

Target Market Segments:

- **Small and Medium Software Development Teams (550 developers):**

  Justification: These teams lack a dedicated senior reviewer yet work at a fast pace where review quality directly impacts product stability.

  Opportunity: A plug-and-play GitHub integration delivering immediate value from the first Pull Request analyzed.

- **Tech Startups in University Incubators:**

  Justification: Early-stage startups need professional code quality tools while keeping operational costs minimal.

  Opportunity: Partnership with CNCSIIU incubators to offer the tool as part of the startup certification package.

- **DevOps and Platform Engineering Teams:**

  Justification: These teams need automated quality gates integrated directly into their CI/CD pipelines.

  Opportunity: Deep integration with GitHub Actions workflows positioning the system as an essential pipeline component.

- **University Computer Science Departments:**

  Justification: Professors need a scalable tool to evaluate student Pull Requests consistently across large cohorts.

  Opportunity: Academic licensing at reduced cost with potential research collaboration opportunities.

Justification for Target Market Selection:

- **High Demand:** All selected segments share the same core pain point fast, reliable code review without increasing human resource costs.
- **Strategic Fit:** The multi-agent architecture directly addresses the technical needs of each identified segment.
- **Market Size:** The combined segments represent a significant and growing portion of the active software development community in Algeria.
- **First-Mover Advantage:** No equivalent tool currently exists in the Algerian market

Potential for Contract Agreements:

- **Software Development Companies:** Multi-year SaaS subscription contracts for unlimited repository analysis.
- **University Incubators:** Institutional agreements within the CNCSIIU startup certification package under Ministerial Decision 1275.
- **DevOps Teams:** Enterprise integration contracts covering custom deployment, dedicated technical support, and full CI/CD pipeline configuration assistance.
- **University Departments:** Academic licensing agreements for semester-based use in software engineering courses and final year project evaluations.

4 Measuring the Level of Competition

This section identifies the main competitors of the AI Code Review system and evaluates their strengths and weaknesses.

4.1 Direct and Indirect Competitors

4.1.1 Direct Competitors

Direct competitors are tools that offer automated, AI-powered code review specifically integrated with GitHub Pull Requests and targeting the same developer audience.

1 CodeRabbit An AI-powered tool that automatically analyzes Pull Requests and provides inline comments on GitHub and GitLab.

Strengths:

- Easy GitHub integration with minimal setup required.
- Provides inline comments directly on Pull Request diffs.
- Supports multiple programming languages out of the box.

Weaknesses:

- No Call Graph for structural impact analysis.
- No specialized security agent.
- Expensive for small teams and startups.
- No risk scoring or Team Lead report.

2 GitHub Copilot Code Review GitHub's native AI assistant that reviews Pull Requests using OpenAI models, backed by Microsoft infrastructure.

Strengths:

- Deep native integration with the GitHub platform and ecosystem.
- Backed by Microsoft and OpenAI infrastructure ensuring high availability and model quality.
- Familiar interface for existing GitHub users requiring no additional tooling.

Weaknesses:

- No Call Graph for impact analysis.
- No multi-agent specialized architecture.
- No Team Lead decision report.
- High cost outside Microsoft enterprise agreements.

4.1.2 Indirect Competitors

Indirect competitors are tools that address related aspects of code quality but do not fully replicate the system's integrated multi-agent review and decision workflow.

1 SonarQube A widely used static code analysis platform detecting bugs and vulnerabilities through rule-based analysis.

Strengths:

- Established reputation and trusted by large enterprises globally.
- Comprehensive rule sets covering multiple programming languages.
- Strong security vulnerability detection based on well-defined patterns.

Weaknesses:

- Does not use LLMs or AI agents ; purely rule-based analysis cannot understand code context, intent, or complex logic.
- Requires complex initial setup and ongoing infrastructure management.
- Does not analyze the functional impact of changes on related functions via a Call Graph.
- Produces no natural language comments or actionable suggestions written for the developer.

2 Snyk Code (DeepCode) An AI-based tool focused on detecting security vulnerabilities in code, integrated with GitHub and CI/CD platforms

Strengths:

- Strong and specialized focus on security and dependency vulnerability detection.
- Integrates with GitHub and common CI/CD pipelines with relative ease.
- Uses machine learning trained on large open-source codebases for pattern recognition.

Weaknesses:

- Security only-does not cover logic or coding standards.
- No Call Graph or Team Lead decision workflow.
- Limited contextual reasoning compared to a full LLM-powered agent.

4.1.3 Our Competitive Advantage

Our system combines three unique capabilities no competitor currently offers:

- **Call Graph Analysis:** Maps all functions impacted by a Pull Request before review begins.
- **Multi-Agent and Module Specialization:** Three dedicated components, a Syntax module, and Semantic and Security agents, independently analyze code from different perspectives.
- **Team Lead Decision Workflow:** A risk-scored final report is delivered to the Team Lead before the Pull Request is merged.

4.2 Marketing strategy

The marketing strategy defines the tools and approaches we use to attract potential customers and motivate them to adopt the system.

- The strategy must take financial capacity into account.
- A balanced marketing mix is essential for success.

4.3 Marketing Mix (7P)

The 7P framework defines how we develop, price, distribute, promote, and deliver the system to our target customers.

4.3.1 Product

The product defines the core value we deliver to customers through the system's capabilities and features.

The AI Code Review system is an intelligent multi-agent platform that integrates directly with GitHub, automatically reviews Pull Requests, and delivers a risk-scored report to the Team Lead.

Key Features:

1. Real-time GitHub integration via webhooks orchestrated through n8n.
2. Call Graph construction to identify all functions impacted by code changes.
3. Two specialized agents and a Module: Syntax, Semantic, and Security.
4. Main Agent consolidating all results into actionable Pull Request comments.
5. Decision Agent generating a risk-scored report for the Team Lead.

Differentiation: Unlike existing tools, the system understands the structural impact of changes across the entire codebase, not just the modified lines.

4.3.2 Price

Pricing is designed to ensure accessibility for all team sizes while generating sustainable revenue from organizational clients. .

- **Free Tier:** For individual developers and open-source projects.
- **Small Teams Plan:** Monthly subscription for small development teams.
- **Universities and Enterprises:** Institutional licensing for university departments and companies.
- **Individual Plan:** Plan for individual developers and freelancers.

4.3.3 Place

Distribution is carried out through digital channels that align naturally with the workflows of our target developer audience.

- **GitHub Marketplace:** Primary listing to reach developers directly within their existing workflow.
- **Direct Sales:** Targeting software companies, DevOps teams, and university departments through outreach and demonstrations.
- **University Incubator Network:** Distribution through the CNCSIIU network as part of the startup support package.
- **Dedicated Online Platform:** A product website with documentation, demo environment, and self-service subscription management.
- **Geographic Focus:** Initially Algeria, with planned expansion into Tunisia, Morocco, and the MENA region.

4.3.4 Promotion

Promotion is focused on building credibility and visibility within the developer community through technical and community-driven channels.

- **Open Source Community Building:** Publishing core components on GitHub to attract developer interest and organic visibility.

- **Technical Content Marketing:** Publishing articles and case studies on Dev.to, Hashnode, and LinkedIn.
- **Developer Events and Hackathons:** Participating in Algerian and regional hackathons and tech meetups.
- **University Partnerships:** Collaborating with computer science departments to integrate the tool into software engineering courses.
- **Social Media Engagement:** Collaborating with computer science departments to integrate the tool into software engineering courses.

4.3.5 People

The people behind the system directly shape the customer experience and build the trust needed for adoption.

1. Development team of specialized students in Networks and Distributed Systems.
2. Developer-first support mindset ensuring early adopter feedback shapes the product.
3. University supervisors and incubator mentors providing credibility with institutional clients.

4.3.6 Process

The service delivery process is designed to be fully automated and seamless from the moment a team installs the system.

1. Developer opens a Pull Request on GitHub.
2. GitHub sends a webhook notification to n8n.
3. The Context Module uses the pre-built Call Graph to identify all impacted functions.
4. The analysis tools analyze the affected code sequentially.
5. The Main Agent consolidates all results into Pull Request comments.
6. The Decision Agent generates the final risk report and delivers it to the Team Lead.

4.3.7 Physical Evidence

Physical evidence consists of all visible outputs that customers can reference to evaluate the quality and reliability of the system.

- **Pull Request Comments:** AI-generated comments appearing directly on our system's website.
- **Risk Score Reports:** Structured reports generated by the Decision Agent for the Team Lead.
- **Dashboard and Analytics:** A web-based dashboard showing review history, risk trends, and agent performance metrics.
- **Brand and Documentation:** A professional product website and comprehensive technical documentation reinforcing product credibility.

4.4 SWOT Analysis

The SWOT analysis evaluates internal strengths and weaknesses alongside external opportunities and threats.



Figure 36: SWOT Analysis of the Proposed System [AIGC 1]

5 Fourth Axis: Production and Organizational Plan

5.1 Production process

The system follows a fully automated pipeline triggered by developer activity on GitHub:

- **Event Triggering:** When a developer opens a pull request, GitHub automatically fires a webhook event that initiates the entire review pipeline.
- **Workflow Orchestration:** n8n receives the webhook and coordinates all agents, API calls, and execution flow.
- **Syntax Analysis:** GitHub Actions runs flake8 on the code, providing 100% accurate syntax error detection.
- **AI Multi-Agent Analysis:** Three sequential LLM agents perform semantic analysis, security vulnerability detection, and report consolidation. A final Decision Agent produces the accept or reject verdict.
- **Call Graph Construction:** A dedicated Django view extracts function definitions and call relationships from the source files, enabling impact analysis on the codebase.
- **Dashboard & Storage:** All results are stored in PostgreSQL and displayed on a Django management dashboard where the team lead reviews findings and makes the final decision.
- **Deployment:** The entire system runs in Docker Compose containers and is fully reproducible with a single command.

5.2 Supply Chain / Procurement

The system relies primarily on open-source and cloud-based tools, which significantly reduces procurement complexity and cost. The key dependencies are:

Resource	Type	Source	Cost
GitHub API & Webhooks	Cloud service	GitHub	Free tier available, paid plans for advanced features
GitLab API & Webhooks	Cloud service	GitLab	Free tier available, paid plans for advanced features
n8n	Open-source / Self-hosted	n8n.io	Free for individual use, whether self-hosted or cloud-hosted, a paid subscription is required for team or enterprise usage.
LLM API (Claude / Gemini / GPT)	Cloud API	Anthropic / Google / OpenAI	Limited free tier (daily token quota often insufficient for production use); paid subscription required for reliable usage
flake8	Open-source library	PyPI	Free
Django	Open-source framework	PyPI	Free
PostgreSQL	Open-source database	PostgreSQL.org	Free
Docker Compose	Open-source tool	Docker Inc.	Free

Several important cost considerations must be highlighted:

- **n8n:** In our current setup, n8n is self-hosted locally, which is free for individual use but requires ngrok to expose local webhooks to the internet. For a production deployment, migrating to n8n cloud would eliminate the need for ngrok and improve reliability, but would require a paid subscription once used beyond individual scope (e.g., for a team or enterprise).
- **LLM API:** Most major LLM providers (Anthropic, Google, OpenAI) offer only a limited free daily token quota, which is generally insufficient for continuous production use across multiple repositories and pull requests. A paid API plan would be necessary for any serious deployment.
- **GitHub vs GitLab:** Both platforms offer free tiers with webhook and API access sufficient for basic usage. However, advanced features such as fine-grained access control or enterprise integrations require a paid plan on either platform.

5.3 Workforce

Our project currently requires a small, technically skilled team during the development phase. However, for a future commercial deployment, a broader team covering technical, commercial, and financial functions would be necessary:

- **Development phase (current):**The team of 3 engineers covers all technical roles: backend development, AI agent design, workflow orchestration, and frontend dashboard development. The system itself requires no manual workforce during operation, as the entire review pipeline runs automatically from webhook trigger to final verdict.
- **Commercial deployment phase:**A complete team would need to include the following profiles:

Profile	Role
Backend / DevOps Engineer	Maintain infrastructure, manage deployments and containers
AI Engineer	Monitor and improve agent performance and accuracy
Frontend Developer	Maintain and evolve the management dashboard
Marketing & Sales	Client acquisition and partnerships
Financial Officer	Manage budgeting, revenue tracking, and financial reporting

Table 12: Project Team Roles

The highly automated nature of the system keeps operational workforce lean on the technical side, but building a sustainable commercial product requires investment in business, marketing, and financial expertise as well.

5.4 Key Partnerships

The project relies on key partners that support its development and scalability through technical resources, academic support, and practical validation. These include technology providers offering Large Language Model (LLM) APIs, cloud computing services, and database systems for efficient data processing. The system also uses open-source workflow orchestration tools to automate operations. Academic support is provided by the business incubator of the University of 20 August 1955, Skikda, along with research laboratories that contribute to system improvement. Public institutions, innovation support organizations, and potential investors may offer funding and strategic support, while collaboration with technical bodies ensures compliance with quality and cybersecurity standards. The project also involves partnerships with innovation support agencies and local software development companies to test the system and gather feedback, with future opportunities for global collaboration. Finally, the system integrates with platforms such as GitHub and GitLab, enabling automated code review within real development environments, supporting its transition to a scalable, market-ready solution.

6 Fifth Axis: Financial Plan

6.1 Costs and Expenses

6.1.1 Capital Expenditure (CapEx)

CAPEX (Dépenses d'Investissement)			
En Dinars Algériens (DA)			
Désignation	**Quantité**	**Coût unitaire**	**Total**
Équipements informatiques			
Ordinateurs — PC développeurs	3	200,000	600,000
Ordinateurs — PC bureau	2	100,000	200,000
Sous-total Équipements informatiques			**800,000**
Mobilier de bureau			
Chaises de bureau	5	17,000	85000
Bureaux	5	13,000	65000
Sous-total Mobilier de bureau			**150,000**
Équipements du local			
Caméra de surveillance	1	25,000	25000
Installation caméra	1	6,000	6000
Équipements divers du local	-	30000	30000
Sous-total Équipements du local			**61,000**
Immobilisation incorporelle			
Développement logiciel	-	3600000	3600000
Sous-total Immobilisation incorporelle			**3,600,000**
Frais de création d'entreprise			
Dénomination sociale	1	800	800
Statuts juridiques notariés	1	15,000	15,000
Capital social initial (20%)	1	20,000	20,000
Domiciliation du siège social	1	60,000	60,000
Publication au Journal Officiel	1	6,110	6,110
Timbre fiscal	1	4,000	4,000
Frais d'enregistrement	1	10,000	10,000
CASNOS (première inscription)	1	34,000	34,000
Ouverture de compte bancaire	1	15,000	15,000
Sous-total Frais de création			**164,910**
TOTAL CAPEX (hors développement logiciel)			**1,175,910**
TOTAL CAPEX GLOBAL (avec logiciel)			**4,775,910**

Figure 37: Capital Expenditure (CapEx) Breakdown of the CodeWatchAI Project

6.1.2 Operational Expenditure (OpEx)

OPEX DÉPENSES D'EXPLOITATION PRÉVISIONNELLES							
En Dinars Algériens (DA)							
Désignation	**Mensuel**	**N**	**N+1**	**N+2**	**N+3**	**N+4**	**N+5**
1. Charges technologiques							
Abonnement n8n (cloud)	6,600	79,200	79200	79200	79200	79200	79200
Hébergement VPS	2,000	24,000	24000	24000	24000	24000	24000
API LLM	20,000	240,000	240000	240000	240000	240000	240000
Internet	2,200	26,400	26400	26400	26400	26400	26400
➤ Sous-total Charges technologiques		**369,600**	**369,600**	**369,600**	**369,600**	**369,600**	**369,600**
2. Charges fixes							
Loyer du local	30,000	360,000	360,000	360,000	360,000	360,000	360,000
Électricité	2,000	8,000	8,000	8,000	8,000	8,000	8,000
➤ Sous-total Charges fixes		**368,000**	**368,000**	**368,000**	**368,000**	**368,000**	**368,000**
3. Charges fiscales et sociales obligatoires							
Impôts et taxes annuels	-	32,000	32,000	32,000	32,000	32,000	32,000
CASNOS (cotisation annuelle)	-	42,000	42,000	42,000	42,000	42,000	42,000
➤ Sous-total Charges fiscales		**74,000**	**74,000**	**74,000**	**74,000**	**74,000**	**74,000**
4. Charges commerciales							
Budget publicité	-	10,000	10,000	10,000	10,000	10,000	10,000
➤ Sous-total Charges commerciales		**10,000**	**10,000**	**10,000**	**10,000**	**10,000**	**10,000**
5. Charges de personnel							
Salaire brut (50 K DZD/mois × nb salariés)	50,000	-	600,000	1,200,000	1,800,000	2,400,000	3,000,000
CNAS employeur (25 % du salaire brut)	-	-	150,000	300,000	450,000	600,000	750,000
Avantages (transport, repas, mutuelle…)	30000	-	360,000	720,000	1,080,000	1,440,000	1,800,000
➤ Sous-total Charges de personnel		-	**1,110,000**	**2,220,000**	**3,330,000**	**4,440,000**	**5,550,000**
TOTAL OPEX ANNUEL		**821,600**	**1,931,600**	**3,041,600**	**4,151,600**	**5,261,600**	**6,371,600**

Figure 38: Operational Expenditure (OpEx) Forecast

6.1.3 Balance Sheet

BILAN PRÉVISIONNEL								
En Dinars Algériens (DA)								
Rubriques	**N-2**	**N-1**	**N**	**N+1**	**N+2**	**N+3**	**N+4**	**N+5**
ACTIF								
Actif non courant — Immobilisations								
Immobilisations Incorporelles								
Software	-	-	240,000	120,000	0	0	0	0
Frais d'établissement								
Total Frais d'établissement			131,928	98,946	65,964	32,982	0	
Immobilisations Corporelles								
Ordinateurs PC développeurs	-	-	400,000	200,000	0	0	0	0
Ordinateurs PC bureau	-	-	133,333	66,667	0	0	0	0
Chaises de bureau	-	-	76,500	68,000	59,500	51,000	42,500	34,000
Bureaux	-	-	58,500	52,000	45,500	39,000	32,500	26,000
Caméra de surveillance + installation	-	-	24,800	18,600	12,400	6,200	0	0
Équipements divers du local	-	-	24,000	18,000	12,000	6,000	0	0
ACTIF NON COURANT	**-**	**-**	**1,089,061**	**642,213**	**195,364**	**135,182**	**75,000**	**60,000**
Actif courant								
Clients (créances)	-	-	0	0	0	0	0	0
Trésorerie	-	-	2,960,490	2,418,890	2,087,290	2,238,500	2,690,010	5,233,228
ACTIF COURANT	**-**	**-**	**2,960,490**	**2,418,890**	**2,087,290**	**2,238,500**	**2,690,010**	**5,233,228**
TOTAL ACTIF	**-**	**-**	**4,049,551**	**3,061,103**	**2,282,654**	**2,373,682**	**2,765,010**	**5,293,228**
PASSIF								
Capitaux propres								
Capital émis (Fondateurs)	-	-	20,000	20,000	20,000	20,000	20,000	20,000
Primes et réserves légales (5% bén.)	-	-	0	0	0	0	4,551	24,118
Résultat net de l'exercice	-	-	-658,449	-988,449	-778,449	91,028	391,328	2,528,218
Autres capitaux propres- report à nouveau	-	-	4,656,000	3,997,551	3,009,102	2,230,653	2,317,130	2,688,891
CAPITAUX PROPRES	**-**	**-**	**4,017,551**	**3,029,102**	**2,250,653**	**2,341,681**	**2,733,009**	**5,261,227**
Passifs non courants								
Emprunts et dettes financières	-	-	0	0	0	0	0	0
Impôt différé passif	-	-	0	0	0	0	0	0
Compte courant associé — CCA	-	-	0	0	0	0	0	0
PASSIFS NON COURANTS	**-**	**-**	**0**	**0**	**0**	**0**	**0**	**0**
Passifs courants								
Impôts	-	-	32,000	32,000	32,000	32,000	32,000	32,000
Fournisseurs et comptes rattachés	-	-	0	0	0	0	0	0
Trésorerie passif (découvert bancaire)	-	-	0	0	0	0	0	0
PASSIFS COURANTS	**-**	**-**	**32,000**	**32,000**	**32,000**	**32,000**	**32,000**	**32,000**
TOTAL PASSIF	**-**	**-**	**4,049,551**	**3,061,102**	**2,282,653**	**2,373,681**	**2,765,009**	**5,293,227**

Figure 39: Five-Year Forecast Balance Sheet of the CodeWatchAI Startup Project

6.2 Revenue

TABLEAU DU CHIFFRE D'AFFAIRES PRÉVISIONNEL								
En Dinars Algériens (DA)								
RUBRIQUES	**N-2**	**N-1**	**N**	**N+1**	**N+2**	**N+3**	**N+4**	**N+5**
1. ABONNEMENTS PETITES ÉQUIPES (Small Teams)								
Nombre de clients (Small Teams)	-	-	4	8	12	18	25	35
Prix mensuel unitaire HT	-	-	5,000	5,000	5,000	5,000	5,000	5,000
Mois de facturation	-	-	12	12	12	12	12	12
SOUS-TOTAL Small Teams	-	-	**240,000**	**480,000**	**720,000**	**1,080,000**	**1,500,000**	**2,100,000**
2. UNIVERSITÉS & ENTREPRISES								
Nombre de clients (Mid Market)	-	-	1	2	4	6	8	10
Prix mensuel unitaire HT	-	-	15,000	15,000	15,000	15,000	15,000	15,000
Mois de facturation	-	-	12	12	12	12	12	12
SOUS-TOTAL Universités & Entreprises	-	-	**180,000**	**360,000**	**720,000**	**1,080,000**	**1,440,000**	**1,800,000**
3. ABONNEMENTS INDIVIDUELS								
Nombre de clients (Individuels)	-	-	5	15	35	60	80	160
Prix mensuel unitaire HT	-	-	3,000	3,000	3,000	3,000	3,000	3,000
Mois de facturation	-	-	12	12	12	12	12	12
SOUS-TOTAL Individuels	-	-	**180,000**	**540,000**	**1,260,000**	**2,160,000**	**2,880,000**	**5,760,000**
CHIFFRE D'AFFAIRES GLOBAL HT	-	-	**600,000**	**1,380,000**	**2,700,000**	**4,320,000**	**5,820,000**	**9,660,000**

Figure 40: Projected Revenue Growth for the Period N to N+5

6.3 Projected Income Statement

COMPTE DE RÉSULTAT PRÉVISIONNEL (TCR)								
En Dinars Algériens (DA)								
Rubriques	**N-2**	**N-1**	**N**	**N+1**	**N+2**	**N+3**	**N+4**	**N+5**
PRODUITS D'EXPLOITATION								
Ventes et produits annexes	-	-	600,000.00	1,380,000.00	2,700,000.00	4,320,000.00	5,820,000.00	9,660,000.00
PRODUCTION DE L'EXERCICE	-	-	**600,000.00**	**1,380,000.00**	**2,700,000.00**	**4,320,000.00**	**5,820,000.00**	**9,660,000.00**
CHARGES D'EXPLOITATION								
Services extérieurs et autres consommations								
Abonnement n8n (cloud)	-	-	79,200.00	79,200.00	79,200.00	79,200.00	79,200.00	79,200.00
Hébergement VPS	-	-	24,000.00	24,000.00	24,000.00	24,000.00	24,000.00	24,000.00
API LLM	-	-	240,000.00	240,000.00	240,000.00	240,000.00	240,000.00	240,000.00
Internet	-	-	26,400.00	26,400.00	26,400.00	26,400.00	26,400.00	26,400.00
Électricité	-	-	8,000.00	8,000.00	8,000.00	8,000.00	8,000.00	8,000.00
Loyer du local	-	-	360,000.00	360,000.00	360,000.00	360,000.00	360,000.00	360,000.00
Consommation de l'exercice	-	-	**737,600.00**	**737,600.00**	**737,600.00**	**737,600.00**	**737,600.00**	**737,600.00**
Valeur ajoutée d'exploitation	-	-	**-137,600.00**	**642,400.00**	**1,962,400.00**	**3,582,400.00**	**5,082,400.00**	**8,922,400.00**
Charges de personnel								
Total charges de personnel	-	-	**0.00**	**1,110,000.00**	**2,220,000.00**	**3,330,000.00**	**4,440,000.00**	**5,550,000.00**
Impôts et taxes								
Impôts et taxes annuels	-	-	32,000.00	32,000.00	32,000.00	32,000.00	32,000.00	32,000.00
CASNOS (cotisation annuelle)	-	-	42,000.00	42,000.00	42,000.00	42,000.00	42,000.00	42,000.00
Total impôts et taxes	-	-	**74,000.00**	**74,000.00**	**74,000.00**	**74,000.00**	**74,000.00**	**74,000.00**
EXCÉDENT BRUT D'EXPLOITATION (EBE)	-	-	**-211,600.00**	**-541,600.00**	**-331,600.00**	**178,400.00**	**568,400.00**	**3,298,400.00**
Dotations aux amortissements	-	-	446,849.00	446,849.00	446,849.00	60,182.00	60,182.00	15,000.00
Résultat opérationnel	-	-	**-658,449.00**	**-988,449.00**	**-778,449.00**	**118,218.00**	**508,218.00**	**3,283,400.00**
Produits financiers	-	-	0.00	0.00	0.00	0.00	0.00	0.00
Charges financières	-	-	0.00	0.00	0.00	0.00	0.00	0.00
Résultat financier	-	-	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
Résultat ordinaire avant impôt	-	-	**-658,449.00**	**-988,449.00**	**-778,449.00**	**118,218.00**	**508,218.00**	**3,283,400.00**
IBS (23% si résultat > 0)	-	-	0.00	0.00	0.00	27,190.14	116,890.14	755,182.00
RÉSULTAT NET DE L'EXERCICE	-	-	**-658,449.00**	**-988,449.00**	**-778,449.00**	**91,027.86**	**391,327.86**	**2,528,218.00**
Total produits des activités ordinaires	-	-	**600,000.00**	**1,380,000.00**	**2,700,000.00**	**4,320,000.00**	**5,820,000.00**	**9,660,000.00**
Total charges des activités ordinaires	-	-	**1,258,449.00**	**2,368,449.00**	**3,478,449.00**	**4,228,972.14**	**5,428,672.14**	**7,131,782.00**

Figure 41: Projected Income Statement for the Period N to N+5

6.4 Cash Flow Plan

TABLEAU DES FLUX DE TRÉSORERIE PRÉVISIONNEL (TFT)

En Dinars Algériens (DA)

Rubriques	N-2	N-1	N	N+1	N+2	N+3	N+4	N+5
A — Flux provenant des activités opérationnelles								
Résultat net de l'exercice	-	-	-658,449.00	-988,449.00	-778,449.00	91,028.00	391,328.00	2,528,218.00
+ Amortissements et provisions	-	-	446,849.00	446,849.00	446,849.00	60,182.00	60,182.00	15,000.00
± Variation des impôts différés	-	-	0.00	0.00	0.00	0.00	0.00	0.00
± Variation des stocks	-	-	0.00	0.00	0.00	0.00	0.00	0.00
± Variation des clients et créances	-	-	0.00	0.00	0.00	0.00	0.00	0.00
± Variation des fournisseurs et dettes	-	-	0.00	0.00	0.00	0.00	0.00	0.00
▶ Flux de trésorerie opérationnels (A)	-	-	-211,600.00	-541,600.00	-331,600.00	151,210.00	451,510.00	2,543,218.00
B — Flux provenant des opérations d'investissement								
Décaissements sur acquisition d'immobilisations	-	-	-1,175,910.00	0.00	0.00	0.00	0.00	0.00
Encaissements sur cessions d'immobilisations	-	-	0.00	0.00	0.00	0.00	0.00	0.00
▶ Flux de trésorerie liés aux investissements (B)	-	-	-1,175,910.00	0.00	0.00	0.00	0.00	0.00
C — Flux provenant des opérations de financement								
Dividendes versés aux actionnaires	-	-	0.00	0.00	0.00	0.00	0.00	0.00
Augmentation de capital — Investisseur	-	-	4,328,000.00	0.00	0.00	0.00	0.00	0.00
Augmentation de capital — Fondateurs	-	-	20,000.00	0.00	0.00	0.00	0.00	0.00
Injection en compte courant associé (CCA)	-	-	0.00	0.00	0.00	0.00	0.00	0.00
Remboursements CCA	-	-	0.00	0.00	0.00	0.00	0.00	0.00
▶ Flux de trésorerie liés au financement (C)	-	-	4,348,000.00	0.00	0.00	0.00	0.00	0.00
★ Variation de trésorerie de la période (A+B+C)	-	-	2,960,490.00	-541,600.00	-331,600.00	151,210.00	451,510.00	2,543,218.00
Trésorerie d'ouverture (début de période)	-	-	0.00	2,960,490.00	2,418,890.00	2,087,290.00	2,238,500.00	2,690,010.00
★ Trésorerie de clôture (fin de période)	-	-	2,960,490.00	2,418,890.00	2,087,290.00	2,238,500.00	2,690,010.00	5,233,228.00

Figure 42: Projected Cash Flow Plan for the Period N to N+5

6.5 Business Model Canvas (BMC)

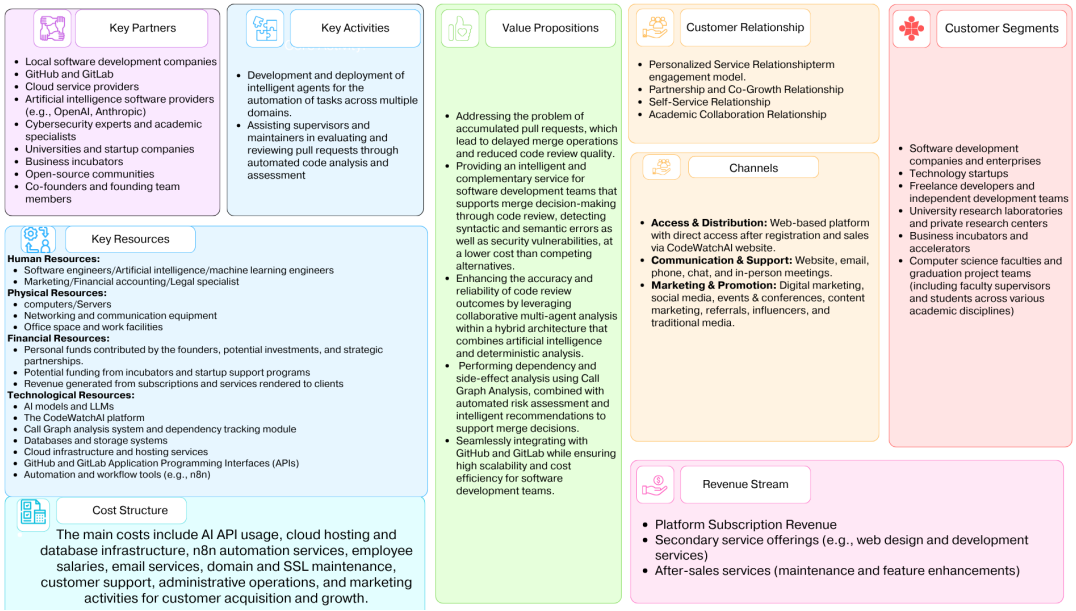


Figure 43: Business Model Canvas (BMC) of the CodeWatchAI Startup Project

7 Sixth Axis: Prototype

7.1 User Onboarding Interfaces

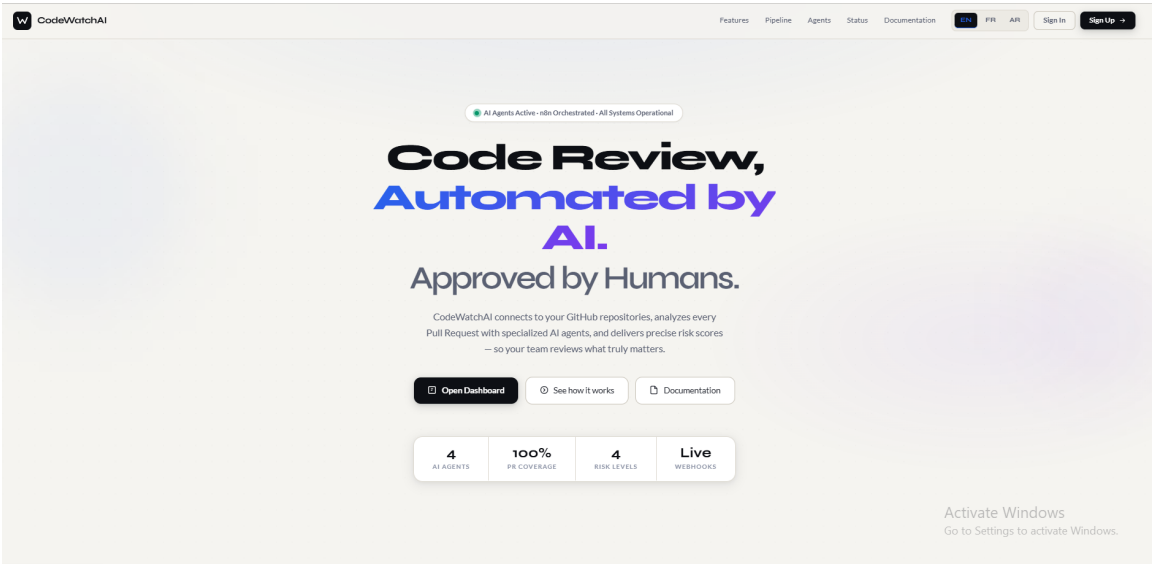


Figure 44: CodeWatchAI Official Landing Page-Platform Overview and Value Proposition

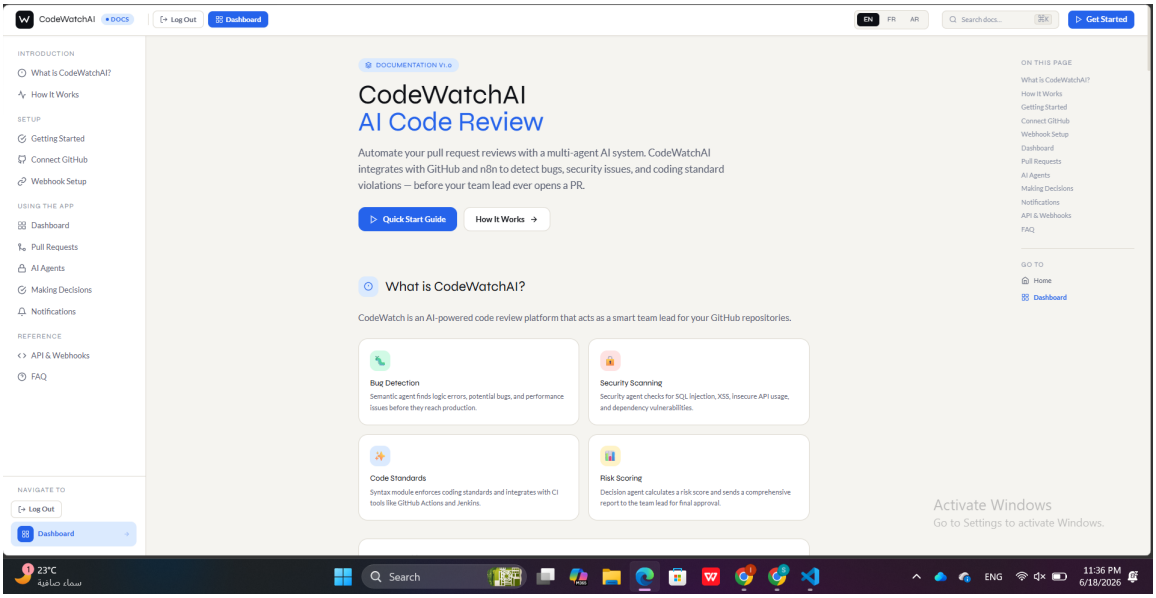


Figure 45: System Documentation Page-User Guides and AI Agents Overview

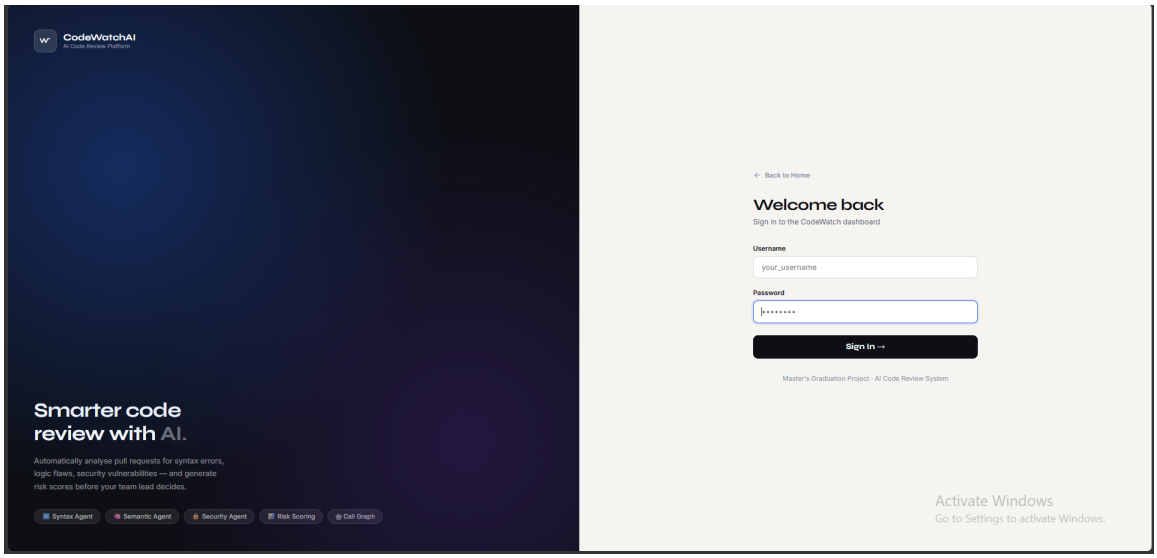


Figure 46: User Sign-In Interface-Secure Access to the Dashboard

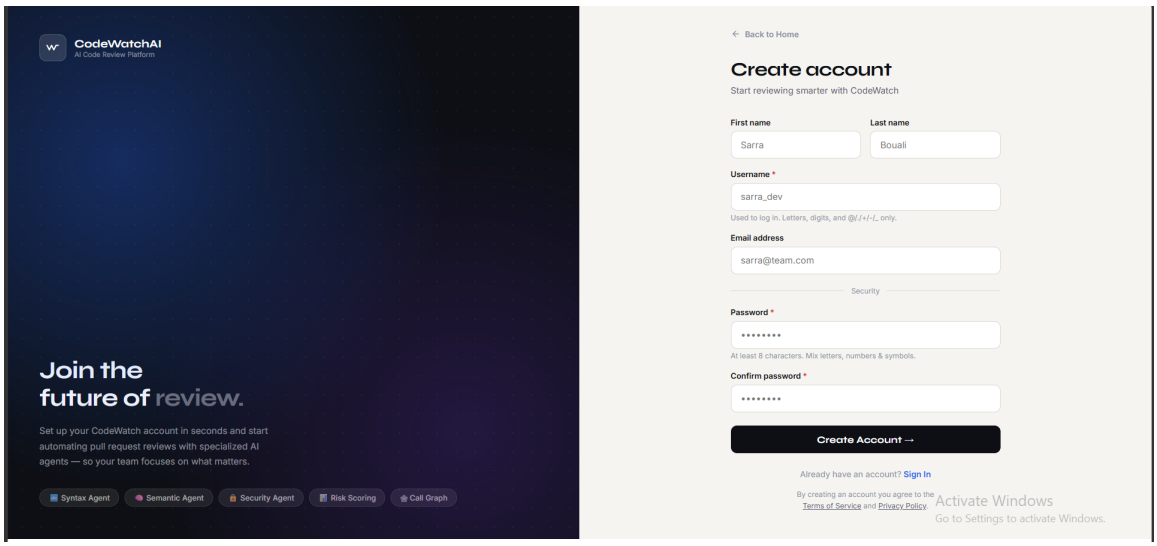


Figure 47: Account Creation Form- Registering a New Developer Profile

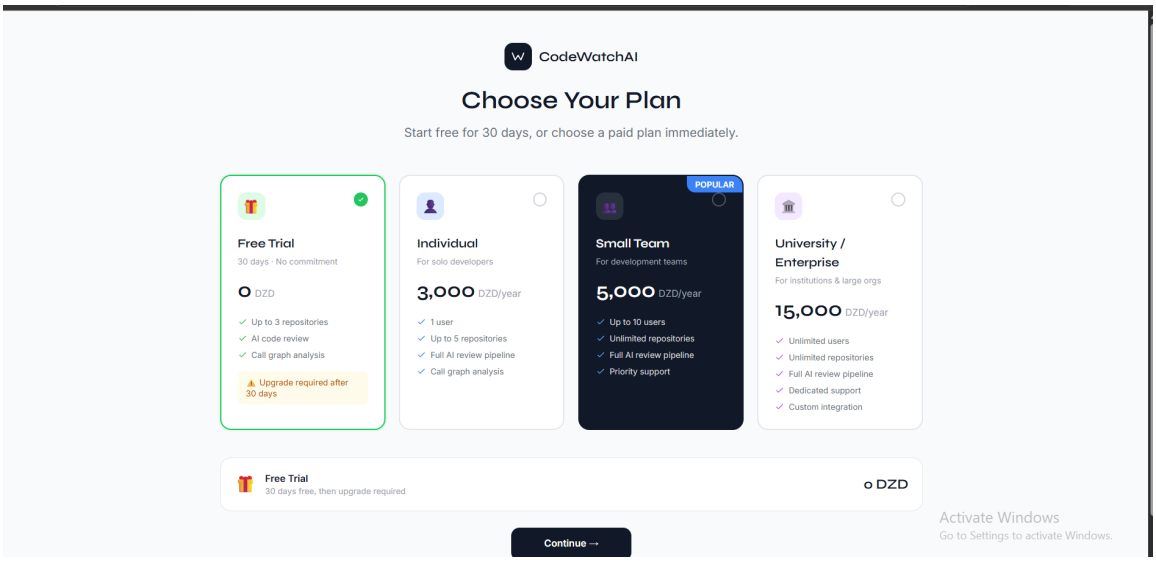


Figure 48: Subscription Plans Selection Page-Pricing and Feature Comparison Matrix

7.2 System Interfaces and Dashboard

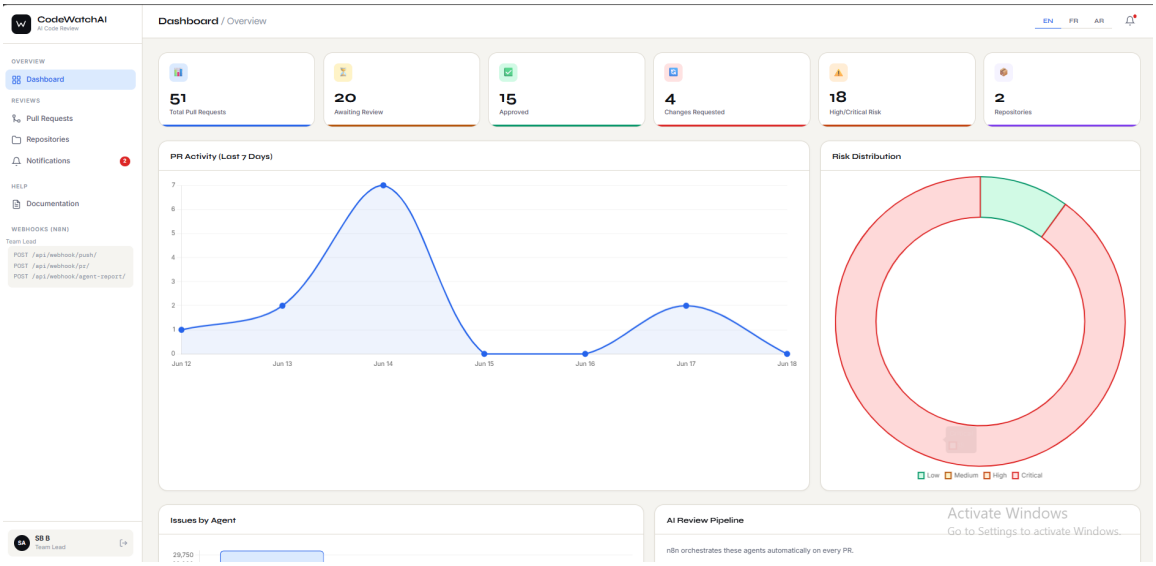


Figure 49: Main Dashboard Overview-Metrics Cards, Weekly PR Activity Graph, and Risk Distribution Chart

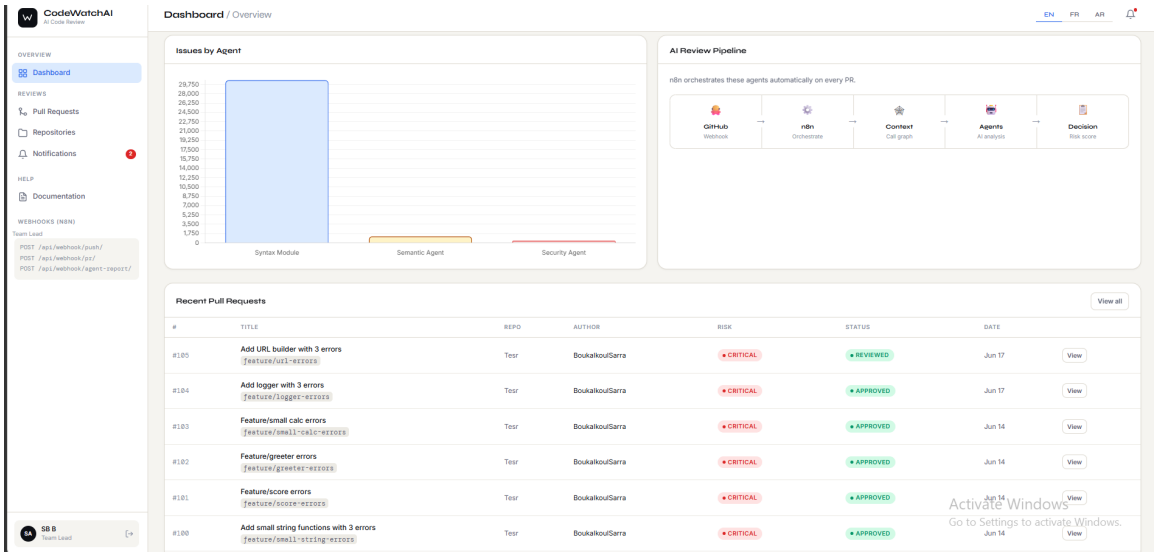


Figure 50: Dashboard Analytics-Issues Detected by AI Agents and Recent Pull Requests Table

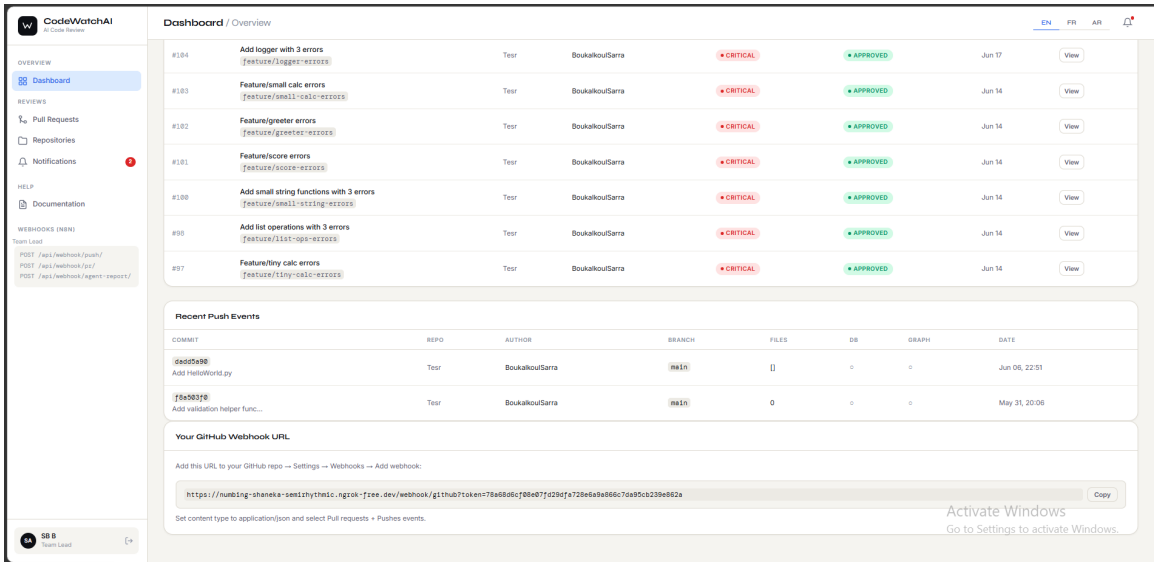


Figure 51: Repository Tracking Interface-Recent Push Events and GitHub Webhook Configuration URL

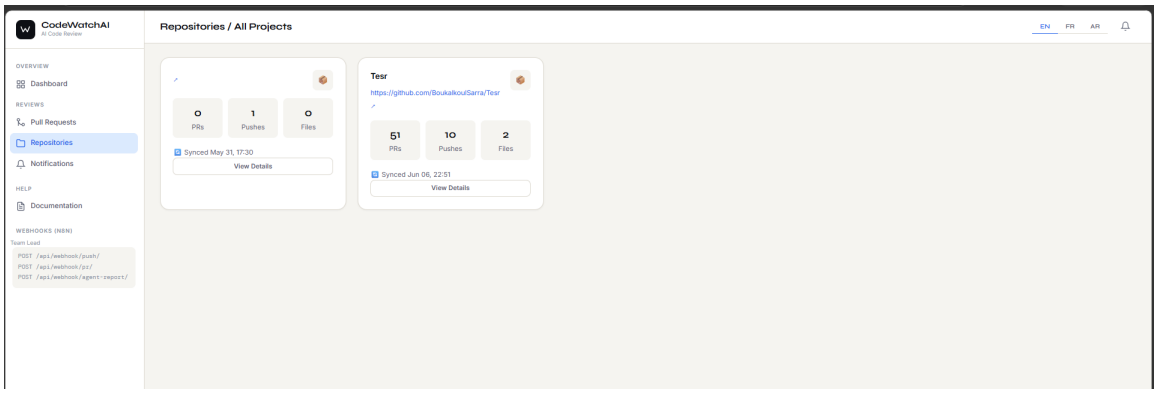


Figure 52: Repositories Management Portal-Overview of Synced Projects and Activity Summary

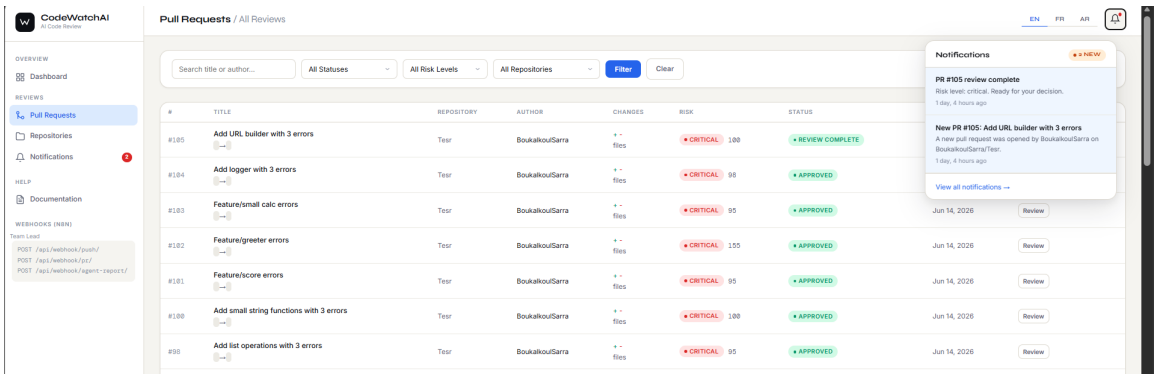


Figure 53: Pull Requests Page with Active Notifications Dropdown Menu Overlay

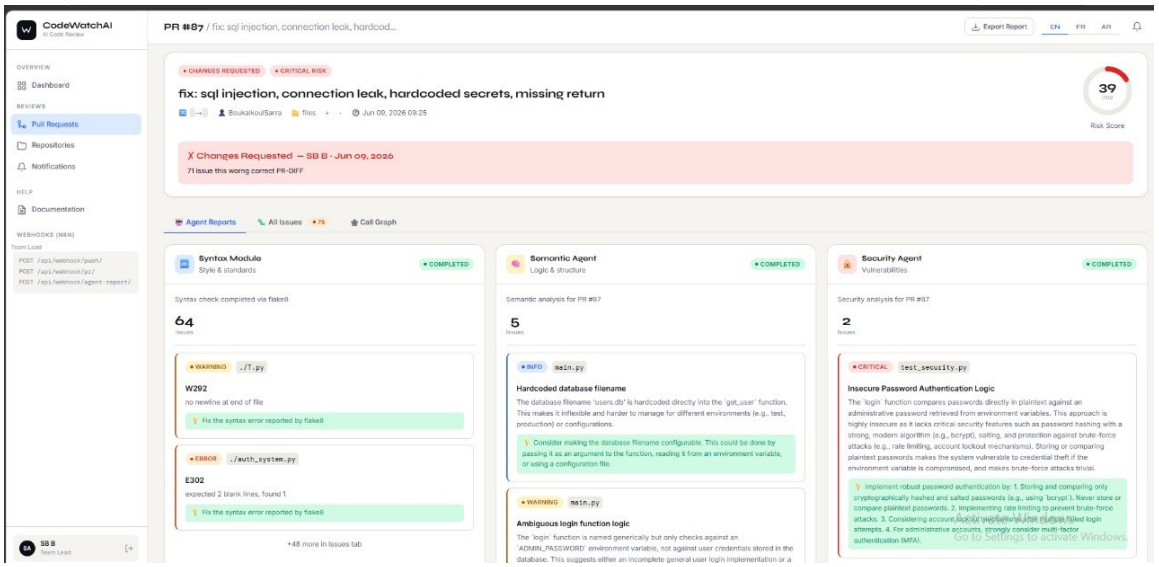


Figure 54: Pull Request Review Page-Syntax module and Semantic Agent and Security Agent Reports

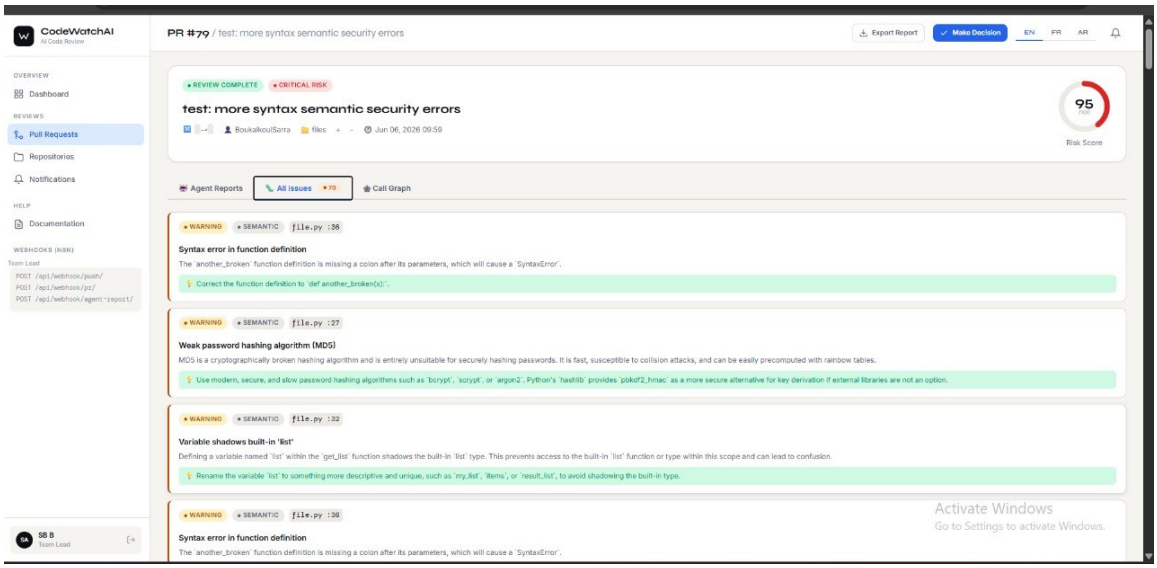


Figure 55: Issues List Page-Detailed View of All Detected Problems

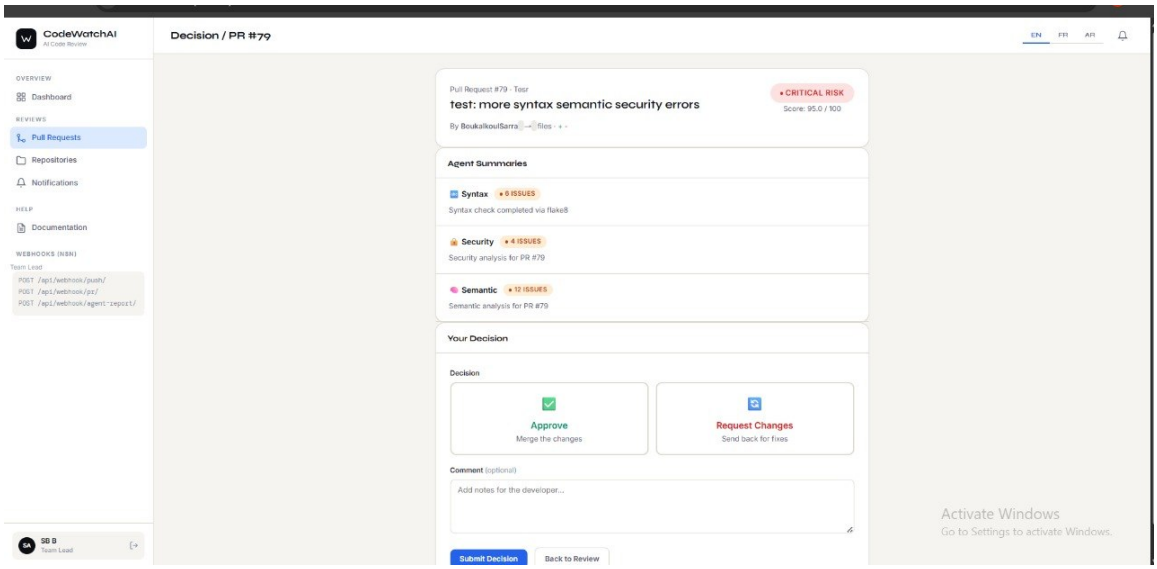


Figure 56: Decision-Making Interface- Approve or Request Changes for a Pull Request

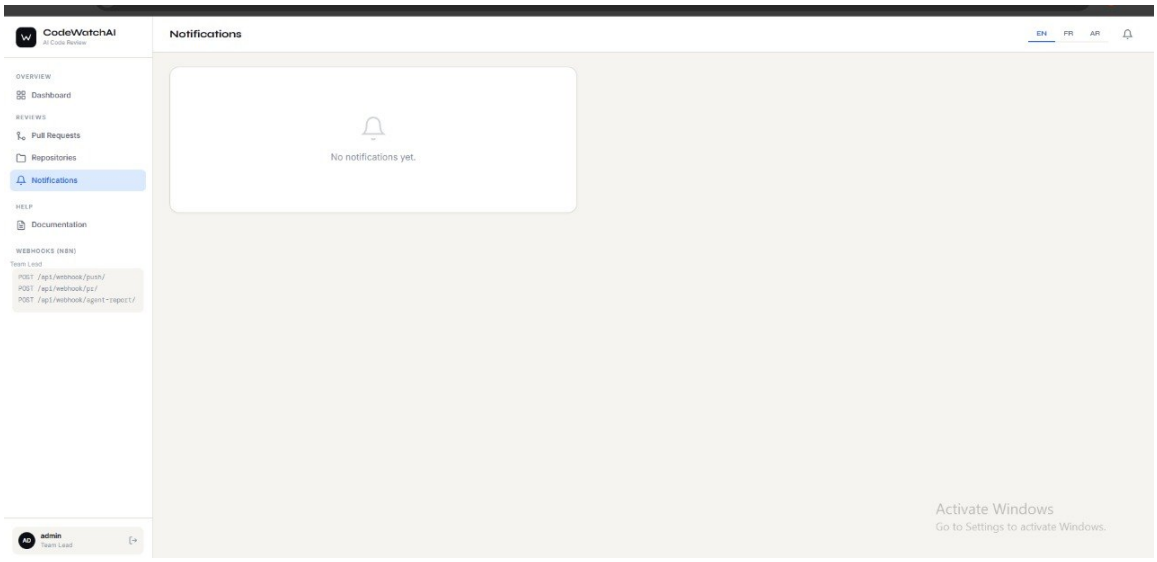


Figure 57: Notifications Page-Empty State Before Receiving Data

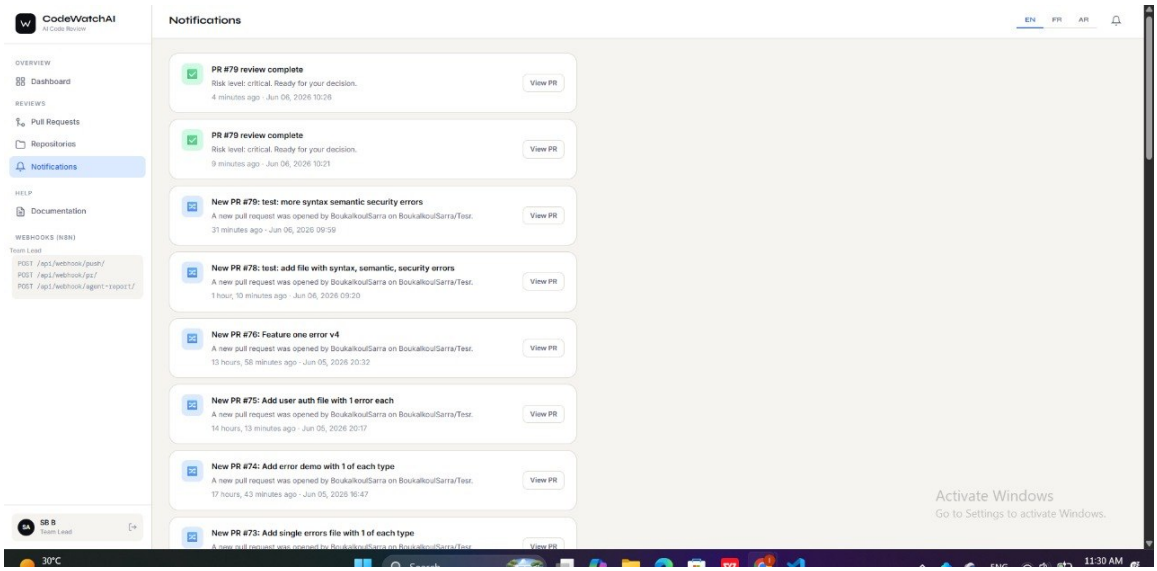


Figure 58: System Notifications Centre-Archive View of Historical Repository Actions and Past Review Statuses

7.3 Subscription Payment and Checkout Workflow

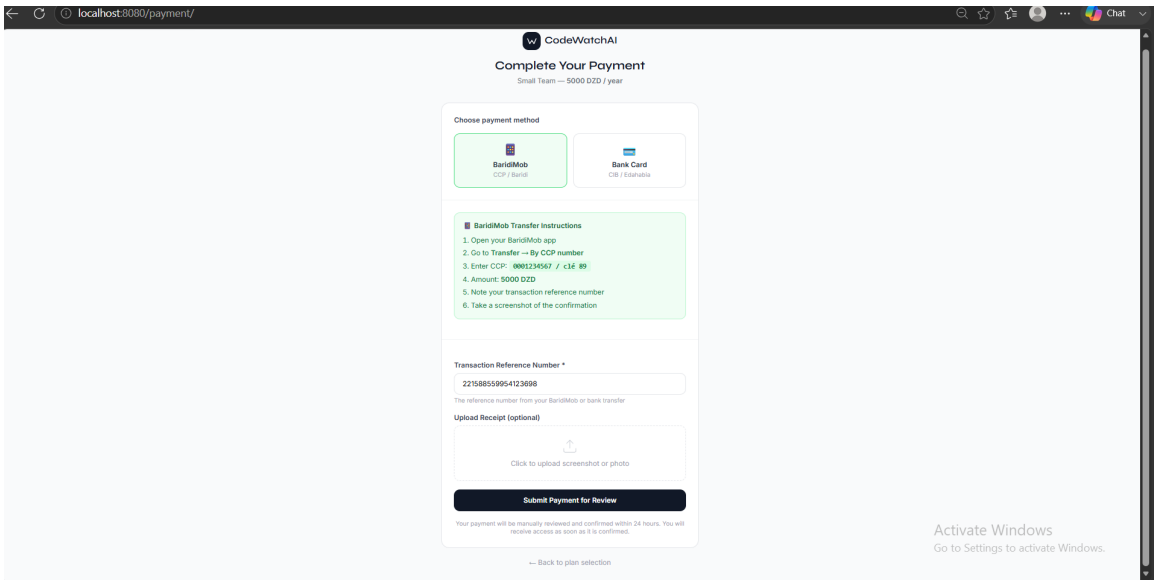


Figure 59: Payment Checkout Portal-Submitting Transaction Reference Number and Uploading Receipt

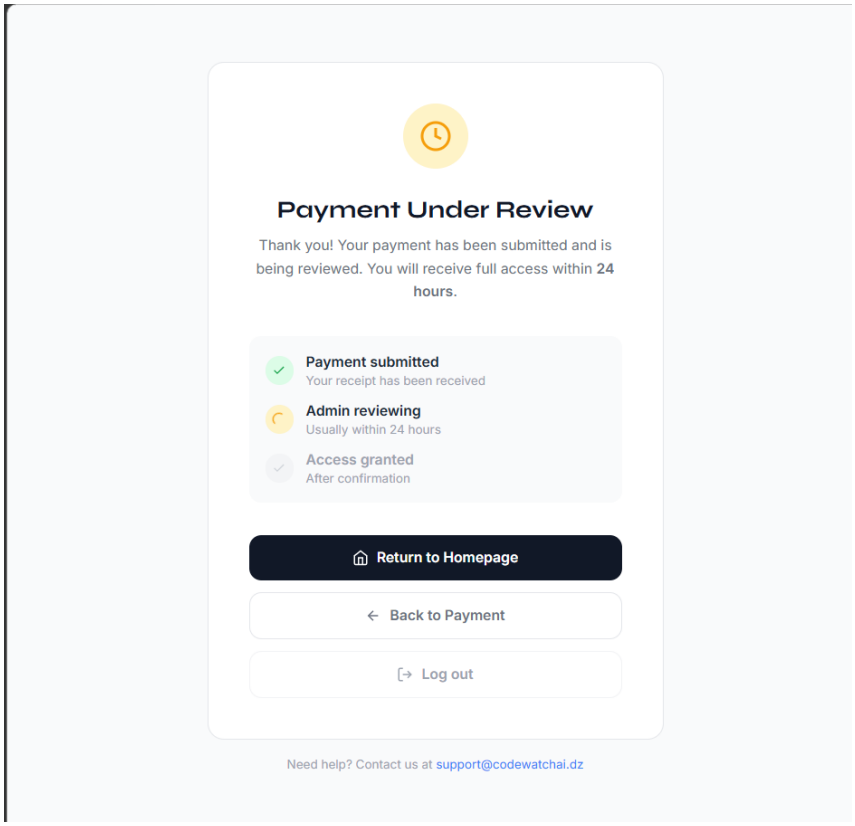


Figure 60: Payment Under Review Page-Tracking Verification Progress and System Administrative Approval

Bibliography

[1] J. McCarthy, M. L. Minsky, N. Rochester, and C. E. Shannon, "A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence, August 31, 1955," *AI Magazine*, vol. 27, no. 4, pp. 12–14, 2006. (Originally written in 1955, reprinted in 2006)

[2] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed. Hoboken, NJ, USA: Pearson, 2021.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[4] P. H. Winston, *Artificial Intelligence*, 3rd ed. Reading, MA, USA: Addison-Wesley, 1992.

[5] A. Vaswani et al., "Attention Is All You Need," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017, pp. 5998–6008.

[6] T. B. Brown et al., "Language Models are Few-Shot Learners," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020, pp. 1877–1901.

[7] A. V. Arunkumar, G. R. Gangadharan, and R. Buyya, *Agentic Artificial Intelligence (AI): Architectures, Taxonomies, and Evaluation of Large Language Model Agents*, arXiv preprint arXiv:2601.12560, 2026.

[8] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[9] M. Adawadkar, *The Evolution of Generative and Agentic AI: From Rule-Based Systems to Autonomous Intelligence*, American Journal of Technology Advancement, vol. 2, no. 11, pp. 1–11, 2025.

[10] E. A. Feigenbaum, *Expert Systems: Principles and Practice*, in *The Encyclopedia of Computer Science and Engineering*, A. Ralston and E. Reilly (eds.), 3rd ed., New York, NY: Van Nostrand Reinhold, 1991.

[11] S. Bubeck *et al.*, *Sparks of Artificial General Intelligence: Early experiments with GPT-4*, arXiv preprint arXiv:2303.12712, 2023.

[12] S. Chacon and B. Straub, *Pro Git*, 2nd ed. Apress, Berkeley, CA, USA, 2014.

[13] J. Loeliger and M. McCullough, *Version Control with Git: Powerful Tools and Techniques for Collaborative Software Development*, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2012.

[14] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, "Social coding in GitHub: Transparency and collaboration in an open software repository," in *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW)*, Seattle, WA, USA, 2012, pp. 1277–1286.

[15] F. Zampetti, S. Scalabrino, R. Oliveto, G. Canfora, and M. Di Penta, "How open source projects use static code analysis tools in continuous integration pipelines," in *Proceedings of the International Conference on Mining Software Repositories (MSR)*, 2017, pp. 334–344.

[16] V. Lenarduzzi, N. Saarimäki, and D. Taibi, "The technical debt dataset," in *Proceedings of the 15th International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE)*, 2019, pp. 2–11.

[17] V. Lenarduzzi, T. Besker, D. Taibi, A. Martini, and F. A. Fontana, "A systematic literature review on technical debt prioritization: Strategies, processes, factors, and tools," *Journal of Systems and Software*, vol. 171, p. 110827, 2021.

[18] H. Cui, M. Xie, T. Su, C. Zhang, and S. H. Tan, "An empirical study of false negatives and positives of static code analyzers from the perspective of historical issues," *arXiv preprint arXiv:2408.13855*, 2024.

[19] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, "Why don't software developers use static analysis tools to find bugs?" in *Proceedings of the 35th International Conference on Software Engineering (ICSE)*, 2013, pp. 672–681.

[20] J. Lu, L. Yu, X. Li, L. Yang, and C. Zuo, "LLaMA-Reviewer: Advancing code review automation with large language models through parameter-efficient fine-tuning," in *ISSRE*, 2023, pp. 647–658.

[21] Z. Rasheed, M. A. Sami, M. Waseem, K.-K. Kemell, X. Wang, A. Nguyen, K. Systä, and P. Abrahamsson, "AI-powered code review with LLMs: Early results," *arXiv preprint arXiv:2404.18496*, 2024.

[22] Z. Li, S. Lu, D. Guo, N. Duan, S. Jannu, G. Jenks, D. Majumder, J. Green, A. Svyatkovskiy, S. Fu, and N. Sundaresan, "Automating code review activities by large-scale pre-training," in *ESEC/FSE*, 2022, pp. 1–13.

[23] B. Berabi, J. He, V. Raychev, and M. Vechev, "DeepCode AI Fix: Fixing security vulnerabilities with large language models," *arXiv preprint arXiv:2402.13291*, 2024.

[24] J. Lu *et al.*, "Towards practical defect-focused automated code review," *arXiv preprint arXiv:2505.17928*, 2025.

[25] U. Cihan, V. Haratian, A. İcöz, M. K. Gül, Ö. Devran, E. F. Bayendur, B. M. Uçar, and E. Tüzün, "Automated code review in practice," in *ICSE-SEIP*, 2025.

[26] I. S. Göçmen, A. S. Cezayir, and E. Tüzün, "Enhanced code reviews using pull request based change impact analysis," *Empirical Software Engineering*, vol. 30, art. 64, 2025.

[27] C. Wang *et al.*, "LLMDFA: Analyzing dataflow in code with large language models," *arXiv preprint arXiv:2402.10754*, 2024.

Webographie

[**Web 1**] https://www.openxcell.com/artificial-intelligence/ (accessed 05 mai 2026)

[**Web 2**] https://www.rapidinnovation.io/pillar-pages/ai-software-development-the-ultimate-guide (accessed 05 mai 2026)

[**Web 3**] https://medium.com/@ramyasruthi.pedakolimi/ai-agents-explained-a-beginner-friendly-guide-to-the-new-era-of-intelligent-systems-de30dfde0cba (accessed 05 mai 2026)

[**Web 4**] https://www.microcenter.com/site/mc-news/article/five-types-of-ai-agent.aspx (accessed 05 mai 2026)

[**Web 5**] https://zigment.ai/blog/what-is-agentic-ai-a-definitive-guide (accessed 05 mai 2026)

[**Web 6**] https://www.cyberark.com/resources/blog/the-rise-of-ai-agents-collaborative-intelligence (accessed 05 mai 2026)

[**Web 7**] https://bigr.io/agentic-ai/ (accessed 05 mai 2026)

[**Web 8**] https://www.squareonemea.com/ar/blogs/agentic-ai-in-banking-use-cases-benefits-future-trends/ (accessed 05 mai 2026)

[**Web 9**] https://www.xenonstack.com/blog/contact-center-agentic-ai (accessed 05 mai 2026)

[**Web 10**] https://www.linkedin.com/pulse/agentic-ai-redefining-future-supply-chain-logistics-sahil-gupta-pvt4c/ (accessed 05 mai 2026)

[**Web 11**] https://www.atlassian.com/git/tutorials/comparing-workflows (accessed 05 mai 2026)

[**Web 12**] GitHub Docs, About pull requests, https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-pull-requests (accessed 05 mai 2026)

[**Web 13**] GitHub Docs, About Webhooks, https://docs.github.com/en/developers/webhooks-and-events/webhooks/about-webhooks (accessed 05 mai 2026)

[**Web 14**] https://www.sonarsource.com/products/sonarqube/ (accessed 05 mai 2026)

[**Web 15**] https://www.digitalocean.com/community/tutorials/linting-and-formatting-with-eslint-in-vs-code (accessed 05 mai 2026)

[**Web 16**] Amazon Web Services, What is Amazon CodeGuru Reviewer?, https://docs.aws.amazon.com/codeguru/latest/reviewer-ug/welcome.html (accessed 05 mai 2026)

[**Web 17**] https://aws.amazon.com/codeguru/reviewer/ (accessed 05 mai 2026)

[**Web 18**] CodeRabbit AI Code Review Platform, https://coderabbit.ai (accessed 05 mai 2026)

[**Web 19**] https://snyk.io/platform/deepcode-ai/ (accessed 05 mai 2026)

[**Web 20**] Anthropic, Claude Code — AI Code Review, https://www.anthropic.com/claude (accessed 05 May 2026)

[**Web 21**] n8n Workflow Automation Platform — Official Documentation, https://docs.n8n.io (accessed 05 mai 2026)

[**Web 22**] IBM Think Topics, What is n8n?, https://www.ibm.com/think/topics/n8n (accessed 05 May 2026)

[**Web 23**] IBM Think Topics, What is PostgreSQL?, https://www.ibm.com/think/topics/postgresql (accessed 05 May 2026)

[**Web 24**] IBM Think Topics, What is Docker?, https://www.ibm.com/think/topics/docker (accessed 05 May 2026)

[**Web 25**] Microsoft Learn, Windows Subsystem for Linux, https://learn.microsoft.com/en-us/windows/wsl/about (accessed 05 May 2026)

[**Web 26**] IBM Think Topics, What is Django?, https://www.ibm.com/think/topics/django (accessed 05 May 2026)

[**Web 27**] Tailwind CSS, Official Documentation, https://tailwindcss.com/docs (accessed 05 May 2026)

[**Web 28**] Google DeepMind, Gemini Technology, https://deepmind.google/technologies/gemini (accessed 05 May 2026)

[**Web 29**] ngrok, Official Documentation, `https://ngrok.com/docs` (accessed 05 May 2026)

[**Web 30**] Microsoft, Visual Studio Code Documentation, `https://code.visualstudio.com/docs` (accessed 05 May 2026)

[**Web 31**] Git, Official Documentation, `https://git-scm.com/doc` (accessed 05 May 2026)

[**Web 32**] Python, Official Documentation, `https://docs.python.org` (accessed 05 May 2026)

[**Web 33**] MDN Web Docs, HTML: HyperText Markup Language, `https://developer.mozilla.org/en-US/docs/Web/HTML` (accessed 05 May 2026)

[**AIGC 1**] Anthropic, Claude Sonnet 4, AI-generated visual content.