Athena Code Assistant

Provides Security and efficiency in On-Premise environment

uracle

Code Assistant?

AI Tools for the Entire Organization, Beyond Coding

Code Generation

Instantly generate functions, classes, and scripts using natural language commands.

Document Automation

Reduce documentation effort by auto-generating docs.

Code Review & Debugging

Provides expert-level code reviews

Business Tasks

Extends capabilities to planning, operations, enabling SQL generation, automation scripts and more.

Legacy System Analysis

Refactoring Recommendations & Dependency Visualization

Knowledge Management

Contextual Understanding Based on Internal Codebases and Knowledge

Limitations of Existing Solutions

Why Existing AI Code Assistants Fall Short in Enterprise Environments

Blocking Access to External AI Services Within Secure Networks

Cloud-based AI code assistants require external API access, making them unsuitable for secure and network-segregated environments.

Inability to Control AI Usage at the Team and Organizational Levels

Personal SaaS tools lack organization-wide access control, usage limits, and AI usage visibility.

Difficulty Ensuring the Quality and Security Reliability of AI-Generated Code

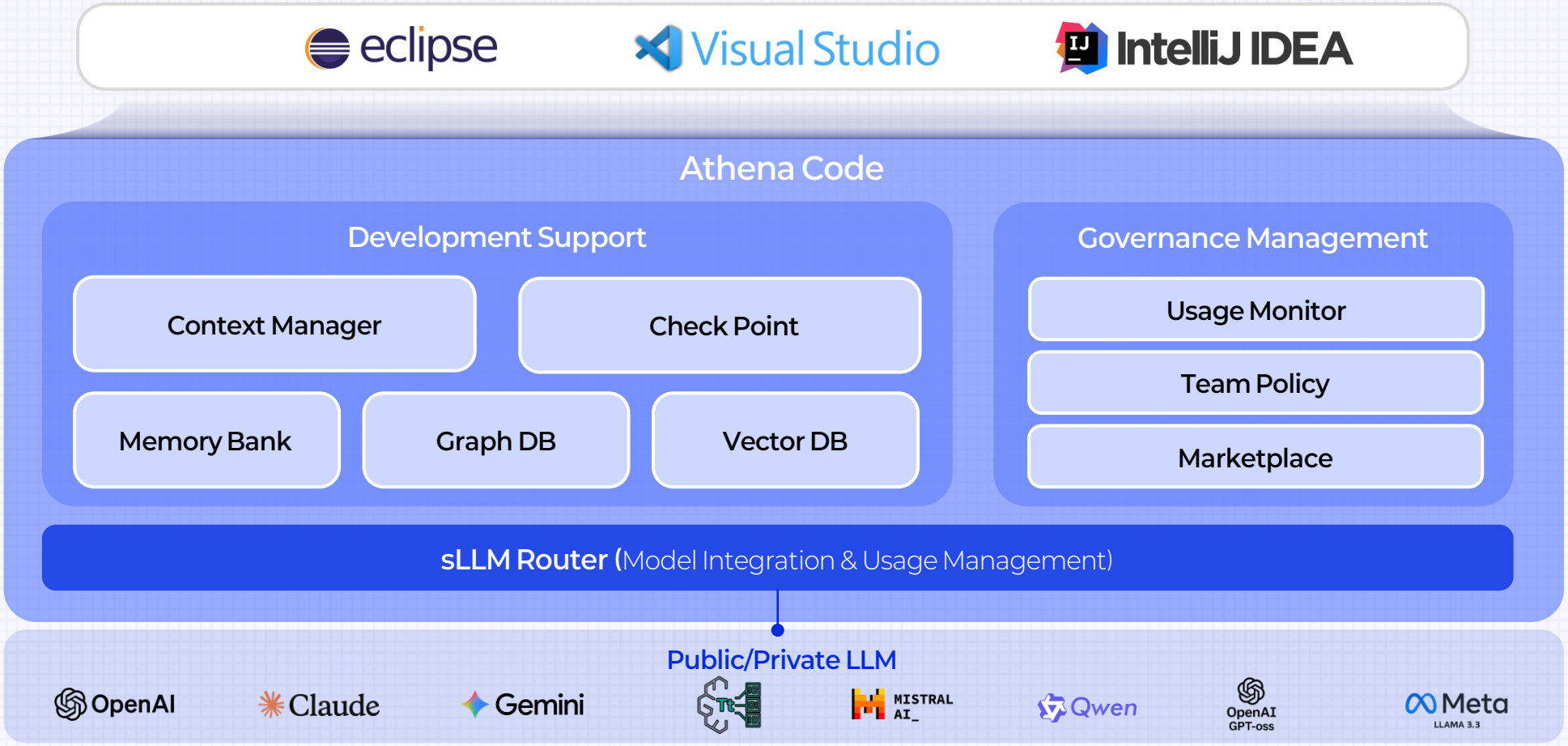
AI-generated code requires quality and security validation, with auditability essential in financial and public-sector environments.

Limitations in Understanding Large-Scale Legacy Codebases

File-level AI tools lack system-wide context, limiting their understanding of complex dependencies and code impact.

Enterprise Optimization

Athena Code Assistant for Enterprise Organizations



Solution

Overcoming the Limitations of Existing AI Code Assistants

Enhanced Security

Isolated Network Operation

FOCUS

- Independent Internal Network Operation
- Air-Gapped Ready

Technologies Applied

- Selective Public/Private LLM
- sLM Router–Based Model Management

Automated Audit Compliance

FOCUS

- Automated AI Usage Logging
- Admin Dashboard Management & Monitoring

Technologies Applied

- Automated Audit Logging
- Team Policy–Based Access Control

Strengthened Development Capabilities

Contextual Understanding of Legacy Code

FOCUS

- Unified System Architecture
- Instant Impact Analysis

Technologies Applied

- Graph DB Visualization
- Memory Bank & Vector DB Knowledge Search

Visibility into AI Usage

FOCUS

- Real-Time Usage Monitoring
- User-Based Model Access

Technologies Applied

- Usage Monitor Dashboard
- User-Based Model Usage Limits

Feature Overview

Develop Faster, Stay Safer.

01 Automation of Repetitive Tasks

Context-Persistent Development Environment

Context | CheckPoint | / Command

02 Legacy & Large-Scale Code Analysis

Instant Structural Analysis of Large-Scale Codebases

Graph DB | Memory Bank

03 Security & Audit Compliance

Complete AI Usage Tracking & Auditability

Log | Compliance

04 Cost & Usage Control

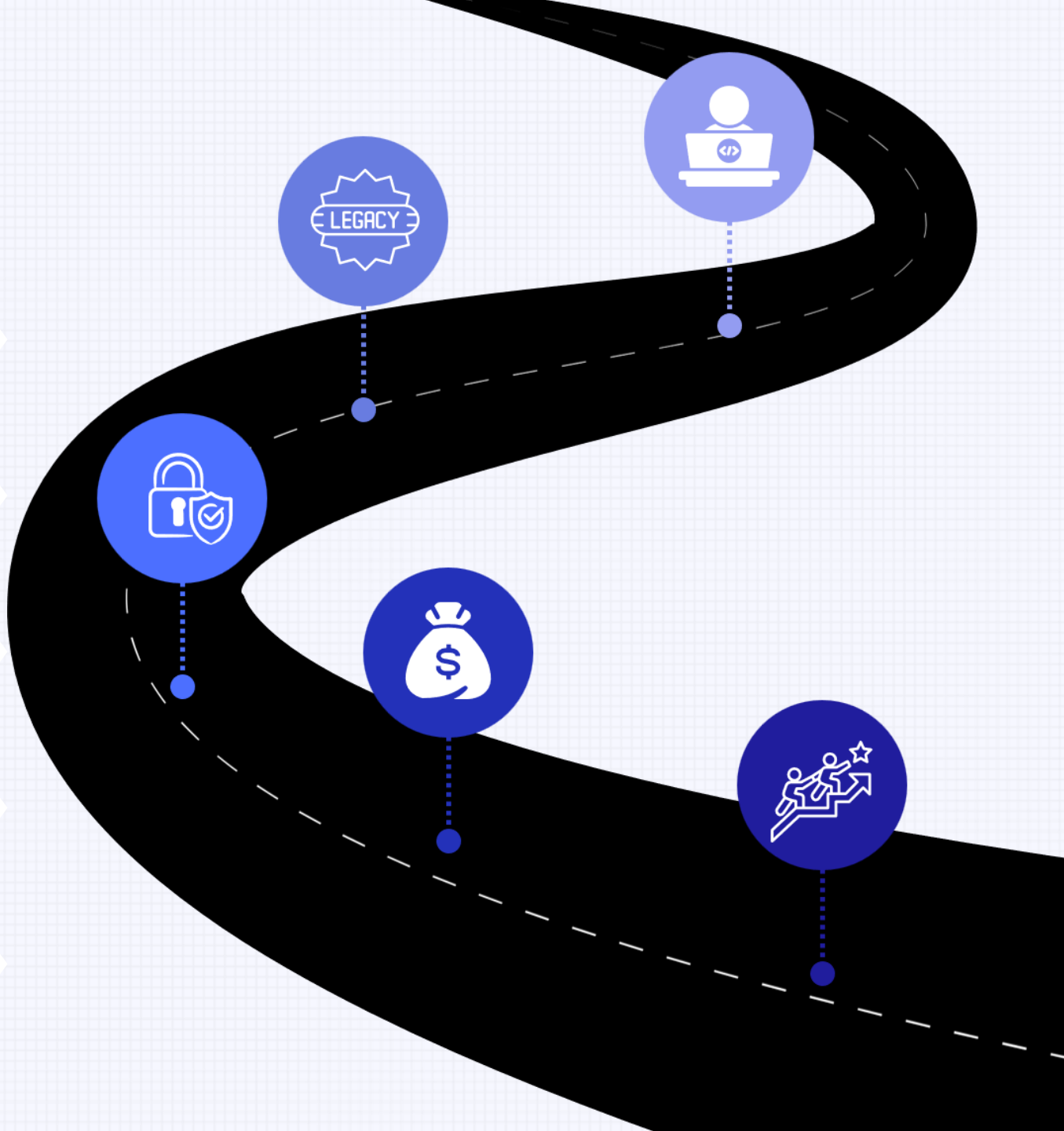
Real-Time Cost Visibility by Team & User

sLLM Router | AI Monitoring | Usage

05 Organization-Specific Customization

Organization-Specific AI Environment Configuration

Rules | MCP | Market Place



Repetitive Task Automation

Modes

Define AI's scope of operation by task to prevent unnecessary file access and code changes.

Code Mode

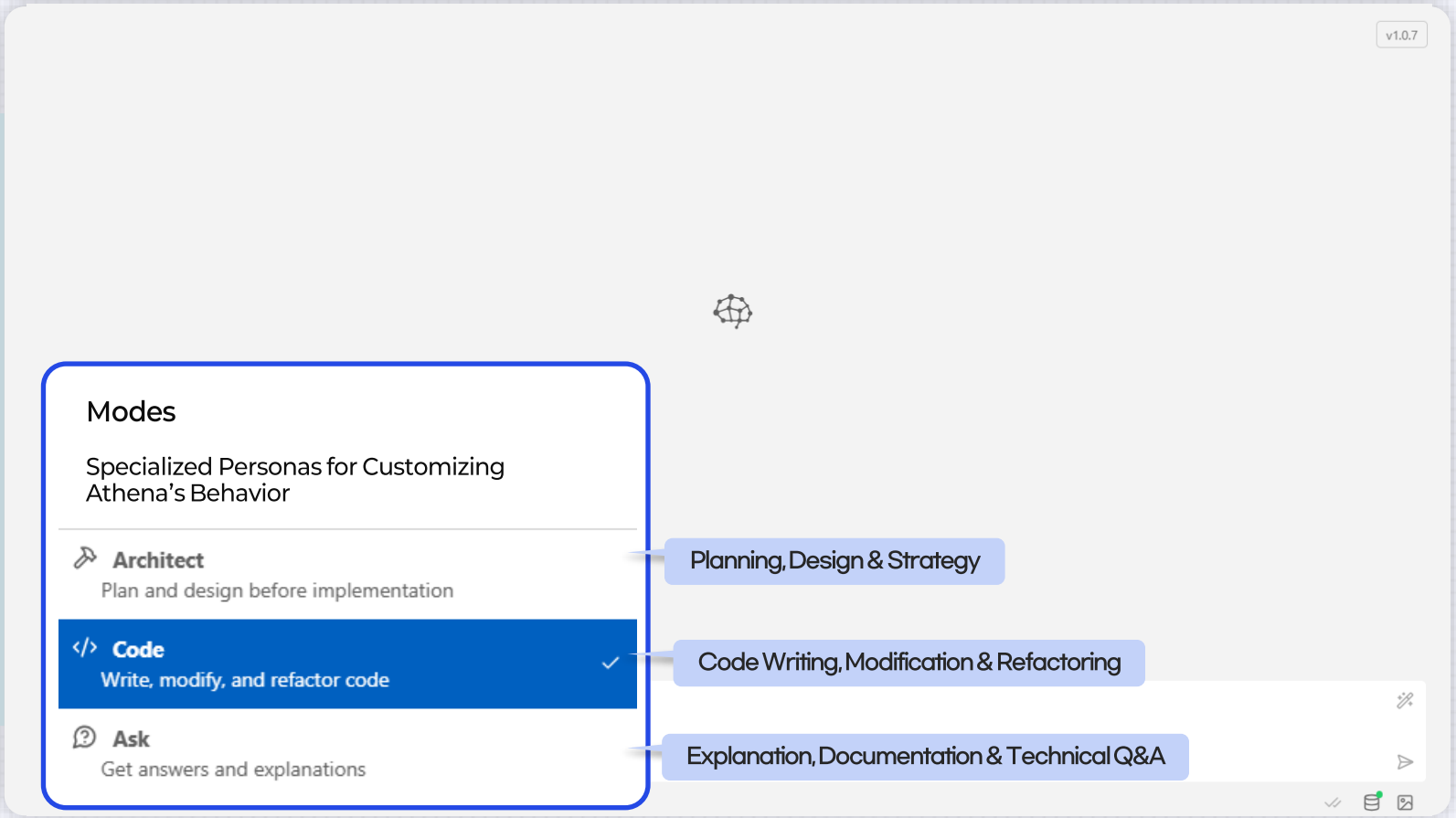
Optimized for code generation, modification, and refactoring, including file editing and bug fixing to accelerate development.

Architect Mode

Optimized for system design, technology stack selection, and project architecture planning to ensure design completeness before implementation.

Ask Mode

Optimized for concept explanations and technical queries, performing read-only analysis to prevent unnecessary code changes.



Repetitive Task Automation

Context Setting

Controls the information included in the AI's context window, influencing token usage and response quality.

Context Adjustment

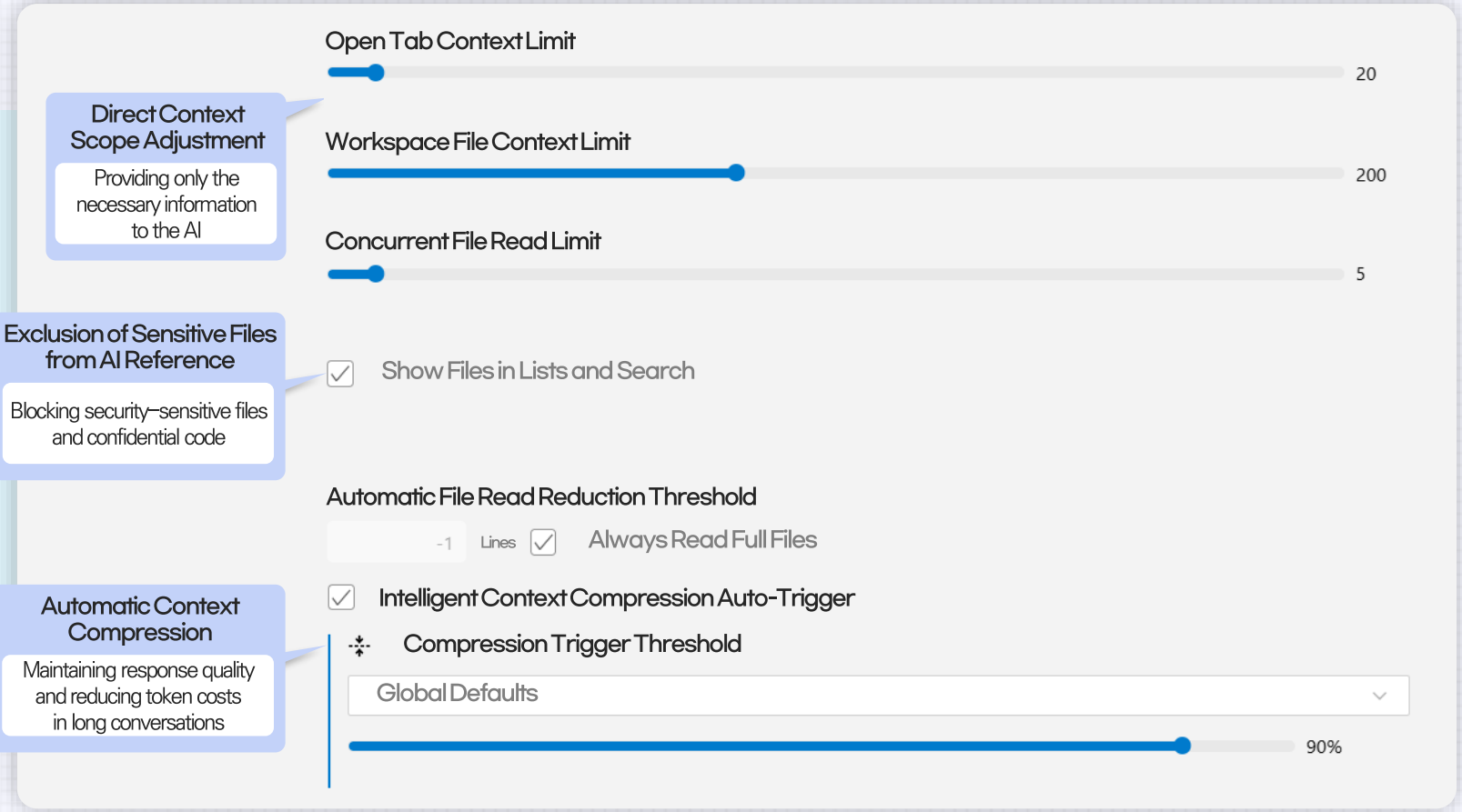
Adjusting open tabs, workspace files, and concurrent read limits to optimize cost, response quality, and overall performance

Automatic Blocking

Automatically excluding sensitive files and directories from AI access and search to ensure transparent security governance

Ensuring Task Stability

Automatically summarizing and compressing previous conversations when the context usage threshold is reached, enabling uninterrupted task continuity



Repetitive Task Automation

Check Points

Automatically creates a snapshot before modifying files, allowing you to safely restore the previous state at any time.

Auto-Snap Shot

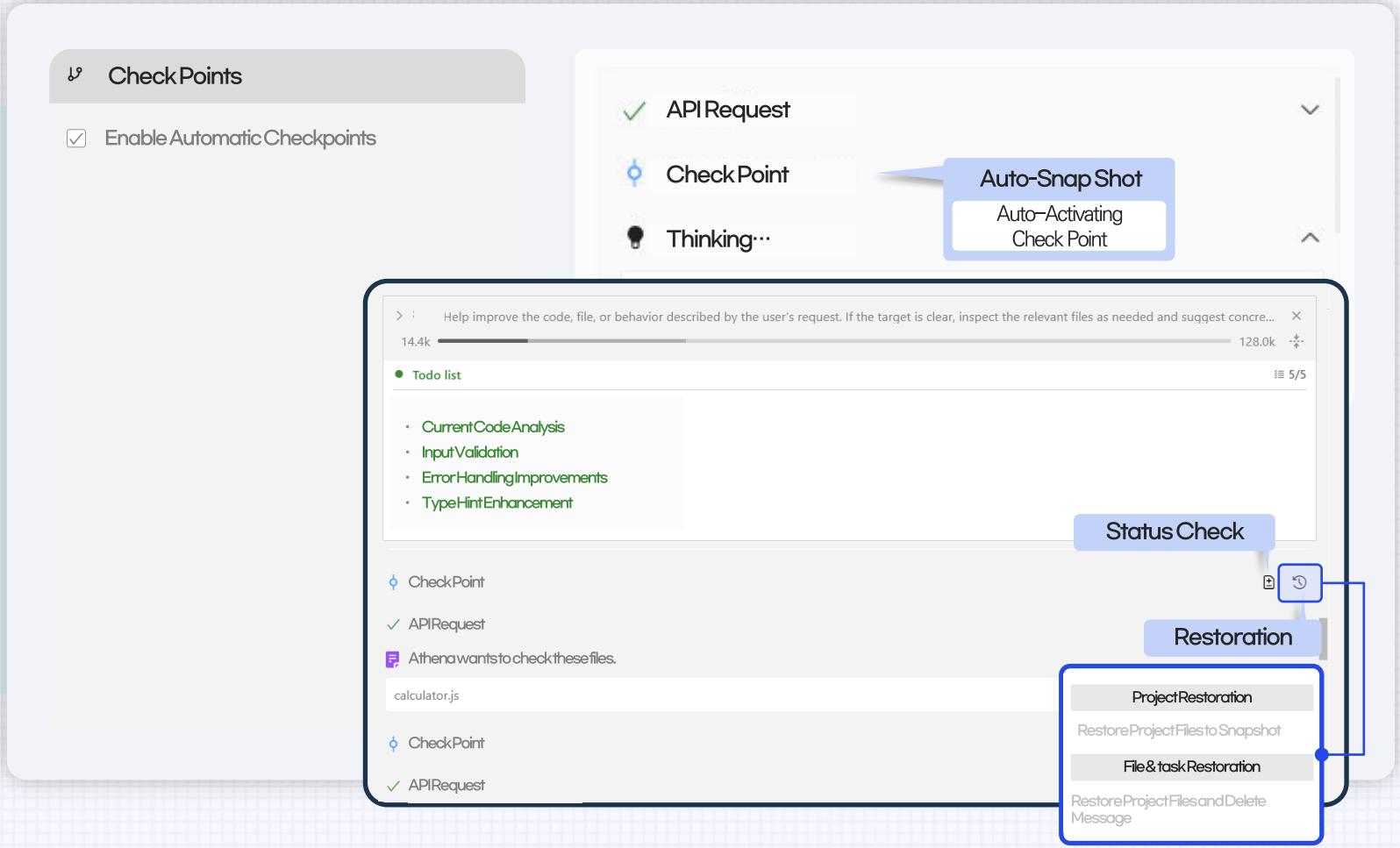
Automatically creates Git-based snapshots before file modifications begin, preserving the complete change history.

State Restoration

Enables restoration to a checkpoint at any desired point, preventing data loss caused by mistakes or unintended changes.

Workflow Stability

Automatically managed during work without additional configuration, maintaining a stable development environment without interrupting the workflow.



Repetitive Task Automation

Context Mentions & Slash Commands

Use @mentions to specify the resources the AI should reference, and execute repetitive tasks instantly with commands.

Context Source

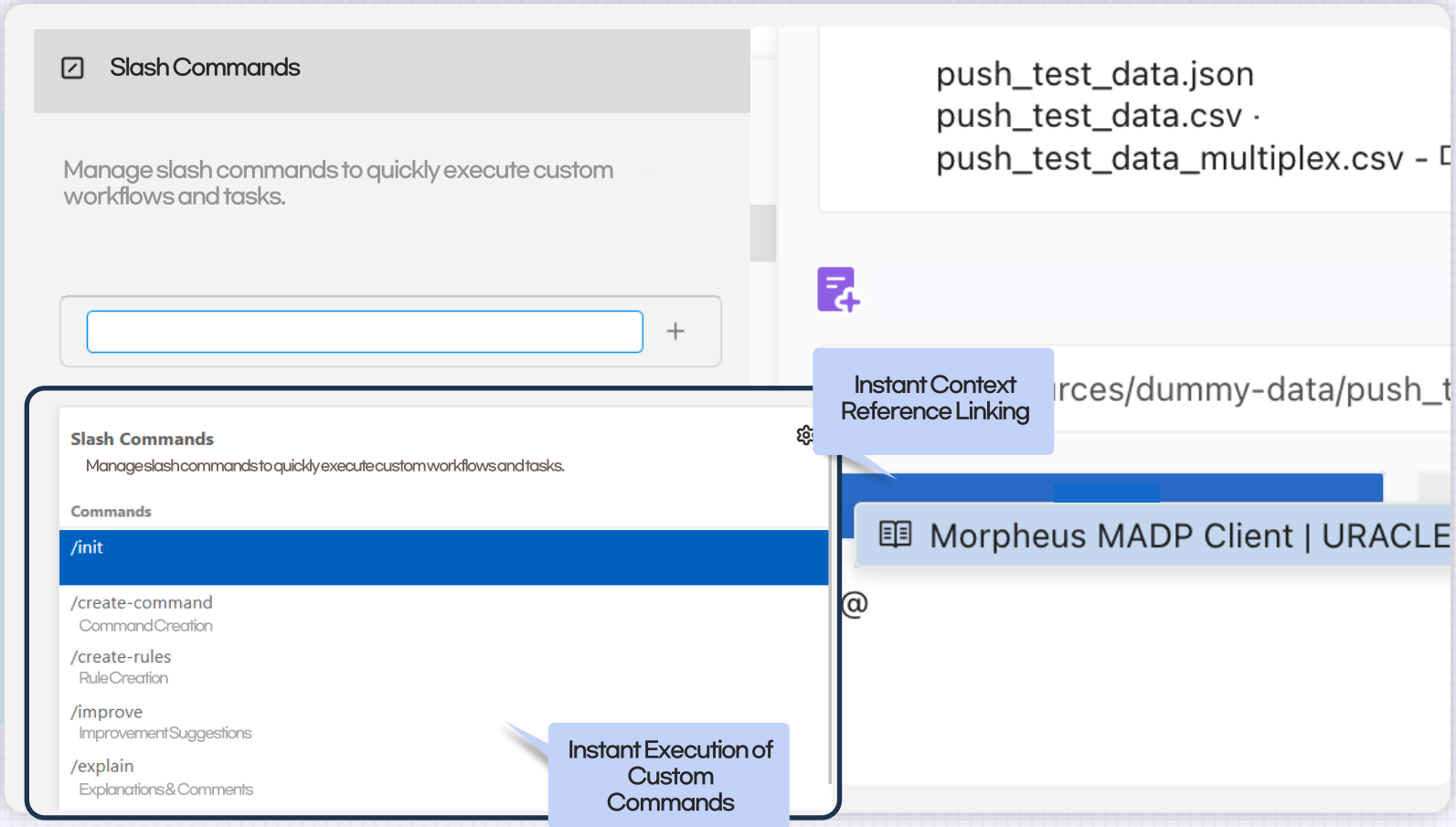
Reference files, folders, terminals, Git, and URLs with @mentions to provide information to the AI.

Quick Execution

Register frequently used task patterns as slash (/) commands for instant execution without writing prompts.

Shared Standard Code

Share team-defined Rules and Commands through a marketplace and apply them instantly across all team members' IDEs.



Legacy & Large-Scale Code Analysis

Memory Bank

Structures Important Decisions, Tasks, and Patterns to maintain consistent understanding across sessions.

Project Context Preservation

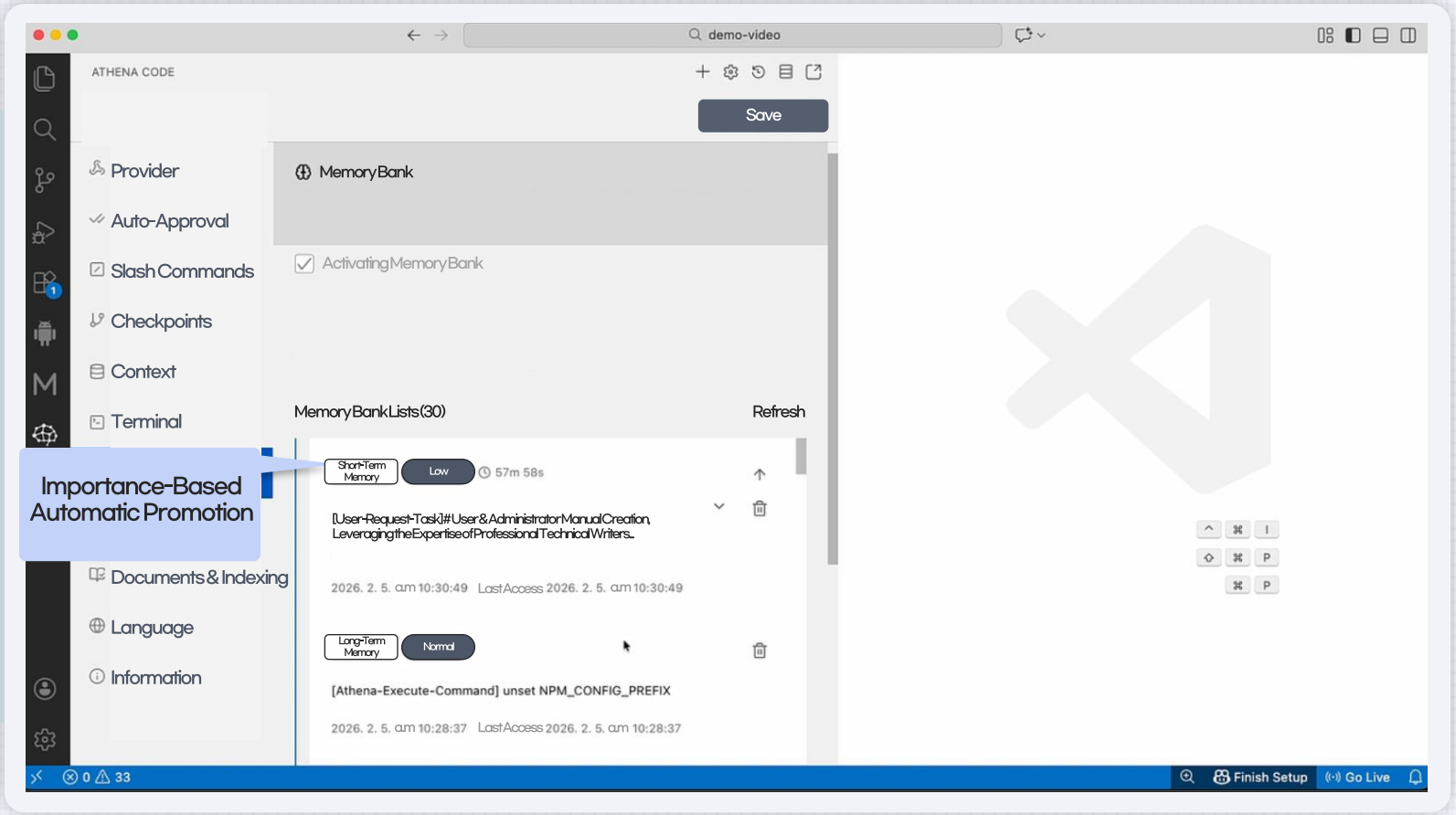
Preserve design decisions and work history across sessions using a hierarchical STM (Short-Term Memory) and LTM (Long-Term Memory) structure.

Non-Redundant Knowledge

Automatically detect duplicates based on vector similarity and reconcile conflicting information with existing content to maintain up-to-date knowledge integrity.

Instant Context Retrieval

Instantly retrieve relevant context from project-specific, independent knowledge bases to maintain continuity without interrupting the workflow.



Legacy & Large-Scale Code Analysis

Graph DB-Based Code Analysis

Visualize codebase call relationships and dependencies to quickly understand even large-scale legacy code.

Code Relationship Analysis

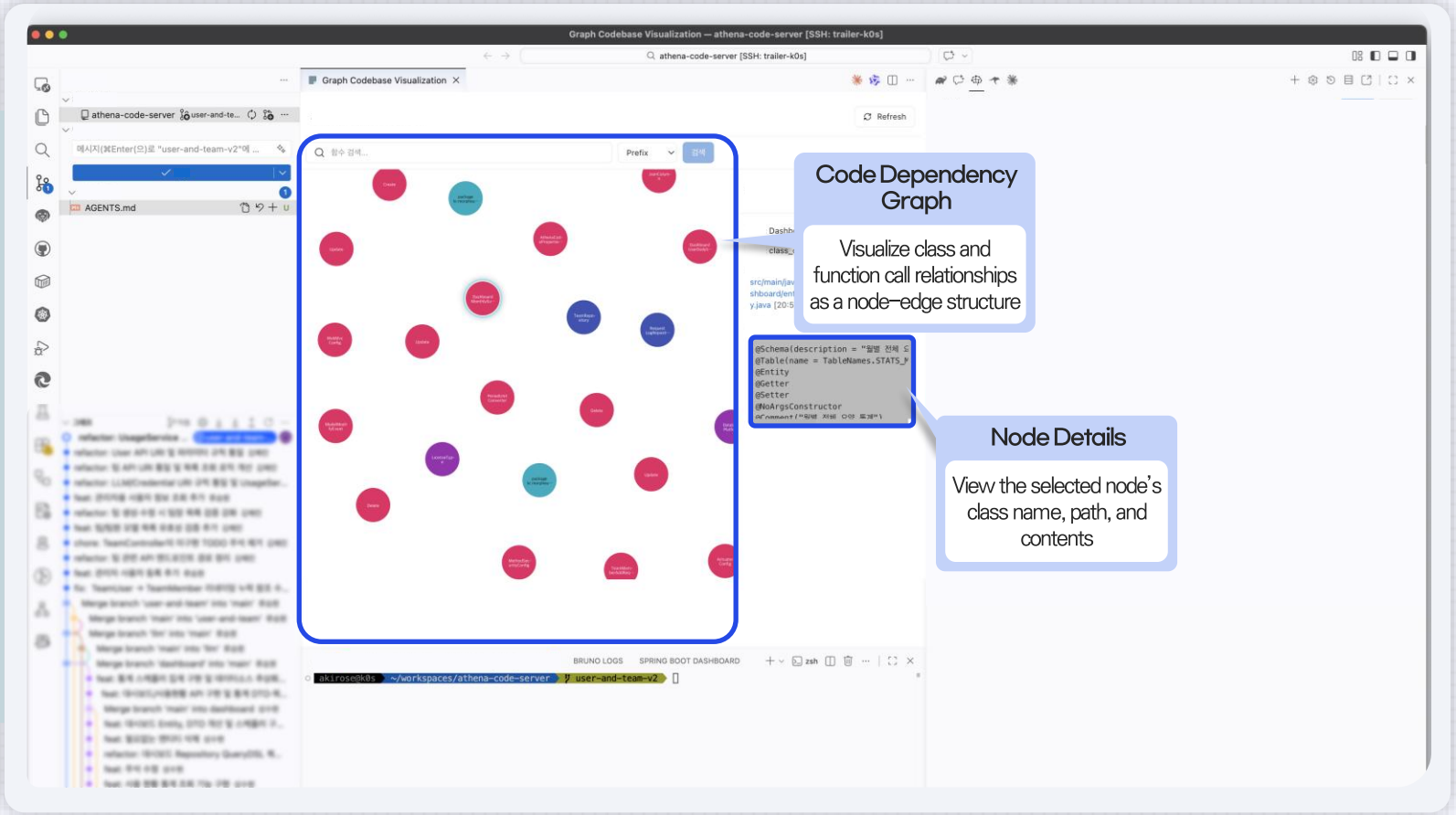
Visualize class and function call relationships using a graph database to understand the overall code structure.

Semantic Code Chunking

Automatically split code into function- and class-level units to enable precise code retrieval through natural language queries.

Legacy Code Understanding

Automatically index large-scale legacy code using dependency graphs, significantly reducing the time required for newly assigned developers to understand the codebase.



Security & Audit Compliance

Audit Logs & Compliance

Track AI usage history across the entire team in real time, with the ability to view and export detailed information for each request at any time.

History Review

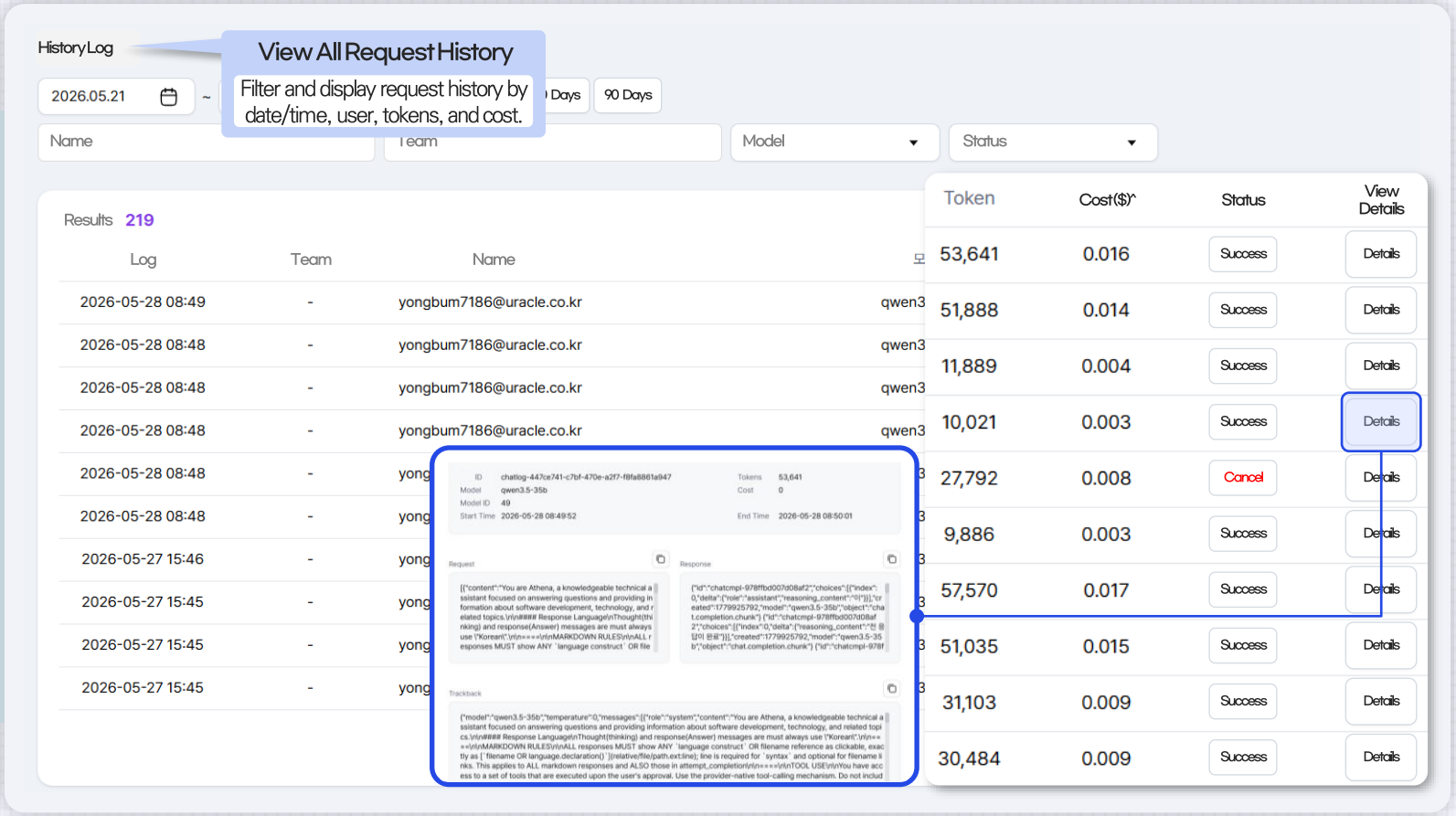
Record and retrieve all AI request history in real time based on date/time, team, user, model, tokens, cost, and status.

Request History Review

Review the full Request, Response, and Trackback details of individual requests to assess response quality and identify anomalies.

Audit Compliance

Filter logs by period, team, user, and model, then export them for internal audits and compliance requirements.



Cost & Usage Control

AI Usage Visibility & Cost Control

Monitor AI usage and costs in real time by team, user, budget, and model to control AI usage across the organization.

Usage Monitoring

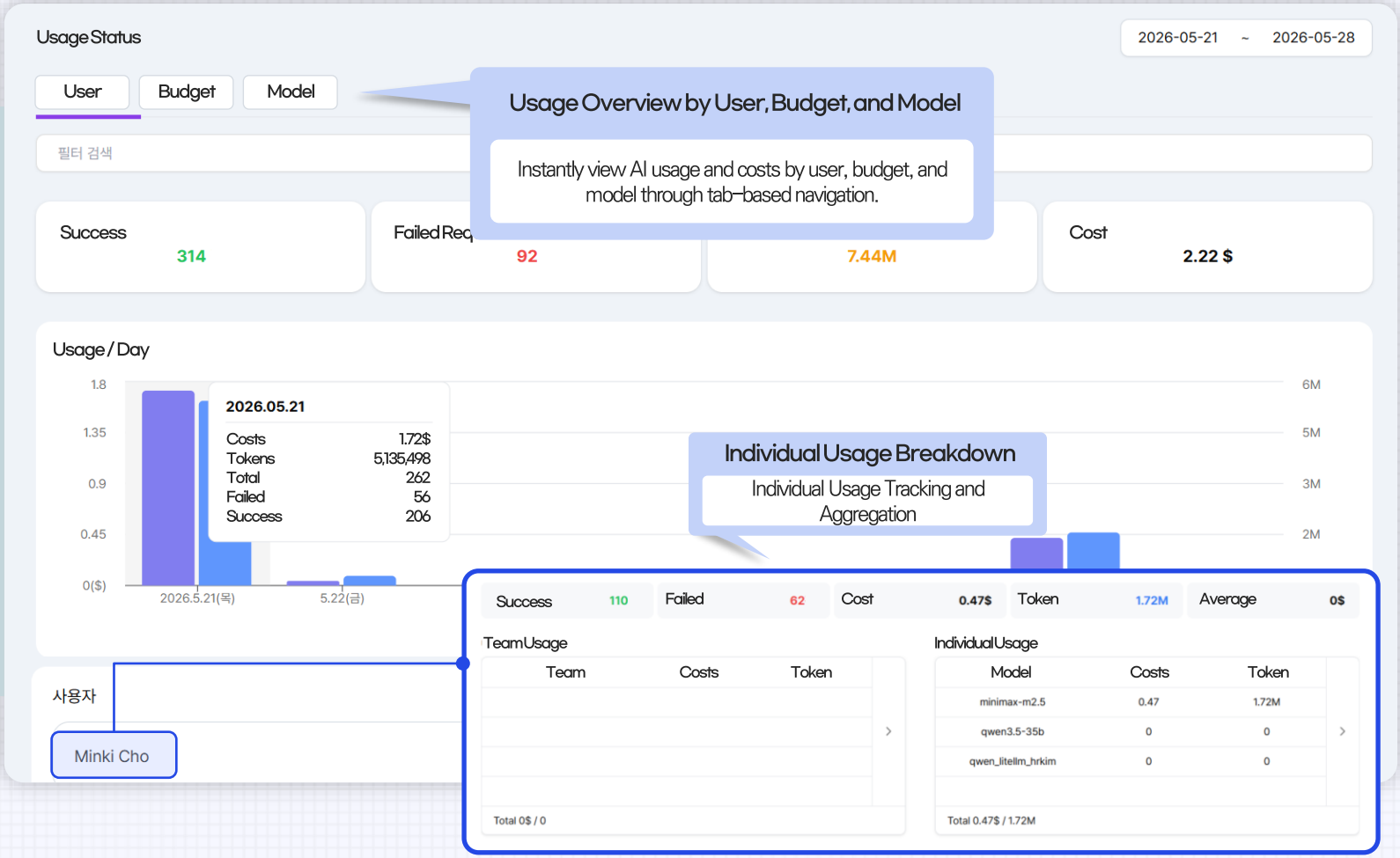
Real-Time Aggregation of Successful/Failed Requests, Tokens, and Costs for Immediate Visibility into Organization-Wide AI Usage.

User Analytics

Identify cost-intensive areas and detect anomalous usage through dedicated tabs for users, budgets, and models.

Cost Configuration

Set spending limits by user and team to proactively block usage before exceeding budgets and prevent unnecessary costs.



Organization-Specific Customization

Rules

Define the team's coding standards in a document, and AI automatically applies them across every developer's IDE.

Standard Distribution

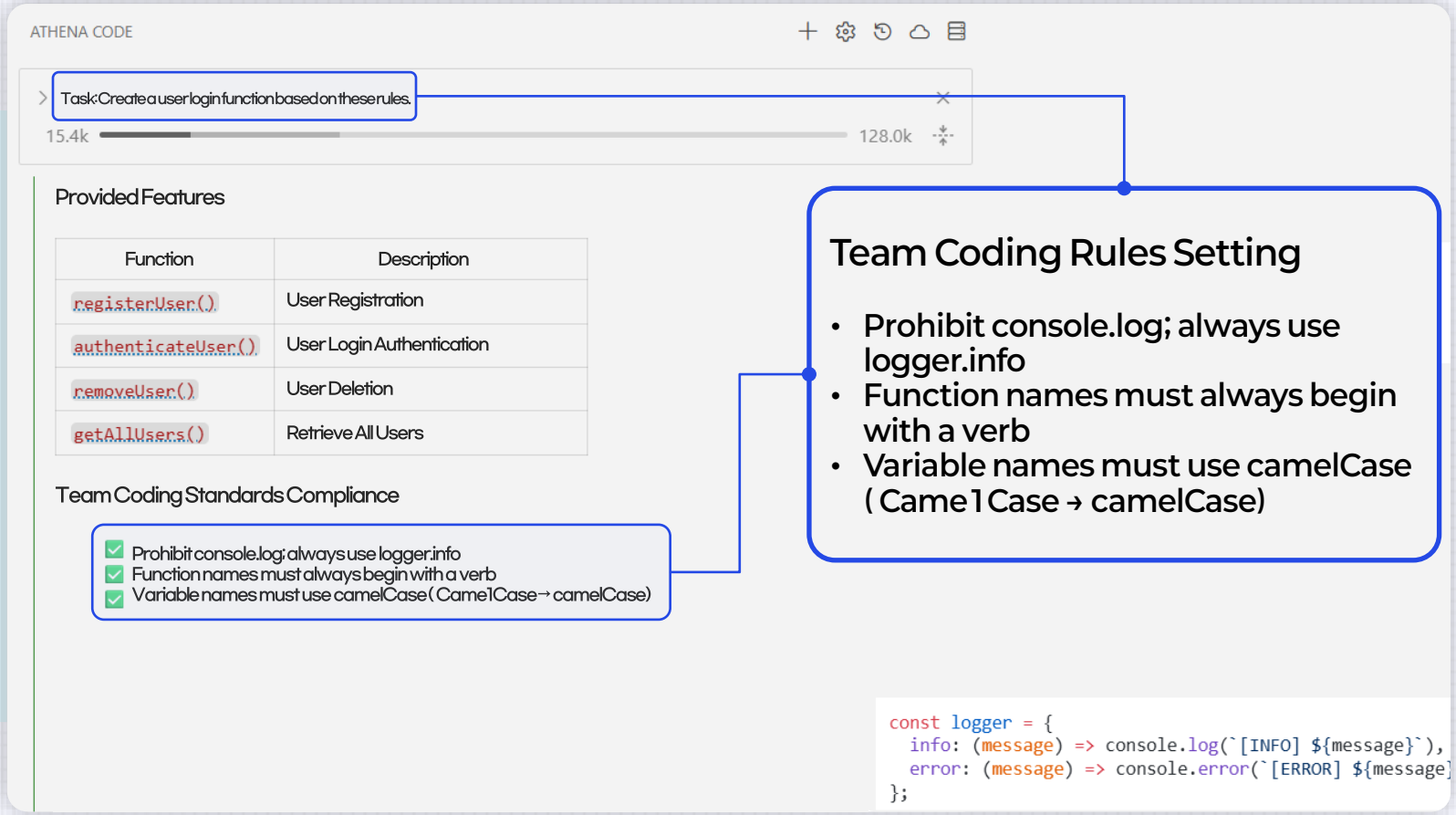
Apply rules defined by team leads and senior developers instantly across all team members' IDEs to ensure consistent code quality.

Rule Enforcement

Detect rule violations in real time during coding and provide automatic fix suggestions to minimize code review overhead.

Project Segmentation

Manage standards flexibly by separating organization-wide rules from project-specific rules.



Key Feature Comparison

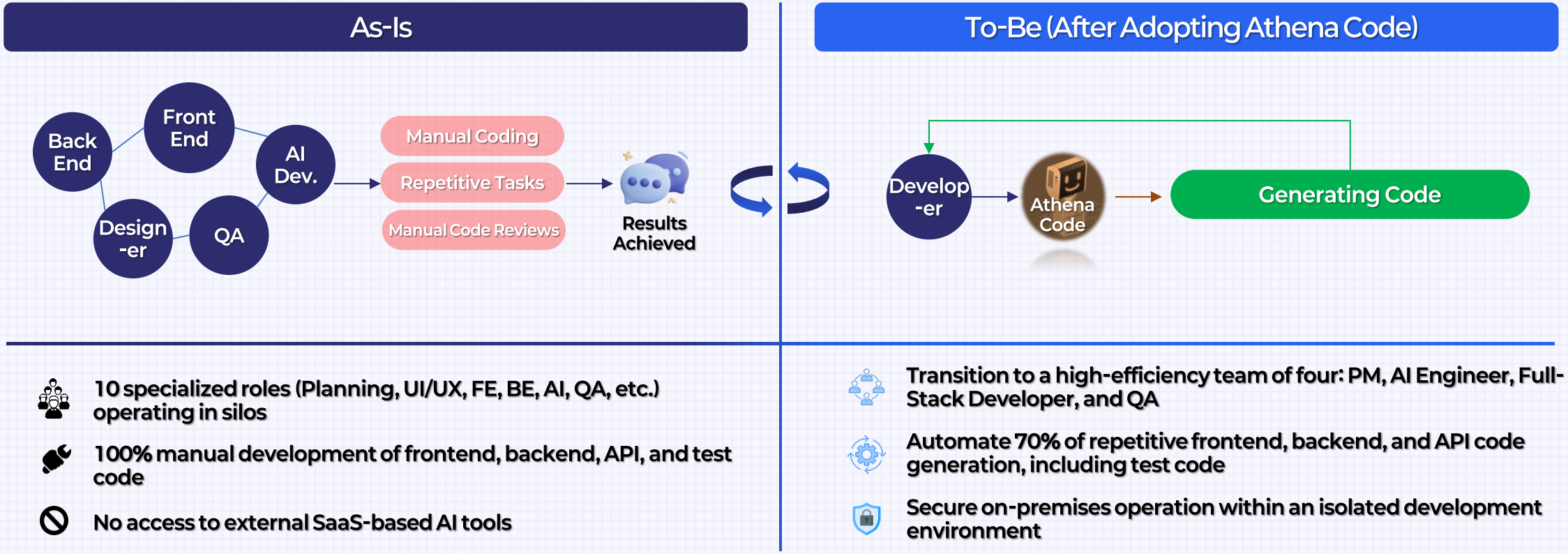
Each product is optimized for different environments and purposes. This reference summarizes the support status of key features based on publicly available information.

※ As of June 2026

Features	CURSOR	Claude	Copilot	Athena Code
Code Autocomplete	✅	✅	✅	✅
Full Codebase Context Understanding	✅	✅	✅	✅
IDE Support (VS Code)	✅	✅	✅	✅
IDE Support (IntelliJ · JetBrains)	✅	✅	✅	✅
IDE Support (Eclipse)	✗	✗	✅	✅
Long-Term Context Preservation (Across Sessions)	✅	✅	✗	✅
Fully Closed-Network & Air-Gapped Operation	✗	✗	✗	✅
On-Premises Deployment	△	✗	✗	✅
Real-Time Team Usage & Cost Monitoring	△	△	△	✅
Automated Audit Logging & Export	△	△	△	✅
Team Coding Standards Synchronization	△	✗	✗	✅
Graph DB-Based Code Structure Visualization	△ Requires separate installation and integration of external tools			✅

Implementation Example

Internal Chatbot Development Use Case



- 10 specialized roles (Planning, UI/UX, FE, BE, AI, QA, etc.) operating in silos
- 100% manual development of frontend, backend, API, and test code
- No access to external SaaS-based AI tools

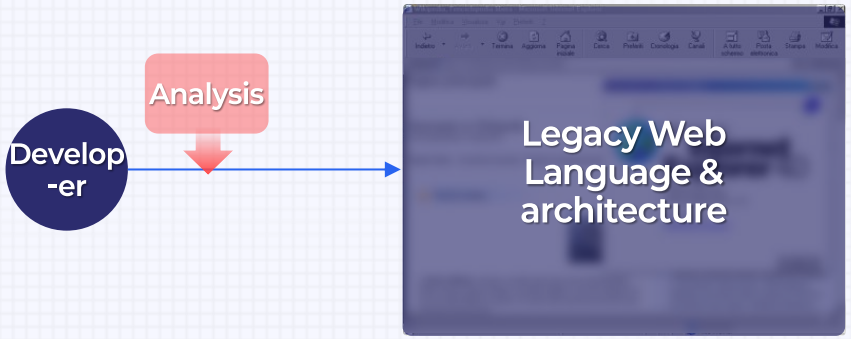
- Transition to a high-efficiency team of four: PM, AI Engineer, Full-Stack Developer, and QA
- Automate 70% of repetitive frontend, backend, and API code generation, including test code
- Secure on-premises operation within an isolated development environment

Based on internal PoC results, approximately 40% reduction in development effort and 50% reduction in development time.

Implementation Example

Legacy Web Modernization Use Case

As-Is



- Core business systems currently operating on an IE-based JSP architecture.
- Developers manually perform the entire process of screen analysis and refactoring.
- SaaS-based AI solutions cannot be used due to Internet access restrictions in the development environment.

To-Be (After Adopting Athena Code)

- AI-powered automatic analysis and conversion of JSP logic and UI directly within the IDE.
- Progressive automated conversion and refactoring to a modern Chrome/Spring-based architecture.
- Secure operation even within an isolated development environment, significantly reducing costs and development time.

Significantly reduce the cost, timeline, and risks of legacy modernization within an isolated development environment.

Deployment Guide

Support for Code Assistant Environments of Various Scales

Supports not only large-scale environments but also small- and medium-sized environments with limited GPU resources.

※ Qwen3.6-35B-A3B sLM

Category	Device	Available Memory	AI Model Memory Usage	Recommended Number of Users	Key Features
Personal	AMD AI Max 395+	128GB (Shared)	~20GB	1	High-Performance Laptop
	MacBook M5 Pro	64GB (Shared)			Personal Laptop Environment Optimization
Team	Mac Studio M3 Ultra	96GB (Shared)		3	Small Team Shared Use, Multi-Session Optimization
	RTX PRO 6000	96GB VRAM		6~10	Medium Team Operations Optimization, Maximum Performance

💡 The Code Assistant does not consume GPU resources while users are typing. Actual GPU utilization is only around 10-20% of the total, meaning a single RTX PRO 6000 can effectively support a team of 6-10 users.

Thank you

H. uracle.co.kr
E. sales@uracle.co.kr
T. +82-2-3479-4400