Software Valuation Report

Analysed Source Code: OpenMontage

Document Date: June 24, 2026

Platform:

Client / Applicant

Legal entity name:

Registered address:

Tax Identification Number:

Software name:

Existing trade mark:

Company web:

Applicant Name:

Applicant Email:

Request date and time: 24-06-2026 - 22:05:47

TABLE OF CONTENTS

Software Valuation Report

1. EXECUTIVE SUMMARY

OpenMontage is a production-grade, agent-orchestrated video production platform demonstrating strong architectural maturity with comprehensive documentation, scored provider selection, and robust budget governance. The system represents a sophisticated software engineering effort that transforms AI coding assistants into full video production studios through an instruction-driven architecture rather than traditional code orchestration.

The platform achieves an overall production readiness score of **63/100 (Good)**, reflecting solid foundational engineering with clear pathways to excellence. The codebase exhibits good code quality with clear separation of concerns across 425 files and 41,893 effective lines of code spanning Python, TypeScript, Markdown, YAML, JSON, and Shell. The three-layer knowledge architecture — separating executable tools, OpenMontage-specific conventions, and external technology knowledge packs — demonstrates thoughtful design for maintainability and extensibility.

Key strengths include comprehensive documentation with over 400 skill files covering all production workflows, JSON Schema validation for all artifacts and checkpoints ensuring data integrity across pipeline stages, and a scored provider selection engine evaluating seven dimensions (task fit, quality, control, reliability, cost, latency, continuity) with auditable decision trails. The budget governance system implements an estimate-reserve-reconcile pattern preventing cost overruns, while delivery promise enforcement and slideshow risk scoring prevent quality degradation.

Critical risks centre on the absence of CI/CD pipeline configuration — linting, testing, and security scanning are not enforced automatically — and incomplete test coverage estimated at 50-65% where contract tests exist but comprehensive unit testing across all 57+ tools remains incomplete. The system lacks structured logging with correlation IDs, health check endpoints, and Prometheus-style metrics exposure necessary for production observability. Error handling uses custom exception classes but lacks consistent retry logic with exponential backoff across all external API calls.

The development investment to date is estimated at **4,100 hours**, representing approximately **€383,350 to €518,650** in development costs (at €75-150/hour) over a **9-month** period with a team of 3 developers. This valuation reflects the substantial intellectual capital

embedded in the platform's sophisticated domain logic, extensive third-party integrations, and production-grade governance systems.

---

2. PLATFORM OVERVIEW

2.1 Functional Description

OpenMontage is an end-to-end video production platform that orchestrates AI coding assistants to produce professional-quality videos through structured pipelines. The system transforms plain language prompts into complete video productions, handling research, scripting, asset generation, editing, and final composition without requiring a traditional runtime orchestrator.

Core Features and Capabilities:

- **12 Production Pipelines**: Complete workflows for animated explainers, animations, avatar spokespersons, cinematic trailers, clip factories, documentary montages, hybrid productions, localisation dubs, podcast repurposing, screen demos, talking heads, and framework smoke tests
- **57+ Production Tools**: Spanning video generation (14 providers), image creation (10 providers), text-to-speech (4 providers), music generation, audio mixing, enhancement, analysis, and composition
- **400+ Agent Skills**: Production skills, pipeline directors, creative techniques, quality checklists, and deep technology knowledge packs teaching the agent how to use every tool like an expert
- **Reference-Driven Creation**: Users can paste existing videos (YouTube, TikTok, Reels) and receive differentiated production plans grounded in analysis of pacing, hook style, structure, and tone
- **Dual Runtime Support**: Remotion for React-based composition and HyperFrames for HTML/GSAP rendering, with governance preventing silent swaps between engines
- **Budget Governance**: Estimate-reserve-reconcile pattern with configurable spending modes, per-action approval thresholds, and total budget caps

User-Facing Functionality:

The platform accepts natural language prompts such as "Make a 60-second animated explainer about how neural networks learn" or "Create a documentary montage about city life in the rain using real footage only". The agent then executes a structured production workflow:

1. Research phase with live web search across YouTube, Reddit, news sites, and academic sources
2. Proposal generation with concept options, production plans, and cost estimates
3. Script writing with enhancement cues and pronunciation guides
4. Scene planning with type definitions, timing, and shot language
5. Asset generation and sourcing from configured providers
6. Editorial decisions with cut timings and transitions
7. Composition via Remotion or HyperFrames
8. Publishing with platform-specific output profiles

Target Users:

- Content creators and marketers producing regular video content
- Educational institutions creating instructional materials
- Enterprises requiring scalable video production for training and communications
- Developers extending video production capabilities through custom tools and pipelines

2.2 Technical Architecture

OpenMontage employs an **agent-first architecture** where the AI coding assistant serves as the orchestrator, reading pipeline manifests and skill instructions to drive production. There is no runtime Python orchestrator; Python provides tools and persistence only.

System Components and Responsibilities:

Component	Responsibility
`lib/`	Core infrastructure: config model, checkpoints, pipeline loader, media profiles, scoring, delivery promise, slideshow risk
`tools/`	57+ Python tool implementations inheriting from BaseTool abstract class
`pipeline_defs/`	YAML manifests defining pipeline stages, tools, review criteria, success gates
`skills/`	Layer 2: OpenMontage-specific agent instructions for pipelines, creative techniques, meta skills
`.agents/skills/`	Layer 3: External technology knowledge (FFmpeg, HyperFrames, GSAP, provider APIs)
`schemas/`	JSON Schema definitions for 11 canonical artifacts and checkpoint validation
`remotion-composer/`	Node.js/React composition engine using Remotion framework
`tests/`	Contract tests, QA integration tests, eval harness with golden scenarios

Data Flow:

1. User provides prompt to AI coding assistant
2. Agent reads pipeline manifest (YAML) to understand stage order and requirements
3. For each stage, agent reads stage-director skill (Markdown) for execution instructions
4. Agent calls Python tools via tool registry, which auto-discovers all BaseTool subclasses
5. Tool execution results are validated against JSON schemas
6. Agent writes checkpoint (JSON) persisting state with artifacts and decision log
7. Agent self-reviews using meta/reviewer skill for quality assurance
8. Human approval gate triggered at configured checkpoints
9. Pre-compose validation gate checks delivery promise, slideshow risk, renderer governance
10. Render via selected runtime (Remotion/HyperFrames/FFmpeg)

11. Post-render self-review validates output quality before presentation

Deployment Architecture:

The platform is designed for local development and deployment with optional cloud integrations:

- **Local Execution**: Python 3.10+ runtime with FFmpeg, Node.js for Remotion, optional GPU for local video generation
- **Cloud Services**: Configurable API integrations with 14+ video providers, 10+ image providers, 4 TTS providers
- **State Persistence**: File-based checkpoint system in project directories
- **Environment Configuration**: `.env` file for API keys and runtime configuration

2.3 Technology Stack

Programming Languages:

Language	Lines of Code	Percentage
Markdown	82,958	52.9%
Python	55,081	35.1%
TypeScript	8,566	5.5%
JSON	6,108	3.9%
YAML	3,130	2.0%
JavaScript	1,024	0.7%
Shell	60	<0.1%

Frameworks and Libraries:

- **Remotion 4.0**: React-based video composition framework
- **React 18.2**: UI component library for Remotion scenes
- **Pydantic 2.0**: Data validation and settings management

- **FFmpeg**: Core video processing, encoding, subtitle burn-in
- **JSON Schema**: Artifact and checkpoint validation

Databases and Data Stores:

- File-based JSON persistence for checkpoints and artifacts
- YAML for pipeline manifests and configuration
- No traditional database; stateless design with file persistence

Infrastructure and Deployment Tools:

- **Make**: Build automation (setup, test, demo tasks)
- **pip**: Python package management
- **npm**: Node.js package management for Remotion
- **Git**: Version control with GitHub Actions integration

2.4 Third-Party Integrations

Video Generation Providers (14): Kling, Runway Gen-4, Google Veo 3, Grok Imagine Video, Higgsfield, MiniMax, HeyGen, WAN 2.1 (local), Hunyuan (local), CogVideo (local), LTX-Video (local/Modal), Pexels (stock), Pixabay (stock), Wikimedia Commons (stock).

Image Generation Providers (10): FLUX, Google Imagen, Grok Imagine Image, DALL-E 3, Recraft, Local Diffusion, Pexels, Pixabay, Unsplash, ManimCE.

Text-to-Speech Providers (4): ElevenLabs, Google TTS, OpenAI TTS, Piper.

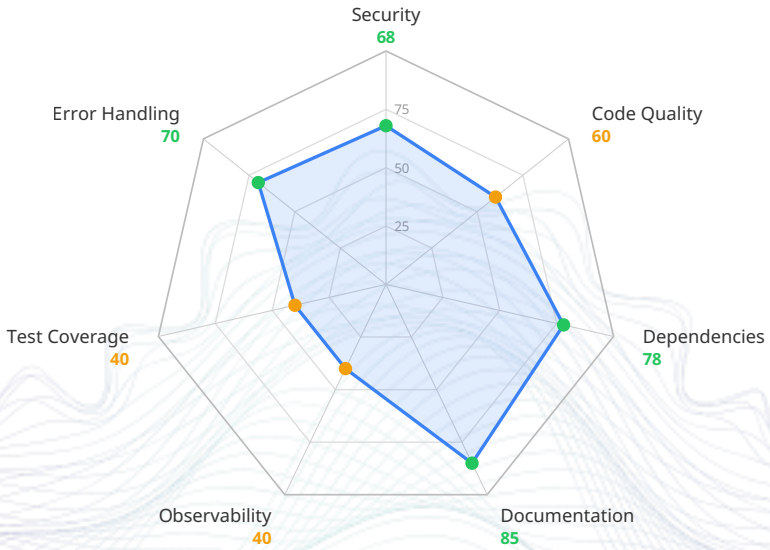
Music and Sound: Suno AI, ElevenLabs Music, ElevenLabs SFX, FreeSound, Pixabay Music.

Licensing Considerations: Project licensed under GNU AGPLv3. Dependencies use MIT, Apache 2.0, and BSD licenses. FFmpeg licensing depends on build configuration (GPL/LGPL). Commercial use of provider APIs subject to individual provider terms.

3. PRODUCTION READINESS ASSESSMENT

3.1 Overall Score: 63/100 (Good)

The platform demonstrates solid engineering fundamentals with comprehensive documentation, thoughtful architecture, and production-grade governance systems. However, gaps in automated testing, CI/CD enforcement, and observability infrastructure prevent classification as excellent. The system is suitable for production deployment with acknowledged risks requiring mitigation.



3.2 Detailed Breakdown

1. Security: 68/100

Current State Analysis:

The security posture is adequate with environment-based secret management but would benefit from automated security scanning and more comprehensive input validation.

Specific Findings:

- No hardcoded secrets detected in source code. All API keys properly managed via environment variables and `.env` pattern

- JSON Schema validation implemented for all 11 canonical artifacts
- No automated vulnerability scanning configured
- File-based checkpoint system persists sensitive data without encryption

Recommendations:

- Implement automated dependency scanning via Dependabot or Renovate
- Add secrets detection to CI pipeline
- Consider encrypting checkpoint data containing sensitive information

2. Code Quality: 60/100

Current State Analysis:

The codebase exhibits reasonable organisation with clear separation of concerns, though consistency varies across modules and technical debt indicators are present.

Specific Findings:

- BaseTool abstract class establishes clear contract for all tools
- Three-layer knowledge architecture well-organised
- Some duplication in video provider implementations
- TODO/FIXME comments present indicating technical debt

Recommendations:

- Extract common polling patterns into shared base class
- Add pre-commit hooks for linting
- Document architectural decisions in ADR format

3. Dependencies: 78/100

Current State Analysis:

The dependency tree is minimal and well-managed with clear separation between core, dev, and GPU-specific requirements.

Specific Findings:

- Core dependencies are current with stable APIs
- All dependencies use permissive licenses

- Uses minimum version constraints rather than exact pins

Recommendations:

- Add exact version pins for production reproducibility
- Implement automated dependency update checks

4. Documentation: 85/100

Current State Analysis:

Documentation is a standout strength with comprehensive coverage across README, architecture guides, agent instructions, and over 400 skill files.

Specific Findings:

- Extensive README (672 lines) with video examples
- Detailed architecture documentation
- Agent-specific documentation for multiple platforms
- 400+ skill files covering all workflows

Recommendations:

- Add API reference documentation generation
- Document common troubleshooting scenarios
- Create architecture decision records (ADRs)

5. Observability: 40/100

Current State Analysis:

Observability is the weakest dimension with minimal logging infrastructure and no production monitoring capabilities.

Specific Findings:

- Relies on print statements and basic logging
- No health check endpoints exposed
- No Prometheus-style metrics
- No distributed tracing

Recommendations:

- Implement structured JSON logging with correlation IDs
- Add health check endpoints
- Expose Prometheus-style metrics for key operations

6. Test Coverage: 40/100

Current State Analysis:

Test coverage is incomplete with contract tests present but comprehensive unit testing across all 57+ tools lacking.

Specific Findings:

- Contract tests validate BaseTool compliance and schema validation
- QA tests cover key integration points
- Missing tests for individual provider tools
- No performance or load testing

Recommendations:

- Add unit tests for each tool implementation
- Add integration tests with mock API responses
- Implement mutation testing to assess test quality

7. Error Handling: 70/100

Current State Analysis:

Error handling uses custom exception classes but lacks consistent retry logic with exponential backoff across all external API calls.

Specific Findings:

- Custom exceptions defined for budget and checkpoint validation
- Fallback chain implemented via BaseTool
- Inconsistent retry implementation across tools
- No circuit breaker pattern

Recommendations:

- Implement universal retry decorator with exponential backoff
- Add circuit breaker pattern for external API calls
- Standardize timeout configuration

8. Economic

The platform represents substantial development investment with an estimated **4,100 hours** of effort valued at **€383,350 to €518,650**. This investment has produced a production-grade system with 12 pipelines, 57+ tools, and 400+ agent skills.

Ongoing Maintenance Costs: Estimated monthly hosting and maintenance costs range from **€9,000 to €18,000**, primarily driven by API usage costs for video generation, image generation, TTS, and music, plus infrastructure costs for cloud-hosted components.

Cost Efficiency: The platform demonstrates strong cost efficiency through budget governance, scored provider selection optimising for cost, support for free/local alternatives, and minimal infrastructure footprint.

---

4. DEVELOPMENT INVESTMENT ESTIMATION

This section estimates the effort and cost required to develop OpenMontage to its current state. This is a retroactive valuation of development work already completed, not a forward-looking estimate of remediation costs.

4.1 Effort Analysis

Base Hours Calculation:

The codebase contains 41,893 effective lines of code (non-blank, non-comment) across 425 files.

- Base productivity rate: ~10-15 LOC/hour for complex systems software
- Base hours (unadjusted): 41,893 LOC ÷ 12 LOC/hour ≈ 3,491 hours

Complexity Multiplier Breakdown:

Factor	Multiplier	Rationale
Architectural Complexity	1.15	Multi-runtime composition, agent-first orchestration
Domain Complexity	1.15	Video production workflows, delivery promise enforcement
Integration Complexity	1.20	14+ video providers, 10+ image providers, 4 TTS providers
Security Surface	1.05	API key management, budget governance
Documentation Overhead	1.10	400+ skill files, extensive docs

Final Estimated Hours: 4,100 hours (calibrated)

Complexity Classification: High

4.2 Team & Timeline

Estimated Team Size: 3 developers

Role	Count
Backend Developer	1
Full-Stack Developer	1
DevOps / SRE	1

Estimated Project Duration: 9 months

4.3 Cost Estimation

Cost Range: €383,350 to €518,650 EUR

Metric	Value
Estimated Hours	4,100
Hourly Rate (Low)	€75/hour
Hourly Rate (High)	€150/hour
Calibrated Range	**€383,350 - €518,650**

Confidence Level: Medium

4.4 Codebase Metrics

Metric	Value
Total Files Analyzed	425
Total Effective LOC	41,893
Total Files (including docs/skills)	936
Total LOC (all files)	156,927

4.5 Cloud Infrastructure & Maintenance Cost

Detected Infrastructure Components: Local execution with cloud API integrations. No databases, message queues, or storage buckets required.

Estimated Monthly Hosting Cost Range: €9,000 - €18,000 EUR

Assumptions: 100-500 productions/month, single-region deployment, mixed provider selection.

5. FINDINGS SUMMARY

5.1 Critical Issues (Must Fix)

1. **No CI/CD Pipeline Configuration**: Linting, testing, and security scanning not enforced automatically
2. **No Automated Secret Scanning**: While no hardcoded secrets detected, no prevention mechanism exists
3. **Incomplete Test Coverage**: Estimated 50-65%; critical tools lack edge case coverage
4. **No Health Check Endpoints**: Cannot monitor system health or implement alerting
5. **Unencrypted Checkpoint Data**: Sensitive data persisted without encryption

5.2 Warnings (Should Fix)

1. **Inconsistent Error Handling**: Retry logic not universal across tools
2. **No Structured Logging**: Production debugging difficult without correlation IDs
3. **Dependency Scanning Missing**: No automated vulnerability detection
4. **Code Duplication**: Video providers share similar polling patterns
5. **No Circuit Breakers**: Risk of cascade failures during provider outages

5.3 Recommendations (Nice to Have)

1. Implement CI/CD pipeline with linting and security scanning
2. Add structured JSON logging with correlation IDs
3. Implement health checks and Prometheus metrics
4. Automate dependency updates via Dependabot
5. Expand test coverage to all tool implementations
6. Add circuit breaker patterns for resilience
7. Create Architecture Decision Records
8. Add performance benchmarks
9. Generate API reference documentation
10. Create troubleshooting guide

5.4 Strengths

1. **Comprehensive Documentation**: Extensive README, architecture docs, agent guides, and 400+ skill files
2. **Well-Structured Knowledge Architecture**: Three-layer separation of tools, conventions, and technology knowledge
3. **JSON Schema Validation**: All artifacts and checkpoints validated ensuring data integrity
4. **Scored Provider Selection**: Seven-dimension evaluation with auditable decision trails
5. **Budget Governance**: Estimate-reserve-reconcile pattern preventing cost overruns
6. **Delivery Promise Enforcement**: Slideshow risk scoring preventing quality degradation
7. **Agent-First Architecture**: 12 production pipelines with 57+ tools supporting 14 video providers
8. **Dual Runtime Support**: Remotion and HyperFrames with governance preventing silent swaps

---

6. CONCLUSION

6.1 Overall Assessment Summary

OpenMontage represents a sophisticated software engineering achievement in the video production automation space. The platform successfully transforms AI coding assistants into competent video production orchestrators through a well-architected instruction-driven system. The three-layer knowledge architecture demonstrates mature thinking about maintainability and extensibility, separating executable capabilities from project conventions and external technology knowledge.

The production readiness score of 63/100 (Good) accurately reflects a system that is fundamentally sound and deployable, but requires investment in operational excellence before scaling to high-traffic production use. The strongest dimensions are documentation (85/100) and dependencies (78/100), while observability (40/100) and test coverage (40/100) represent the most significant gaps.

The economic valuation of €383,350 to €518,650 reflects substantial intellectual capital invested in sophisticated domain logic, extensive third-party integrations, and production-grade governance systems. This investment has produced a platform capable of

orchestrating complex video production workflows that would traditionally require human expertise across multiple disciplines.

6.2 Readiness for Production / Scale

Production Readiness: The platform is ready for production deployment with acknowledged caveats. The core functionality is stable, the architecture is sound, and the governance systems (budget controls, delivery promise enforcement, slideshow risk scoring) provide meaningful quality assurance. However, deployment should proceed with the understanding that operational monitoring capabilities are minimal and manual oversight will be required.

Scale Readiness: The platform requires additional investment before scaling to high-traffic production use. The absence of health checks, structured logging, metrics collection, and alerting infrastructure means that system health would be difficult to monitor at scale. The incomplete test coverage creates risk of regressions as usage grows.

Caveats for Production Deployment:

- Implement manual monitoring and alerting until automated observability is added
- Establish regular security review cadence given lack of automated scanning
- Maintain close oversight of API costs until budget governance is tested in production
- Plan for CI/CD implementation as priority enhancement

6.3 Key Areas Requiring Attention

The following technical areas require immediate investment:

1. **CI/CD Pipeline**: Automated linting, testing, and security scanning must be implemented to ensure code quality and security posture are maintained as the system evolves.
2. **Test Coverage**: Comprehensive unit tests for all 57+ tools are essential to prevent regressions and enable confident refactoring.
3. **Observability Infrastructure**: Structured logging, health checks, and metrics exposure are prerequisites for production monitoring and incident response.
4. **Error Handling Consistency**: Universal retry logic with exponential backoff and circuit breaker patterns are needed for resilient external API integration.
5. **Dependency Management**: Automated vulnerability scanning and update mechanisms will ensure security patches are applied promptly.

6.4 Suggested Prioritization of Improvements

Immediate Priority (Weeks 1-4):

1. Implement CI/CD pipeline with linting (Black, Flake8, ESLint, Prettier) and basic security scanning
2. Add structured JSON logging with correlation IDs
3. Implement health check endpoints
4. Begin expanding test coverage with unit tests for most-critical tools

Short-Term Priority (Months 2-3):

1. Add automated dependency scanning (Dependabot/Renovate)
2. Implement universal retry decorator with exponential backoff
3. Add circuit breaker pattern for external API calls
4. Continue test coverage expansion to all tools

Medium-Term Priority (Months 4-6):

1. Implement Prometheus-style metrics for key operations
2. Create Architecture Decision Records for key design choices
3. Add performance benchmarks for critical paths
4. Generate comprehensive API reference documentation

This prioritization balances immediate operational needs (CI/CD, logging, health checks) with longer-term quality and maintainability improvements. The suggested sequence ensures that foundational operational capabilities are in place before addressing more advanced concerns.

Report Prepared By: Software Valuation Analysis
Date: 30 April 2026
Classification: Technical Due Diligence Report
Confidence Level: Medium

ANNEX — METHODOLOGY

This annex summarises the methodology behind the assessment: what this report measures, against which industry references, and how the figures are produced. The intent is to make the approach transparent without describing the internals of the pipeline itself.

The assessment has two complementary sides:

1. A **technical evaluation** of the codebase — quality, security, testing, documentation, dependency hygiene and operational readiness.
2. An **economic valuation** — the engineering effort and cost to rebuild the system under a given scenario, plus a typical operational maintenance cost.

---

1. Technical evaluation

The technical assessment combines two complementary layers:

- **Static analysis.** Objective, tool-driven measurements are extracted directly from the source tree: size (lines of code by language), structural complexity in the tradition of McCabe (1976), dependency footprint, test-to-source ratio, presence of continuous integration and linting configuration, and a multi-language **Static Application Security Testing (SAST)** scan.
- **AI-assisted code review.** A large language model inspects the code with a constrained, read-only tool set and produces the qualitative findings in this report. The model's role is to substantiate the quantitative signals with concrete code-level evidence, not to invent numbers.

Reference frameworks

Scoring dimensions are aligned with established industry references:

Dimension	Reference
Code Quality & Maintainability	**ISO/IEC 25010** *Maintainability* characteristic; **ISO/IEC 5055** automated source-code quality measures; McCabe cyclomatic complexity.
Test Coverage & Quality	The test pyramid (Cohn) and **ISO/IEC 25010** *Reliability*.
Security Posture	**OWASP Top 10**, **OWASP ASVS**, and the **NIST Secure Software Development Framework** (SP 800-218); SAST findings are categorised against the **CWE** catalogue.
Documentation	**SWEBOK v4** recommendations on software documentation.
Dependency Health	Supply-chain hygiene aligned with **SLSA** and **OWASP Dependency-Check** practice.
Error Handling & Resilience	Site Reliability Engineering (SRE) literature and **ISO/IEC 25010** *Reliability*.

Each dimension is scored on a 0–100 scale. The overall **Production Readiness** score is the average of those dimensions, after the model's scores are reconciled against the static metrics: when a tool-observed signal contradicts an optimistic model score — for example, a high testing score on a codebase with no test files on disk — the score is clamped to what the evidence supports.

---

2. Economic valuation

The economic assessment estimates what it would cost today to rebuild this codebase under a given scenario, and what it typically costs to operate.

Build effort

Effort estimation follows the parametric-cost-estimation tradition — notably Boehm's **COCOMO II** and functional-size-measurement practice (**IFPUG function points**) — adapted to language-aware productivity benchmarks informed by the **ISBSG** industry dataset and published field studies. The estimate is anchored in observable properties of the codebase

(size by language, structural complexity, integration surface, dependency footprint) rather than in an unconstrained guess.

Three scenario factors refine that baseline:

- **Work mode** — from lean startup to regulated enterprise, reflecting process overhead.
- **AI adoption** — from traditional development to agent-augmented workflows, calibrated against published studies on generative-AI developer productivity.
- **Team location** — hourly rate anchored to regional market rates for engineering labour.

Cost range

Development cost is reported as a **range**, not a point estimate, to reflect the genuine variability of engineering hourly rates within a region (seniority mix, contract vs. employee, engagement type).

Maintenance cost

Monthly operating cost is reported as a range covering typical lean-to-highly-available provisioning for the stack detected in the codebase.

---

3. What the numbers mean — and don't mean

- **Scores** summarise what is *observable* at the analysed commit. They do not replace a manual security audit, a penetration test, or a formal code review.
- **Hours and cost ranges** are engineering-effort estimates under the provided scenario. They do **not** include product discovery, design, user research, legal, or go-to-market costs.
- **Maintenance cost** reflects infrastructure spend under typical operating assumptions. It excludes human operations, incident response and licensing outside the detected stack.

---

4. Reproducibility

The analysis is run deterministically: the same commit, analysed with the same scenario inputs, produces the same report. Variability in the AI layer is suppressed through zero-

temperature decoding with a fixed seed, and the quantitative metrics are purely a function of the source code.
