LibVulnWatch: A Deep Assessment Agent System and Leaderboard for Uncovering Hidden Vulnerabilities in Open-Source AI Libraries

Zekun Wu[1,2*] Seonglae Cho[1,2*] Umar Mohammed[1] Cristian Munoz[1]
Kleyton Costa[1] Xin Guan[1] Theo King[1] Ze Wang[1,2]
Emre Kazim[1,2] Adriano Koshiyama[1,2†]
[1]Holistic AI [2]University College London

Abstract

Open-source AI libraries are foundational to modern AI systems, yet they present significant, underexamined risks spanning security, licensing, maintenance, supply chain integrity, and regulatory compliance. We introduce LibVulnWatch, a system that leverages recent advances in large language models and agentic workflows to perform deep, evidence-based evaluations of these libraries. Built on a graph-based orchestration of specialized agents, the framework extracts, verifies, and quantifies risk using information from repositories, documentation, and vulnerability databases. LibVulnWatch produces reproducible, governance-aligned scores across five critical domains, publishing results to a public leaderboard for ongoing ecosystem monitoring. Applied to 20 widely used libraries—including ML frameworks, LLM inference engines, and agent orchestration tools—our approach covers up to 88% of OpenSSF Scorecard checks while surfacing up to 19 additional risks per library, such as critical RCE vulnerabilities, missing SBOMs, and regulatory gaps. By integrating advanced language technologies with the practical demands of software risk assessment, this work demonstrates a scalable, transparent mechanism for continuous supply chain evaluation and informed library selection.

1 Introduction

The rapid adoption of AI systems in high-stakes domains has intensified the need for robust technical governance and risk assessment. While policy frameworks increasingly call for transparency, accountability, and safety, a persistent gap remains between these governance objectives and the engineering practices required to realize them (Reuel et al., 2025). Open-source libraries and frameworks, which underpin most modern machine learning systems, introduce complex legal, security, maintenance, and regulatory risks that are often overlooked by conventional assessment tools (Wang et al., 2025; Alevizos et al., 2024). These tools typically provide surface-level checks and lack the depth needed to uncover nuanced vulnerabilities in the AI software supply chain.

Recent progress in large language models and agentic workflows has enabled new approaches to structured, evidence-based analysis across diverse domains. In this work, we introduce LibVulnWatch, a system that leverages these advances to perform deep, multi-domain evaluations of open-source AI libraries. The system coordinates specialized agents to assess five critical risk domains—licensing, security, maintenance, dependency management, and regulatory compliance—drawing on verifiable evidence from repositories, advisories, and documentation.

To enable continuous ecosystem monitoring and evidence-based decision-making, we publish every assessment on a public leaderboard[1]. Evaluating 20 widely used AI libraries—including ML frameworks, inference engines, and agent orchestration tools—LibVulnWatch demonstrates:

- Up to **88% coverage** of OpenSSF Scorecard checks;
- **Up to 19 additional risks** per library, including RCEs, missing SBOMs, and compliance gaps;
- **Governance-aligned, reproducible scores** for transparent comparison and risk management.

By integrating advanced language technologies with the practical demands of software risk assessment, LibVulnWatch offers a scalable, transparent mechanism for operationalizing governance principles in open-source AI infrastructure.

*Equal contributions
†Corresponding author

[1]The leaderboard and all per-library assessment reports are publicly available on Hugging Face at `https://huggingface.co/spaces/holistic-ai/LibVulnWatch`.

2 Related Work

Research on vulnerabilities in AI pipelines has expanded beyond adversarial inputs and data poisoning to encompass system-level risks in the software supply chain (Wang et al., 2025). Studies have analyzed large-scale LLM supply chain issues, revealing flaws in application and serving components, while others have documented recurring bugs in widely used frameworks such as TensorFlow and PyTorch (Chen et al., 2023). LLM-based vulnerability detection has shown promise for code analysis (Zhou et al., 2024), though challenges such as false positives and domain adaptation remain. Broader supply chain threats—including dependency confusion and package hijacking—are well-documented (Ladisa et al., 2023; Ohm et al., 2020).

Efforts to assess open-source project hygiene, such as the OpenSSF Scorecard (Zahan et al., 2023), provide valuable surface metrics but often lack the depth required for comprehensive vulnerability analysis. Recent advances in multi-agent orchestration frameworks, including LangChain and LangGraph (LangChain AI, 2025a,b), have enabled more structured and scalable approaches to information extraction and evaluation, forming the basis for several assessment pipelines.

3 Methodology

Our approach leverages recent advances in language models and multi-agent systems to address complex challenges in software risk assessment. By adapting NLP techniques for information extraction, knowledge synthesis, and structured reasoning, we operationalize key Technical AI Governance capacities through a multi-stage evaluation pipeline. This section details the pipeline's architecture, risk assessment framework, evaluation protocol, and benchmarking procedures.

3.1 Risk Assessment Framework

We define a comprehensive risk assessment framework adapted from established open-source and AI risk taxonomies. It encompasses five governance-relevant domains, each with specific factors for evaluation:

- **License Analysis:** Assessing license type (e.g., MIT, Apache 2.0, GPL), version, commercial use compatibility, distribution rights, patent grant provisions, attribution requirements, and overall conformance with open-source compliance standards.
- **Security Assessment:** Evaluating known Common Vulnerabilities and Exposures (CVEs) within the last 24 months (count and severity), the existence and adequacy of a security disclosure policy, responsiveness to security issues, evidence of security testing (e.g., CI/CD test coverage), and the handling of released binaries or signed artifacts.
- **Maintenance Indicators:** Analyzing release frequency and the date of the latest release, the number and activity levels of contributors (including diversity and organizational backing), issue resolution metrics (e.g., response times, recent commit activity), and the project governance model and packaging workflow.
- **Dependency Management:** Examining Software Bill of Materials (SBOM) availability and format (e.g., CycloneDX, SPDX), direct and transitive dependency counts, policies and tools for dependency updates, and the identification of known vulnerable dependencies.
- **Regulatory Considerations:** Reviewing documentation for alignment with relevant compliance frameworks (e.g., GDPR, AI Act), the availability of explainability features (especially for AI/ML libraries), stated data privacy provisions, and the presence of audit documentation or support for audit readiness.

Each of these five domains, as depicted as parallel tracks at the top of Figure 1, is operationalized as a distinct assessment module within the agentic workflow, guided by engineered prompts enforcing key concept coverage and quantifiable metric extraction.

3.2 Agentic Workflow

Our system employs a structured, agentic workflow implemented as a DAG using a modern agent orchestration framework. Our implementation was inspired by the Open Deep Research repository[2]. We redesigned the graph design and defined domain-specific prompts that adapt language model capabilities to the specific knowledge requirements of security, licensing, and compliance assessment. All experiments used `gpt-4.1-mini` (costing approx. $0.10 per library). OpenSSF Scorecard (Zahan et al., 2023) checks were run on the primary GitHub repository of each target library, and we used the Google Search API for evidence retrieval.

[2]https://github.com/langchain-ai/open_deep_research

The automated workflow addresses particular challenges of applying language models to evidence-based assessment, including factuality verification and domain-specific knowledge extraction. It begins with high-level search-based planning, followed by domain-specific iterative retrieval until sufficient evidence is gathered for each of the five domains. These are processed in parallel to generate draft findings, which are combined into a full report including an executive summary. The report is then validated by identifying the main GitHub repository, running the Scorecard, and comparing outputs using an LLM. This approach ensures modularity, consistency, and parallelism. Integrating the LLM's text understanding, structured data handling, and search capabilities, the overall agentic workflow is illustrated in Figure 1 and comprises the following key stages:

- **Planning:** An initial *Assessment Planner* agent (top of Figure 1) generates a detailed assessment plan for the target library, adhering strictly to the five core risk domains detailed in Section 3.1 and formulates initial research queries.
- **Iterative Evidence Gathering and Drafting (Per Domain):** For each of the five risk domains, operating in parallel:
  - *Query Generation:* Targeted search queries are formulated.
  - *Evidence Retrieval:* A dedicated agent iteratively performs searches against authoritative sources (e.g., official documentation, security databases, repository metadata using specialized query patterns via Search API / Local RAG) to aggregate evidence. This includes the use of advanced search operators and repository-specific query patterns (e.g., for GitHub) to extract structured data and metrics where direct API access is not assumed.
  - *Draft Findings:* The retrieved evidence is synthesized into initial draft findings for the specific domain.
  - *Quality Check & Refinement Loop:* A quality check (QC) assesses if sufficient evidence has been gathered and if the findings meet predefined criteria. If the QC is not passed and the maximum search depth (k) has not been reached, the process loops back to generate refined queries and retrieve more evidence. This iterative loop continues until the QC is passed or the depth limit is reached.

  This entire synthesis process is strictly governed by prompts engineered to adapt language understanding capabilities to the software security context, enforcing structured reporting (e.g., with sections for an executive overview, emergency issues, and a detailed table of findings with columns for Risk Factor, Observed Data, Rating, Reason for Rating, and Key Control), quantification, evidence citation, and handling of missing information. The specific instruction sets (prompts) used for each key agent are detailed in Appendix A.2.
- **Synthesis & Report Compilation:** Once drafting for all domains is complete (marked as Done in Figure 1), a final agent synthesizes the individual domain findings into a consolidated, structured report. This includes an executive summary, a risk dashboard, highlighted emergency issues, prioritized controls, and a mitigation strategy.
- **Benchmark Validation:** Before final publication, the generated report undergoes a validation step. This involves identifying the main repository of the target library, running the OpenSSF Scorecard, and comparing the Scorecard output with the agentś report (often using an LLM for an *Archive Evaluation*) to assess alignment and novelty, as depicted in Figure 1.
- **Public Reporting and Ecosystem Monitoring:** The validated and finalized report is programmatically published to a public leaderboard, which is implemented as an interactive Gradio application hosted on Hugging Face Spaces (see Appendix A.1 for details and screenshots). This facilitates *Ecosystem Monitoring* and accountability by dynamically ranking libraries by Trust Score and highlighting key risks. We follow responsible disclosure practices for any new, non-public vulnerabilities identified during the assessment.

3.3 Evaluated AI Libraries

We evaluated 20 diverse open-source AI libraries spanning the AI lifecycle, selected for representative coverage (see Table 1 for list and scores). Libraries were chosen from three key functional categories, aiming for diversity in function, community size, maturity, and impact:

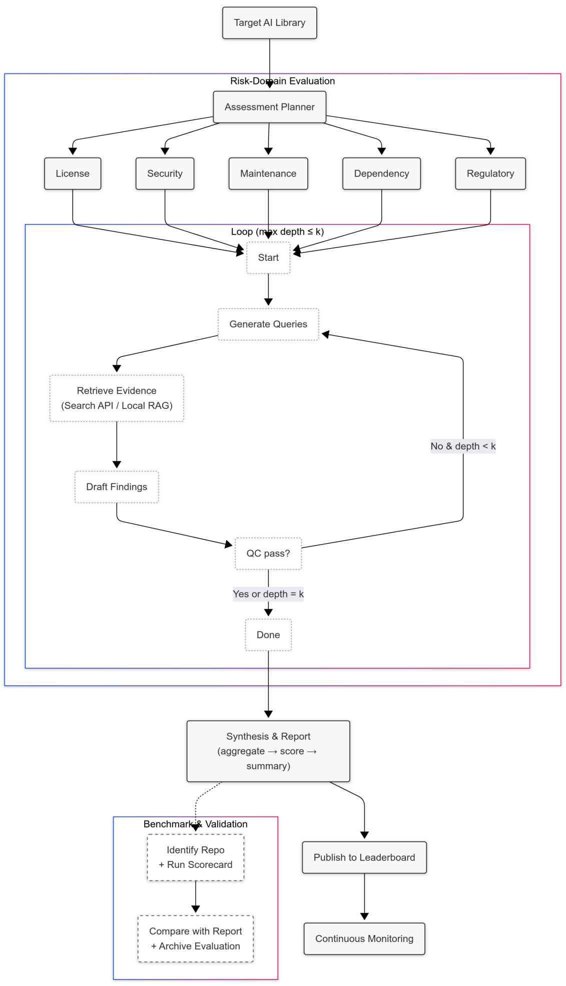


Figure 1: Workflow of the automated agent. Each risk domain (License, Security, Maintenance, Dependency, Regulatory) runs in parallel, with controlled-depth evidence retrieval and drafting. The results are synthesized into a report, benchmarked using the OpenSSF Scorecard, and then published with monitoring.

- **Core ML/DL Frameworks:** PyTorch (Paszke et al., 2019), TensorFlow (Abadi et al., 2016), ONNX (ONNX, 2025), Huggingface Transformers (Wolf et al., 2020), and JAX (Bradbury et al., 2025).
- **LLM Inference & Orchestration Tools:** TensorRT (NVIDIA, 2025), LlamaIndex (Liu, 2022), SGLang (Zheng et al., 2023), vLLM (Kwon et al., 2023), LangChain (LangChain AI, 2025a), and Text Generation Inference (Hugging Face, 2025).
- **AI Agent Frameworks:** Browser Use (Müller and Žunič, 2024), CrewAI (CrewAI, 2025), MetaGPT (Zhang and colleagues, 2024), LangGraph (LangChain AI, 2025b), SmolAgents (Roucher et al., 2025), Stagehand (Browserbase, 2025), Composio (Composio, 2025), Pydantic AI (Pydantic, 2025), and Agent Development Kit (Google, 2025).

Each library underwent the full protocol; results are public.

3.4 Risk Scoring

We employ a 1-5 numerical scale for risk rating within each of the five governance-relevant domains outlined above (Section 3.1), where 1 indicates High Risk, 3 Medium Risk, and 5 Low Risk. As detailed in the workflow description (Section 3.2), each rating requires justification tied to specific, verifiable evidence thresholds defined in the prompts. **The risk scoring within each domain is anchored by the following criteria derived from the agent system prompts:**

- **Low Risk (Score 5)** is indicated by: *License:* Permissive (e.g., MIT, Apache 2.0, BSD) with clear terms and compatibility; *Security:* No CVEs in the past 24 months, a robust security policy, and rapid fixes (e.g., <7 days); *Maintenance:* More than 10 active contributors, monthly or more frequent releases, and prompt issue response (e.g., <24 hours); *Dependencies:* SBOM available, fewer than 20 direct dependencies, and evidence of automatic updates; *Regulatory:* Clear compliance documentation and a complete audit trail.
- **Medium Risk (Score 3)** is indicated by: *License:* Moderate restrictions or unclear patent provisions; *Security:* 1-3 minor CVEs in the past 12 months, a basic security policy, and moderate response times (e.g., 7-30 days); *Maintenance:* 3-10 active contributors, quarterly releases, and issue response times of 1-7 days; *Dependencies:* Partial SBOM, 20-50 direct dependencies, and some transitive visibility; *Regulatory:* Incomplete compliance documentation or partial audit readiness.
- **High Risk (Score 1)** is indicated by: *License:* Restrictive terms (e.g., GPL/AGPL), incompatible terms, or other legal concerns; *Security:* Critical or multiple CVEs, a missing security policy, or slow response times (e.g., >30 days); *Maintenance:* Fewer than 3 active contributors, infrequent releases (e.g., >6 months), or poor issue response; *Dependencies:* No SBOM, more than 50 direct dependencies, or known vulnerable transitive dependencies; *Regulatory:* Missing compliance documentation or failure to meet essential regulations.

Critically, the absence of necessary information for assessment (e.g., no public security policy or SBOM) on any key risk factor is also explicitly defined as a High Risk indicator (Score 1). Furthermore, the system is designed to critically evaluate all available information to identify the most significant or concerning risk factor within each domain, even if other factors appear satisfactory, ensuring a thorough and conservative risk posture. Intermediate scores (2 or 4) may be assigned based on the agent's assessment when evidence suggests a risk level between these defined thresholds. The overall Trust Score provides a composite measure by aggregating the five domain scores (Li, Se, Ma, De, Re): $\text{Trust}(l) = \frac{1}{5}\sum_{d\in\{Li,Se,Ma,De,Re\}} d(l)$.

3.5 Benchmarking and Novelty Analysis

We use the OpenSSF Scorecard (Zahan et al., 2023) as a baseline to evaluate our agent. This involves identifying the main repository, running the Scorecard, and comparing its output with our agentś report to derive two key metrics:

- **Baseline Alignment(%)**: The percentage of relevant Scorecard checks addressed in the agentś report, calculated against applicable checks (i.e., excluding checks with non-conclusive scores such as ?́)́ from the Scorecard output. This is calculated as $\text{Coverage} = \frac{\text{\# matched checks}}{\text{\# applicable checks}} \times 100$.
- **Novelty Yield (#)**: The number of unique, meaningful issues or deeper contextual insights identified by the agent but not explicitly surfaced by the Scorecard. This is defined as $\text{Yield} = \text{\# unique agent-only findings}$.

4 Results

Our methodology identified novel vulnerabilities in Open-source AI libraries, often missed by static analysis. Benchmarking against OpenSSF Scorecard (Zahan et al., 2023), detailed in Section 4.1, quantified alignment and unique contextual findings. Section 4.3 presents illustrative examples. For a detailed example of a full assessment output (the analysis report) for the JAX library, please see Appendix A.3; its corresponding baseline evaluation is presented in Appendix A.4.

4.1 Benchmarking and Alignment Analysis

We benchmarked our agentic system against the OpenSSF Scorecard to evaluate alignment and identify unique contributions. Table 1 presents key metrics defined in Section 3—Baseline Alignment (overlap with Scorecard checks) and Novelty Yield (unique findings)—across all evaluated libraries, grouped by functional category and including category averages. While observed Baseline Alignment for most libraries ranged from 55% to 88%, indicating substantial overlap, the agentic system consistently surfaced a significant Novelty Yield (typically 5-13 unique findings per library) not captured by baseline tools.

The agents showed particular strengths in connecting disparate information sources and contextualizing findings, though they sometimes missed formal contributor declarations, CI testing evidence, binary artifact identification, and explicit security testing policies flagged by the baseline. This suggests opportunities for complementary approaches combining structured checks with context-aware reasoning. Examples of critical risks identified through contextual analysis that went beyond conventional automated scans, contributing to Novelty Yield, include:

- Complex RCEs from insecure defaults or subtle data processing flaws.
- Systemic SBOM absence and supply chain/-transitive dependency risks.
- Pervasive regulatory/privacy compliance gaps (GDPR, HIPAA, AI Act).
- Widespread lack of governance mechanisms (audit trails, explainability, privacy controls).
- Undocumented telemetry/data collection (e.g., in one AI agent framework).
- Potential patent risks from unclear/insufficient licensing for core ML algorithms.

4.2 Aggregated Domain Risk Findings and Patterns

Table 2 presents the detailed library-by-library trust scores across the five primary domains and the composite Trust Score. The context-sensitive analysis enabled by our approach revealed nuanced patterns across evaluated libraries that would be difficult to detect with traditional rules-based assessment. Aggregate Trust Scores varied by category, with Core ML/DL frameworks generally scoring higher than newer AI Agent frameworks, potentially reflecting greater maturity. Common weaknesses were observed across the ecosystem, particularly in:

- **Dependency Management:** Widespread absence of SBOMs hindering transparency, poorly managed transitive dependencies, and lack of automated vulnerability scanning were

Table 1: Baseline Alignment and Novelty Yield Across Libraries

Library	**Baseline Alignment (%)**	**Novelty Yield (#)**
Core ML/DL Frameworks	***77.1***	*6.8*
PyTorch	88.2	8
JAX	61.1	12
Tensorflow	72.2	5
ONNX	87.5	5
Huggingface Transformers	76.5	4
LLM Inference & Orchestration	*73.7*	*7.8*
TensorRT	68.8	5
LlamaIndex	82.4	7
SGLang	73.3	5
vLLM	73.3	7
LangChain	72.2	19
Text Generation Inference	72.2	6
AI Agent Frameworks	*76.2*	***9.1***
Browser Use	88.2	7
CrewAI	71.4	13
MetaGPT	57.1	7
LangGraph	77.8	7
SmolAgents	73.3	9
Stagehand	83.3	6
Composio	68.8	5
Pydantic AI	88.2	10
Agent Development Kit	77.9	7

Table 2: Detailed Risk Assessment Scores Across Libraries and Domains (Li: License, Se: Security, Ma: Maintenance, De: Dependency, Re: Regulatory, Trust: Trust Score; Scale: 1-5, higher is better)

Library	**Li**	**Se**	**Ma**	**De**	**Re**	**Trust**
Core ML/DL Frameworks						*13.0*
PyTorch	5	1	3	1	3	13
JAX	5	3	4	1	1	**14**
Tensorflow	5	1	3	1	3	13
ONNX	5	1	3	1	1	11
Transformers	5	1	4	1	3	**14**
LLM Inference & Orchestration						*11.8*
TensorRT	5	1	5	1	3	**15**
LlamaIndex	5	1	3	1	3	13
SGLang	5	1	3	1	1	11
vLLM	3	1	4	1	1	10
LangChain	5	1	1	1	3	11
Text Generation Inference	5	1	3	1	1	11
AI Agent Frameworks						*11.4*
CrewAI	5	1	3	1	1	11
MetaGPT	5	1	5	1	1	13
LangGraph	1	1	3	1	3	9
SmolAgents	5	1	1	1	1	9
Stagehand	5	3	1	1	1	11
Composio	1	1	5	1	3	11
Browser Use	5	1	4	1	3	**14**
Pydantic AI	5	1	3	1	1	11
Agent Development Kit	5	3	4	1	1	**14**

common.

- **Regulatory Considerations:** Significant gaps existed regarding comprehensive documentation for GDPR/HIPAA/AI Act compliance and features for model explainability or audit logging.
- **Security:** Many libraries exhibited vulnerabilities like RCEs, unsigned releases, and insecure CI/CD pipelines, with newer frameworks often lacking mature disclosure policies.
- **License Analysis:** While often permissive, nuanced risks like potential patent issues or conflicts with restrictive licenses (e.g., AGPL) were found, and formal patent grants were frequently missing.
- **Maintenance Indicators:** Established libraries showed robust core maintenance, but patterns of unmaintained sub-projects or less transparency/slower resolution in newer frameworks posed risks.

4.3 Illustrative Case Studies

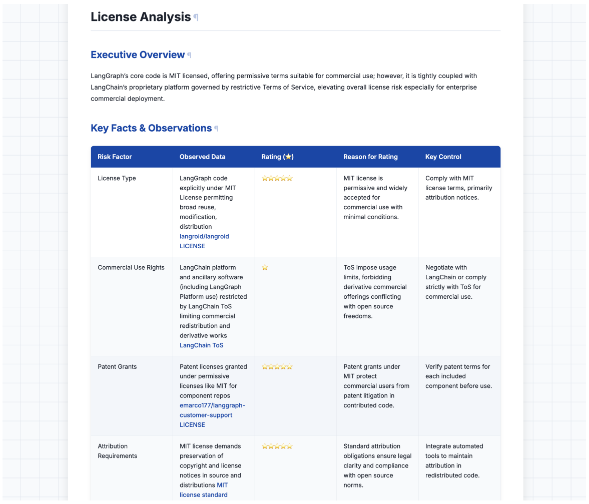
To further illustrate the capabilities of LibVulnWatch, we present five case studies highlighting how semantic understanding and contextual analysis revealed insights that would be challenging to capture through traditional assessment approaches.

License Analysis: LangGraph

Our system identified that while LangGraph specifies an MIT license in its repository, a more comprehensive analysis revealed connections to LangChainś Terms of Use that potentially affect its licensing status. By understanding semantic relationships between documentation sources and interpreting licensing implications, the system provided a more holistic assessment than tools like the OpenSSF Scorecard, which primarily consider repository-level licensing information (see Figure 2).

Regulatory Considerations: Browser Use

For the Browser Use library, designed for web interaction tasks, LibVulnWatch linked its characteristics to emerging requirements under the EU AI Act. The systemś ability to connect library functionality with regulatory frameworks enabled it to identify needs for clear documentation regarding data handling, agent capabilities, and potential risks, which are critical for compliance with high-risk AI system regulations (summarized in Figure 3). This showcases the value of language



License Analysis ¶

Executive Overview ¶

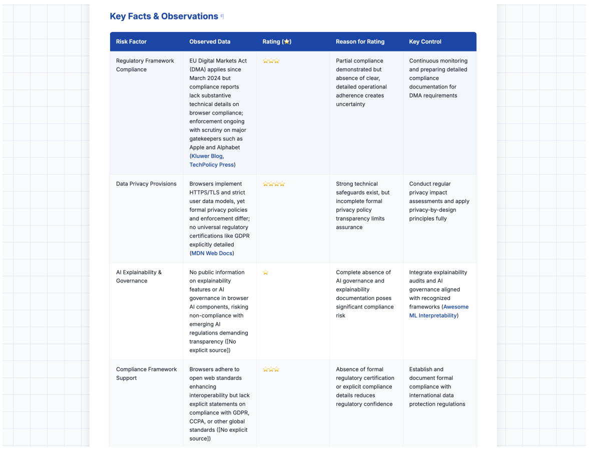
LangGraph's core code is MIT licensed, offering permissive terms suitable for commercial use; however, it is tightly coupled with LangChain's proprietary platform governed by restrictive Terms of Service, elevating overall license risk especially for enterprise commercial deployment.

Key Facts & Observations ¶

Risk Factor	Observed Data	Rating (⭐)	Reason for Rating	Key Control
License Type	LangGraph code explicitly under MIT License permitting broad reuse, modification, distribution langroid/langroid LICENSE	⭐⭐⭐⭐⭐	MIT license is permissive and widely accepted for commercial use with minimal conditions.	Comply with MIT license terms, primarily attribution notices.
Commercial Use Rights	LangChain platform and ancillary software (including LangGraph Platform use) restricted by LangChain ToS limiting commercial redistribution and derivative works LangChain ToS	⭐	ToS impose usage limits, forbidding derivative commercial offerings conflicting with open source freedoms.	Negotiate with LangChain or comply strictly with ToS for commercial use.
Patent Grants	Patent licenses granted under permissive licenses like MIT for component repos emarco177/langgraph-customer-support LICENSE	⭐⭐⭐⭐⭐	Patent grants under MIT protect commercial users from patent litigation in contributed code.	Verify patent terms for each included component before use.
Attribution Requirements	MIT license demands preservation of copyright and license notices in source and distributions MIT license standard	⭐⭐⭐⭐⭐	Standard attribution obligations ensure legal clarity and compliance with open source norms.	Integrate automated tools to maintain attribution in redistributed code.

Figure 2: LangGraph License Analysis from the Generated Report, highlighting potential complexities arising from related Terms of Use.

understanding in assessing alignment with evolving regulatory landscapes.



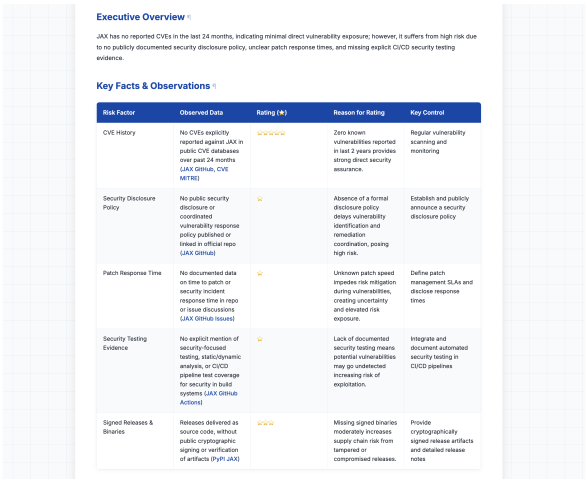
Key Facts & Observations ¶

Risk Factor	Observed Data	Rating (⭐)	Reason for Rating	Key Control
Regulatory Framework Compliance	EU Digital Markets Act (DMA) applies since March 2024 but compliance reports lack substantive technical details on browser compliance; enforcement ongoing with scrutiny on major gatekeepers such as Apple and Alphabet (Kluwer Blog, TechPolicy Press)	⭐⭐⭐	Partial compliance demonstrated but absence of clear, detailed operational adherence creates uncertainty	Continuous monitoring and preparing detailed compliance documentation for DMA requirements
Data Privacy Provisions	Browsers implement HTTPS/TLS and strict user data models, yet formal privacy policies and enforcement differ; no universal regulatory certifications like GDPR explicitly detailed (MDN Web Docs)	⭐⭐⭐⭐	Strong technical safeguards exist, but incomplete formal privacy policy transparency limits assurance	Conduct regular privacy impact assessments and apply privacy-by-design principles fully
AI Explainability & Governance	No public information on explainability features or AI governance in browser AI components, risking non-compliance with emerging AI regulations demanding transparency ([No explicit source])	⭐	Complete absence of AI governance and explainability documentation poses significant compliance risk	Integrate explainability audits and AI governance aligned with recognized frameworks (Awesome ML Interpretability)
Compliance Framework Support	Browsers adhere to open web standards enhancing interoperability but lack explicit statements on compliance with GDPR, CCPA, or other global standards ([No explicit source])	⭐⭐⭐	Absence of formal regulatory certification or explicit compliance details reduces regulatory confidence	Establish and document formal compliance with international data protection regulations

Figure 3: Browser Use Regulatory Analysis from the Generated Report, connecting library features to EU AI Act considerations.

Security Analysis: JAX

In the domain of security, LIBVULNWATCH correctly identified that the JAX library had no reported CVEs for the past two years. More importantly, through semantic analysis of GitHub Action links and repository structure, the system highlighted that JAX lacks an explicit, dedicated security Continuous Integration (CI) workflow, a subtle but important finding for long-term security posture that requires reasoning beyond simple pattern matching (Figure 4).



Executive Overview ¶

JAX has no reported CVEs in the last 24 months, indicating minimal direct vulnerability exposure; however, it suffers from high risk due to no publicly documented security disclosure policy, unclear patch response times, and missing explicit CI/CD security testing evidence.

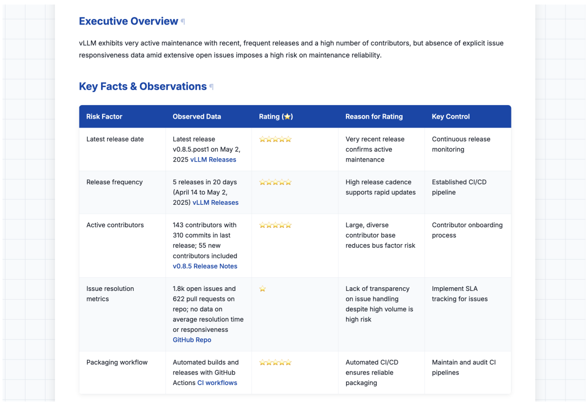
Key Facts & Observations ¶

Risk Factor	Observed Data	Rating (⭐)	Reason for Rating	Key Control
CVE History	No CVEs explicitly reported against JAX in public CVE databases over past 24 months (JAX GitHub, CVE MITRE)	⭐⭐⭐⭐⭐	Zero known vulnerabilities reported in last 2 years provides strong direct security assurance.	Regular vulnerability scanning and monitoring
Security Disclosure Policy	No public security disclosure or coordinated vulnerability response policy published or linked in official repo (JAX GitHub)	⭐	Absence of a formal disclosure policy delays vulnerability identification and remediation coordination, posing high risk.	Establish and publicly announce a security disclosure policy
Patch Response Time	No documented data on time to patch or security incident response time in repo or issue discussions (JAX GitHub Issues)	⭐	Unknown patch speed impedes risk mitigation during vulnerabilities, creating uncertainty and elevated risk exposure.	Define patch management SLAs and disclose response times
Security Testing Evidence	No explicit mention of security-focused testing, static/dynamic analysis, or CI/CD pipeline test coverage for security in build systems (JAX GitHub Actions)	⭐	Lack of documented security testing means potential vulnerabilities may go undetected increasing risk of exploitation.	Integrate and document automated security testing in CI/CD pipelines
Signed Releases & Binaries	Releases delivered as source code, without public cryptographic signing or verification of artifacts (PyPI JAX)	⭐⭐⭐	Missing signed binaries moderately increases supply chain risk from tampered or compromised releases.	Provide cryptographically signed release artifacts and detailed release notes

Figure 4: JAX Security Analysis from the Generated Report, noting absence of CVEs but also lack of explicit security CI.

Maintenance Analysis: vLLM

For vLLM, an LLM inference and serving library, the system analyzed recent GitHub contributions, issue resolution times, and release frequency to assess its maintenance trends. By extracting and synthesizing temporal patterns from repository metadata, the system provided a quantitative overview of project activity, as shown in Figure 5, demonstrating how language models can integrate structured data analysis with contextual understanding.



Executive Overview ¶

vLLM exhibits very active maintenance with recent, frequent releases and a high number of contributors, but absence of explicit issue responsiveness data amid extensive open issues imposes a high risk on maintenance reliability.

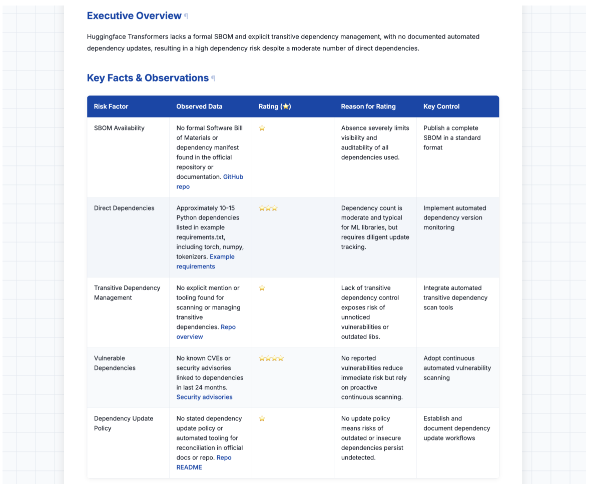
Key Facts & Observations ¶

Risk Factor	Observed Data	Rating (⭐)	Reason for Rating	Key Control
Latest release date	Latest release v0.8.5.post1 on May 2, 2025 vLLM Releases	⭐⭐⭐⭐⭐	Very recent release confirms active maintenance	Continuous release monitoring
Release frequency	5 releases in 20 days (April 14 to May 2, 2025) vLLM Releases	⭐⭐⭐⭐⭐	High release cadence supports rapid updates	Established CI/CD pipeline
Active contributors	143 contributors with 310 commits in last release; 55 new contributors included v0.8.5 Release Notes	⭐⭐⭐⭐⭐	Large, diverse contributor base reduces bus factor risk	Contributor onboarding process
Issue resolution metrics	1.8k open issues and 622 pull requests on repo; no data on average resolution time or responsiveness GitHub Repo	⭐	Lack of transparency on issue handling despite high volume is high risk	Implement SLA tracking for issues
Packaging workflow	Automated builds and releases with GitHub Actions CI workflows	⭐⭐⭐⭐⭐	Automated CI/CD ensures reliable packaging	Maintain and audit CI pipelines

Figure 5: vLLM Maintenance Analysis from the Generated Report, summarizing repository activity trends.

Dependency Management: Huggingface Transformers

LIBVULNWATCH examined the Huggingface Transformers librarýs dependency management practices. Leveraging its ability to interpret diverse information sources, the system evaluated the availability of a Software Bill of Materials (SBOM), analyzed stated policies regarding dependency updates, and assessed the overall approach to managing a complex dependency network. Figure 6 illustrates a segment of this analysis, demonstrating how language-driven assessment can bridge technical details with governance requirements.



Executive Overview ¶

Huggingface Transformers lacks a formal SBOM and explicit transitive dependency management, with no documented automated dependency updates, resulting in a high dependency risk despite a moderate number of direct dependencies.

Key Facts & Observations ¶

Risk Factor	Observed Data	Rating (⭐)	Reason for Rating	Key Control
SBOM Availability	No formal Software Bill of Materials or dependency manifest found in the official repository or documentation. GitHub repo	⭐	Absence severely limits visibility and auditability of all dependencies used.	Publish a complete SBOM in a standard format
Direct Dependencies	Approximately 10-15 Python dependencies listed in example requirements.txt, including torch, numpy, tokenizers. Example requirements	⭐⭐⭐	Dependency count is moderate and typical for ML libraries, but requires diligent update tracking.	Implement automated dependency version monitoring
Transitive Dependency Management	No explicit mention or tooling found for scanning or managing transitive dependencies. Repo overview	⭐	Lack of transitive dependency control exposes risk of unnoticed vulnerabilities or outdated libs.	Integrate automated transitive dependency scan tools
Vulnerable Dependencies	No known CVEs or security advisories linked to dependencies in last 24 months. Security advisories	⭐⭐⭐⭐	No reported vulnerabilities reduce immediate risk but rely on proactive continuous scanning.	Adopt continuous automated vulnerability scanning
Dependency Update Policy	No stated dependency update policy or automated tooling for reconciliation in official docs or repo. Repo README	⭐	No update policy means risks of outdated or insecure dependencies persist undetected.	Establish and document dependency update workflows

Figure 6: Huggingface Transformers Dependencies Analysis from the Generated Report.

5 Discussion and Future Work

Our findings reveal a critical gap: many technically advanced AI libraries exhibit significant shortcomings in enterprise readiness, particularly in supply chain security and regulatory preparedness (Section 4.2). This underscores a pressing need for more nuanced assessment methodologies. The agent-based approach we introduced (Section 3.2), rooted in language understanding, proved effective in identifying complex vulnerabilities—such as RCEs, supply chain flaws, and governance gaps—that elude conventional checks. The substantial Novelty Yield achieved (Table 1, Section 4.1) quantifies this unique contribution, demonstrating how NLP can uncover critical risks requiring deep contextual interpretation, a finding further supported by the patterns detailed in Section 4.2.

Benchmarking our system (Section 4) against established tools like the OpenSSF Scorecard provides a crucial perspective. While the observed Baseline Alignment (Section 4.1, Table 1) confirms our method's capacity to recognize standard risk indicators, the consistent generation of novel insights highlights the added value of recontextualizing NLP for specialized domains. The variations in alignment and novelty across library categories (Table 2, Section 3.3) suggest that a library's functional niche and maturity, rather than mere complexity, influence its risk profile when assessed through this deeper, language-aware lens.

This work offers a clear demonstration of how advanced language understanding capabilities can transform risk assessment methodologies, moving beyond traditional rule-based paradigms (Section 3). The system's proficiency in interpreting diverse documentation, synthesizing disparate information, and reasoning about nuanced implications (Figure 1) facilitates a depth of analysis previously unattainable with conventional tools. Crucially, this approach enables the identification of emergent, cross-cutting patterns, such as systemic deficiencies in regulatory alignment (Section 4.2), thereby offering insights into broader ecosystemic challenges that demand interdisciplinary attention.

Looking ahead, our research points towards several avenues for intensifying NLP's impact in this and related domains. Enhancing the semantic interpretation of code and API interactions, grounded in our current risk framework (Section 3.1), promises more precise intra-implementation vulnerability detection. The successful application of this NLP-driven framework (Section 3) to software assessment strongly motivates its adaptation to other complex ecosystems, such as healthcare informatics or financial technologies, where similar governance and risk assessment challenges persist. Further exploration of few-shot adaptation could democratize such deep assessment capabilities. Ultimately, integrating structured verification techniques with the contextual reasoning inherent in language models could address current limitations while amplifying the discovery of impactful, novel risks, as evidenced by our Novelty Yield results (Table 1, Section 4.1).

Collectively, these contributions signal a paradigm shift: viewing the evaluation of complex systems not merely as a static analysis task, but as a dynamic knowledge synthesis challenge. This perspective directly leverages recent breakthroughs in language understanding and structured reasoning. By effectively bridging NLP with the distinct domain of software governance, LIBVULNWATCH (Section 3, Section 4) provides not only actionable insights for AI library evaluation but also a robust, transferable methodology for tackling multifaceted governance and risk assessment problems across diverse disciplinary boundaries.

6 Limitations

Despite the capabilities of LibVulnWatch, several limitations warrant discussion, offering avenues for future research and refinement.

Refined Agent Capabilities and Scope While LibVulnWatch demonstrates broad alignment with the OpenSSF Scorecard (as discussed in Section 4.1), its agentic reasoning did not consistently capture all specific checklist items, such as the presence of binary artifacts or formal contributor agreements. This suggests that for comprehensive coverage of all standard security hygiene factors, future iterations could benefit from incorporating more specialized, non-agentic tools or targeted heuristics for these highly structured data points, complementing the agentś deep analysis of more nuanced risks.

Dynamic Nature of Open-Source and Information Availability The accuracy and completeness of LibVulnWatch assessments are intrinsically tied to the availability and quality of public information concerning the target libraries. As open-source projects evolve rapidly, any assessment inherently represents a snapshot in time (e.g., data for this paper reflects May 2025, a point also noted in Section 5). While continuous monitoring via the planned public leaderboard (Section 3.2) aims to mitigate the staleness of information, the depth of analysis will always be constrained by what projects choose to disclose publicly and the recency of indexed information by search APIs.

LLM Dependence and Evaluation Robustness LibVulnWatch leverages the capabilities of LLMs (specifically `gpt-4.1-mini`) for complex information extraction and synthesis. Consequently, the quality and consistency of assessments can be influenced by the LLMś inherent knowledge envelope, reasoning limitations, potential training data biases, and sensitivity to prompt engineering, as acknowledged in Section 5. Although our framework emphasizes evidence-backed findings and structured reporting to mitigate subjectivity and ensure verifiability (Section 3.2), future work could explore ensembles of diverse LLMs, more rigorous calibration of prompt variance, or techniques for explicitly surfacing LLM uncertainty in assessments.

Scalability and Resource Implications for Deep, Continuous Analysis Performing deep, source-grounded analysis for a large number of libraries on a continuous basis presents computational resource considerations. While individual library assessments with `gpt-4.1-mini` are relatively cost-effective (approx. $0.10 per library, as detailed in Section 3.2), scaling this to thousands of libraries with high frequency would necessitate significant infrastructure. Future optimizations might involve adaptive assessment depths based on library criticality or observed change frequency, or the development of more efficient caching mechanisms for retrieved evidence.

Ecosystem-Level Constraints on Assessment Depth A significant constraint, external to LibVulnWatch itself, is the current state of documentation within the open-source AI ecosystem. The pervasive lack of comprehensive and standardized documentation regarding regulatory compliance (e.g., GDPR, AI Act alignment), detailed privacy practices, and robust model/data explainability inherently limits the depth and certainty of assessments in these critical governance domains. While our system is designed to identify such gaps (a pattern noted in Section 4.2)—which itself is a valuable finding—it cannot create information that does not exist. This limitation underscores a broader need for community-driven standards and improved transparency from library developers to enable more thorough governance evaluations.

7 Ethical Considerations

The development and deployment of LibVulnWatch raise several ethical considerations that we have aimed to address throughout its design and proposed usage.

Responsible Disclosure and Vulnerability Reporting As stated in our methodology (Section 3.2), LibVulnWatch is designed to identify potential vulnerabilities in open-source AI libraries. We are committed to responsible disclosure practices. For any new, previously non-public vulnerabilities, particularly critical ones such as the RCEs mentioned in our results (Section 4.1), our protocol involves adhering to the ACL Co-ordinated Disclosure Policy. This includes contacting the developers of the affected library privately, providing them with the necessary details, and allowing a minimum 30-day period for them to address the issue before any public disclosure of the specific, novel vulnerability details. All such communications and their timelines would be documented herein or in a publicly available appendix upon final publication if such instances arise during ongoing or future assessments.

Potential for Misuse While LIBVULNWATCH aims to improve the security and governance of the AI ecosystem by highlighting risks, any tool that identifies vulnerabilities could potentially be misused by malicious actors. To mitigate this, our public leaderboard (as referenced in Section 3.2) focuses on aggregated, governance-aligned scores and known risk patterns rather than detailing zero-day exploits. The primary goal is to incentivize proactive security improvements and inform developers and users, with responsible disclosure handling specific sensitive findings. Furthermore, the types of vulnerabilities it highlights (e.g., missing SBOMs, licensing issues, gaps in regulatory documentation) are often systemic issues that benefit from public awareness to drive broader improvements.

LLM Capabilities, Biases, and Reproducibility The assessment quality of LIBVULNWATCH is inherently linked to the capabilities and potential biases of the underlying Large Language Model (LLM), `gpt-4.1-mini`, as noted in our limitations (Section 5). While we employ engineered prompts and a structured, evidence-based framework (Sections 3.1 and 3.2) to guide the LLM and ensure verifiability (e.g., quantification mandate, evidence requirement), the interpretation and synthesis performed by the LLM may still be subject to its training data biases or inherent limitations. We strive for transparency by detailing our methodology, including the use of specific LLM agents and prompts (though full prompt details are beyond the scope of this paper, the principles are outlined). The generated reports, with direct citations to evidence, are designed to be reproducible and allow for independent verification of findings.

Data Privacy LIBVULNWATCH is designed to assess publicly available open-source AI libraries. The data sources it utilizes, as described in Section 3.2, include public code repositories, official documentation, security databases, and information retrieved via public web search APIs. The system does not require access to private codebases or non-public user data, minimizing direct data privacy risks related to proprietary information.

Impact of Public Ranking and Scoring Publishing a leaderboard with risk scores for AI libraries can have a significant societal impact. Our intention is to foster transparency, accountability, and drive improvements in the security and governance of the AI software supply chain. However, we recognize that scores could be misinterpreted or place undue pressure on developers of libraries that score lower. To mitigate this, LIBVULNWATCH emphasizes a multi-dimensional assessment across five domains (Section 3.1), detailed justifications for scores, and evidence-backed findings, rather than a single opaque metric. The OpenSSF Scorecard benchmarking (Section 3.5) also provides a recognized baseline for comparison. We believe the benefits of increased transparency and informed decision-making for users and developers outweigh the potential downsides, especially given the critical nature of these libraries in AI systems.

Fairness and Objectivity We have designed the assessment framework to be as objective as possible by mandating structured reporting, quantification of metrics, and direct evidence for all claims (Section 3.2). The risk rating criteria (Section 3.4) are predefined to ensure consistency across evaluations. While the LLM introduces a layer of interpretation, the requirement for verifiable evidence aims to ground the assessments in factual data.

We believe that by adhering to these principles, LIBVULNWATCH can serve as a valuable and ethical tool for enhancing the trustworthiness of the open-source AI ecosystem.

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, and 3 others. 2016. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 265–283.

Vasileios Alevizos, George A. Papakostas, Akebu Simasiku, Dimitra Malliarou, Antonis Messinis, Sabrina Edralin, Clark Xu, and Zongliang Yue. 2024. Integrating artificial open generative artificial intelligence into software supply chain security. In *2024 5th International Conference on Data Analytics for Business and Industry (ICDABI)*, page 200–206. IEEE.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake

VanderPlas, Skye Wanderman-Milne, and 1 others. 2025. Jax: composable transformations of python+ numpy programs. https://github.com/google/jax. Accessed: 2025-05-12.

Browserbase. 2025. Stagehand: The production-ready framework for ai browser automations. https://github.com/browserbase/stagehand. Accessed: 2025-05-12.

Junjie Chen, Yihua Liang, Qingchao Shen, Jiajun Jiang, and Shuochuan Li. 2023. Toward Understanding Deep Learning Framework Bugs. *ACM Transactions on Software Engineering and Methodology*, 32(6):135:1–135:31.

Composio. 2025. Composio: Production-ready toolset for ai agents. https://github.com/ComposioHQ/composio. Accessed: 2025-05-12.

CrewAI. 2025. CrewAI: Fast and flexible multi-agent automation framework. https://github.com/crewAIInc/crewAI. Accessed: 2025-05-12.

Google. 2025. Agent Development Kit: An open-source, code-first python toolkit for building, evaluating, and deploying sophisticated ai agents with flexibility and control. https://github.com/google/adk-python. Accessed: 2025-05-12.

Hugging Face. 2025. Text Generation Inference: Large language model text generation inference. https://github.com/huggingface/text-generation-inference. Accessed: 2025-05-12.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. *arXiv preprint arXiv:2309.06180*.

Piergiorgio Ladisa, Henrik Plate, Matias Martinez, and Olivier Barais. 2023. SoK: Taxonomy of Attacks on Open-Source Software Supply Chains. In *Proceedings of the 2023 IEEE Symposium on Security and Privacy (SP)*, pages 1509–1526.

LangChain AI. 2025a. LangChain: Large language model application framework. https://github.com/langchain-ai/langchain. Accessed: 2025-05-12.

LangChain AI. 2025b. LangGraph: An open-source ai agent orchestration framework. https://docs.langchain.com/docs/langgraph. Accessed: 2025-05-12.

Jerry Liu. 2022. LlamaIndex. https://github.com/jerryjliu/llama_index.

Magnus Müller and Gregor Žunič. 2024. Browser use: Enable ai to control your browser. https://github.com/browser-use/browser-use.

NVIDIA. 2025. TensorRT: High-performance deep learning inference on nvidia gpus. https://github.com/NVIDIA/TensorRT. Accessed: 2025-05-12.

Marc Ohm, Henrik Plate, Arnold Sykosch, and Michael Meier. 2020. Backstabber's Knife Collection: A Review of Open Source Software Supply Chain Attacks. In *Proceedings of the 17th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, pages 23–43.

ONNX. 2025. ONNX: Open neural network exchange. https://github.com/onnx/onnx. Accessed: 2025-05-12.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, and 2 others. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Pydantic. 2025. Pydantic AI: shim to use pydantic with llms. https://github.com/pydantic/pydantic-ai. Accessed: 2025-05-12.

Anka Reuel, Benjamin Bucknall, Stephen Casper, Timothy Fist, Lisa Soder, Onni Aarne, Lewis Hammond, Lujain Ibrahim, Alan Chan, Peter Wills, Markus Anderljung, Ben Garfinkel, Lennart Heim, Andrew Trask, Gabriel Mukobi, Rylan Schaeffer, Mauricio Baker, Sara Hooker, Irene Solaiman, and 14 others. 2025. Open problems in technical AI governance. *Transactions on Machine Learning Research*. Survey Certification.

Aymeric Roucher, Albert Villanova del Moral, Thomas Wolf, Leandro von Werra, and Erik Kaunismäki. 2025. 'smolagents': a smol library to build great agentic systems. https://github.com/huggingface/smolagents.

Shenao Wang, Yanjie Zhao, Zhao Liu, Quanchen Zou, and Haoyu Wang. 2025. SoK: Understanding Vulnerabilities in the Large Language Model Supply Chain. *arXiv preprint arXiv:2502.12497*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clément Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Téven Le Scao, Sylvain Gugger, and 3 others. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45. Association for Computational Linguistics.

Nusrat Zahan, Parth Kanakiya, Brian Hambleton, Shohanuzzaman Shohan, and Laurie Williams. 2023.

Openssf scorecard: On the path toward ecosystem-wide automated security metrics. *IEEE Security & Privacy*, 21(6):76–88.

[first names omitted for brevity] Zhang and colleagues. 2024. Metagpt: Meta programming sota autonomous multi-agent cooperative llm workflows. *arXiv preprint arXiv:2404.14496*.

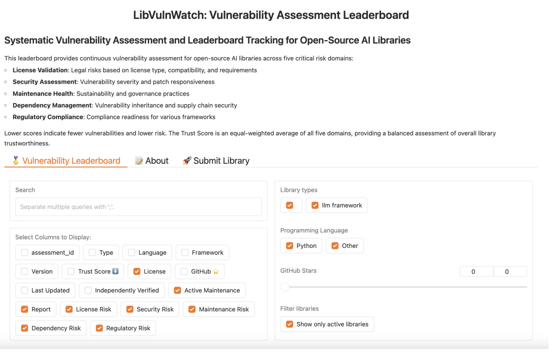
Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. 2023. Sglang: Efficient execution of structured language model programs. *arXiv preprint arXiv:2312.07104*.

Xin Zhou, Sicong Cao, Xiaobing Sun, and David Lo. 2024. Large Language Model for Vulnerability Detection and Repair: Literature Review and the Road Ahead. *ACM Transactions on Software Engineering and Methodology*. To appear.

A Appendix

A.1 Interactive Leaderboard Interface and Implementation

This subsection describes the LibVulnWatch vulnerability assessment leaderboard and presents screenshots of its key functionalities. The leaderboard is implemented as an interactive web application using Gradio (**?**) and is publicly deployed on Hugging Face Spaces. It allows users to search, filter, and view detailed vulnerability assessment reports for a wide range of open-source AI libraries. Users can also find guidelines and submit new libraries for assessment through this interface. The figures below illustrate the main views of the leaderboard, including search and filtering capabilities (Figure 7), library submission guidelines (Figure 8), the tabular display of assessed libraries with links to reports (Figure 9), and the new library submission form (Figure 10).



LibVulnWatch: Vulnerability Assessment Leaderboard

Systematic Vulnerability Assessment and Leaderboard Tracking for Open-Source AI Libraries

This leaderboard provides continuous vulnerability assessment for open-source AI libraries across five critical risk domains:

- **License Validation**: Legal risks based on license type, compatibility, and requirements
- **Security Assessment**: Vulnerability severity and patch responsiveness
- **Maintenance Health**: Sustainability and governance practices
- **Dependency Management**: Vulnerability inheritance and supply chain security
- **Regulatory Compliance**: Compliance readiness for various frameworks

Lower scores indicate fewer vulnerabilities and lower risk. The Trust Score is an equal-weighted average of all five domains, providing a balanced assessment of overall library trustworthiness.

Search: Separate multiple queries with ';'.

Select Columns to Display: ☐ assessment_id ☐ Type ☐ Language ☐ Framework ☐ Version ☐ Trust Score ✓ License ☐ GitHub ☐ Last Updated ☐ Independently Verified ✓ Active Maintenance ✓ Report ✓ License Risk ✓ Security Risk ✓ Maintenance Risk ✓ Dependency Risk ✓ Regulatory Risk

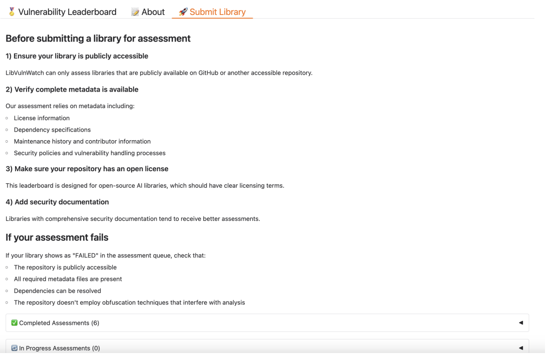
Library types: ✓ ✓ llm framework

Programming Language: ✓ Python ✓ Other

GitHub Stars: 0 0

Filter libraries: ✓ Show only active libraries

Figure 7: The main LibVulnWatch leaderboard view, showing search and filtering options for assessed AI libraries across five risk domains.



Before submitting a library for assessment

1) Ensure your library is publicly accessible

LibVulnWatch can only assess libraries that are publicly available on GitHub or another accessible repository.

2) Verify complete metadata is available

Our assessment relies on metadata including:

- License information
- Dependency specifications
- Maintenance history and contributor information
- Security policies and vulnerability handling processes

3) Make sure your repository has an open license

This leaderboard is designed for open-source AI libraries, which should have clear licensing terms.

4) Add security documentation

Libraries with comprehensive security documentation tend to receive better assessments.

If your assessment fails

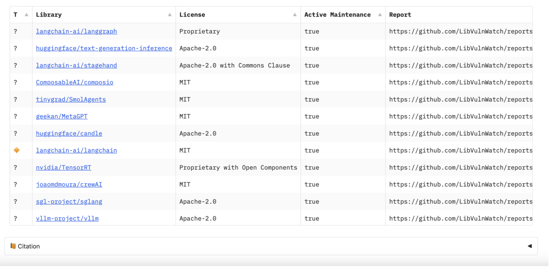
If your library shows as "FAILED" in the assessment queue, check that:

- The repository is publicly accessible
- All required metadata files are present
- Dependencies can be resolved
- The repository doesn't employ obfuscation techniques that interfere with analysis

✅ Completed Assessments (6)

In Progress Assessments (0)

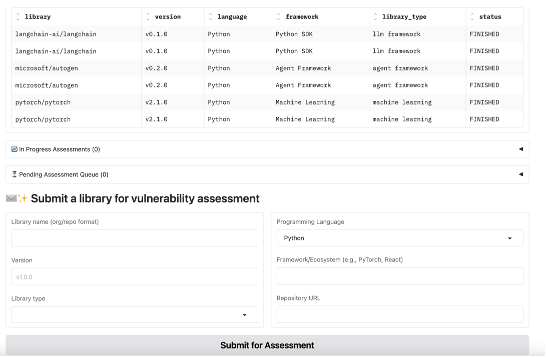
Figure 8: Guidelines and prerequisites for submitting a new library for assessment on the LibVulnWatch platform.



T	Library	License	Active Maintenance	Report
?	langchain-ai/langgraph	Proprietary	true	https://github.com/LibVulnWatch/reports
?	huggingface/text-generation-inference	Apache-2.0	true	https://github.com/LibVulnWatch/reports
?	langchain-ai/stagehand	Apache-2.0 with Commons Clause	true	https://github.com/LibVulnWatch/reports
?	ComposableAI/composio	MIT	true	https://github.com/LibVulnWatch/reports
?	tinygrad/SmolAgents	MIT	true	https://github.com/LibVulnWatch/reports
?	geekan/MetaGPT	MIT	true	https://github.com/LibVulnWatch/reports
?	huggingface/candle	Apache-2.0	true	https://github.com/LibVulnWatch/reports
🔶	langchain-ai/langchain	MIT	true	https://github.com/LibVulnWatch/reports
?	nvidia/TensorRT	Proprietary with Open Components	true	https://github.com/LibVulnWatch/reports
?	joaomdmoura/crewAI	MIT	true	https://github.com/LibVulnWatch/reports
?	sgl-project/sglang	Apache-2.0	true	https://github.com/LibVulnWatch/reports
?	vllm-project/vllm	Apache-2.0	true	https://github.com/LibVulnWatch/reports

Citation

Figure 9: Tabular display of assessed libraries, including details such as license, maintenance status, and direct links to individual vulnerability reports.



library	version	language	framework	library_type	status
langchain-ai/langchain	v0.1.0	Python	Python SDK	llm framework	FINISHED
langchain-ai/langchain	v0.1.0	Python	Python SDK	llm framework	FINISHED
microsoft/autogen	v0.2.0	Python	Agent Framework	agent framework	FINISHED
microsoft/autogen	v0.2.0	Python	Agent Framework	agent framework	FINISHED
pytorch/pytorch	v2.1.0	Python	Machine Learning	machine learning	FINISHED
pytorch/pytorch	v2.1.0	Python	Machine Learning	machine learning	FINISHED

In Progress Assessments (0)

⏳ Pending Assessment Queue (0)

✉️✨ Submit a library for vulnerability assessment

Library name (org/repo format):

Programming Language: Python

Version: v1.0.0

Framework/Ecosystem (e.g., PyTorch, React):

Library type:

Repository URL:

Figure 10: The LibVulnWatch interface for submitting a new open-source AI library for vulnerability assessment and inclusion in the leaderboard.

A.2 Agent Prompts

This section details the core instruction sets (prompts) provided to the various specialized agents within the LibVulnWatch system. These prompts guide the agents in their respective tasks of planning, querying, writing, and evaluating risk assessment information.

A.2.1 Initial Query Formulation for Report Planning

Listing 1: Initial Query Formulation for Report Planning. This agent generates initial search queries to gather context for planning the overall report structure.

```
You are performing comprehensive open source risk management assessment following industry best practices.

<Library input>
{topic}
```

```
</Library input>

<Report organization>
{report_organization}
</Report organization>

<Task>
Your goal is to generate {number_of_queries} web search queries that will gather comprehensive information for assessing the
      ↪ risks of this open source library according to enterprise security standards.

IMPORTANT: The library input may be either a library name (e.g., "TensorFlow", "React") or a repository URL (e.g., "https://
      ↪ github.com/tensorflow/tensorflow"). Adjust your queries accordingly.

<High-Quality Source Guidelines>
Prioritize authoritative and reliable sources by targeting queries toward:
- Official documentation (GitHub repos, project websites, official guides)
- Security databases (NVD, CVE records, security bulletins)
- Industry research (research papers, security firm reports)
- Regulatory bodies (NIST, ISO, CIS documentation)
- Technical forums with verification (StackOverflow with high votes)

Avoid low-quality sources like:
- General blogs without technical expertise
- Marketing materials
- Outdated repositories (>2 years without updates)
- Non-technical news articles

Use site: operators to target specific high-quality domains (e.g., site:github.com, site:nvd.nist.gov).
</High-Quality Source Guidelines>

The queries should comprehensively cover these key risk areas:

1. LICENSE VALIDATION:
   - License type (MIT, Apache 2.0, GPL, etc.)
   - Commercial use compatibility
   - License history and changes
   - Attribution requirements
   - Patent grant provisions

2. SECURITY ASSESSMENT:
   - Common Vulnerabilities and Exposures (CVEs)
   - Security patch frequency and responsiveness
   - Vulnerability scanning reports
   - OWASP dependency risks
   - Historical security incidents

3. MAINTENANCE HEALTH:
   - Release frequency and consistency
   - Number of active contributors (current vs. historical)
   - Issue response time metrics
   - Pull request acceptance rate
   - Governance model (individual, community, foundation)

4. DEPENDENCY MANAGEMENT:
   - Software Bill of Materials (SBOM) availability
   - Transitive dependency tracking
   - Dependency update policies
   - Supply chain security measures
   - CI/CD integration for dependency scanning

5. REGULATORY COMPLIANCE:
   - Explainability requirements (especially for AI libraries)
   - Industry-specific regulatory frameworks applicable
   - Data privacy implications
   - Export control restrictions
   - Audit readiness documentation

Make the queries specific, technical, and designed to retrieve quantifiable metrics from authoritative sources wherever
      ↪ possible. Use site: operators to target specific high-quality domains when appropriate.
</Task>

<Format>
Call the Queries tool
</Format>
```

A.2.2 Report Structure Planning Instructions

Listing 2: Report Structure Planning Instructions. This agent generates the structured plan for the report, outlining the sections to be created.

```
I want a comprehensive open source risk assessment report that meets enterprise governance standards and regulatory compliance
      ↪ requirements.

<Library input>
The library to assess is:
{topic}
</Library input>

<Report organization>
The report should follow this organization:
{report_organization}
</Report organization>

<Context>
Here is context to use to plan the sections of the risk assessment report:
{context}
```

```
</Context>

<Task>
Generate a detailed structure for an enterprise-grade open source risk assessment report on the provided library.

IMPORTANT: The library input may be either a library name (e.g., "TensorFlow", "React") or a repository URL (e.g., "https://
      ↪ github.com/tensorflow/tensorflow"). Identify the specific library from the input.

Your plan should include specialized sections that cover ALL of the following risk domains based on industry best practices:

1. KEY RISK DOMAINS (each requiring full assessment as separate sections):
   - LICENSE ANALYSIS - Terms, compatibility, patent provisions
   - SECURITY ASSESSMENT - CVE history, patch frequency, testing
   - MAINTENANCE INDICATORS - Release cadence, contributors, support
   - DEPENDENCY MANAGEMENT - SBOM, transitive risks, updates
   - REGULATORY CONSIDERATIONS - Compliance frameworks, explainability

NOTE:
- The EXECUTIVE SUMMARY will be generated automatically after all sections are written, so DO NOT include it in your section
      ↪ list.
- RISK MITIGATION RECOMMENDATIONS will be included in the Executive Summary, so DO NOT create it as a separate section.

Each section should have the fields:
- Name - Name for this section of the report.
- Description - Brief overview of what this section assesses.
- Research - Whether to perform web research for this section. IMPORTANT: All main sections MUST have Research=True.
- Content - The content of the section, which you will leave blank for now.

Ensure the structure focuses on quantifiable metrics and evidence-based assessment rather than general descriptions. The
      ↪ report should be highly actionable, non-redundant, and concise.
</Task>

<Feedback>
Here is feedback on the report structure from review (if any):
{feedback}
</Feedback>

<Format>
Call the Sections tool
</Format>
```

A.2.3 Domain-Specific Query Formulation Instructions

Listing 3: Domain-Specific Query Formulation Instructions. This agent generates specific search queries for a given section of the report.

```
You are an enterprise security analyst specializing in open source risk governance and compliance.

<Library input>
{topic}
</Library input>

<Section topic>
{section_topic}
</Section topic>

<Task>
Generate {number_of_queries} highly specific search queries to gather comprehensive data for assessing the open source risks
      ↪ of this library, focusing specifically on {section_topic}.

IMPORTANT: The library input may be either a library name (e.g., "TensorFlow", "React") or a repository URL (e.g., "https://
      ↪ github.com/tensorflow/tensorflow"). Always include the library name explicitly in your queries.

<Advanced GitHub Data Extraction>
Since we do not have API access, use these specialized search patterns to extract public repository metrics:

1. For contributor metrics:
   - "[Library] github.com/[org]/[repo]/graphs/contributors" (finds contributor pages)
   - "[Library] [org]/[repo] number of contributors [year]" (finds specific counts)
   - "[Library] [org]/[repo] top contributors" (finds lead maintainer information)

2. For issue statistics:
   - "[Library] github.com/[org]/[repo]/issues?q=is:issue+is:open+sort:updated-desc" (finds open issues)
   - "[Library] github.com/[org]/[repo]/issues?q=is:issue+is:closed" (finds closed issues)
   - "[Library] average issue resolution time" (finds resolution metrics)

3. For release history:
   - "[Library] github.com/[org]/[repo]/releases" (finds release pages)
   - "[Library] latest release version number date" (finds current version)
   - "[Library] release frequency [year]" (finds release cadence)

4. For security practices:
   - "[Library] github.com/[org]/[repo]/security/advisories" (finds security advisories)
   - "[Library] github.com/[org]/[repo]/blob/master/SECURITY.md" (finds security policies)
   - "[Library] CVE [year] vulnerability" (finds published vulnerabilities)

5. For dependency information:
   - "[Library] github.com/[org]/[repo]/blob/master/requirements.txt" (finds Python dependencies)
   - "[Library] github.com/[org]/[repo]/blob/master/package.json" (finds JS dependencies)
   - "[Library] github.com/[org]/[repo]/network/dependencies" (finds dependency graphs)

6. For license details:
   - "[Library] github.com/[org]/[repo]/blob/master/LICENSE" (finds license file)
   - "[Library] github.com/[org]/[repo]/blob/master/LICENSE.md" (alternative license file)
   - "[Library] license type changed history" (finds license changes)
</Advanced GitHub Data Extraction>
```

```
<High-Quality Source Guidelines>
Prioritize authoritative and reliable sources by targeting queries toward:
- Official documentation (GitHub repos, project websites, official guides)
- Security databases (NVD, CVE records, security bulletins)
- Industry research (research papers, security firm reports)
- Regulatory bodies (NIST, ISO, CIS documentation)
- Technical forums with verification (StackOverflow with high votes)

Avoid low-quality sources like:
- General blogs without technical expertise
- Marketing materials
- Outdated repositories (>2 years without updates)
- Non-technical news articles

Your queries should specifically target these high-quality sources when possible.
</High-Quality Source Guidelines>

Based on the section topic, craft specialized queries from these categories:

LICENSE ANALYSIS:
- "[Library] license type commercial use compatibility site:github.com OR site:opensource.org"
- "[Library] license change history site:github.com/[org]/[repo]"
- "[Library] patent grant provisions license text"
- "[Library] license compliance requirements site:spdx.org OR site:github.com"
- "[Library] GPL/LGPL/AGPL compatibility analysis"
- "[Library] attribution requirements license text site:opensource.org"

SECURITY ASSESSMENT:
- "[Library] CVE history last 3 years site:nvd.nist.gov OR site:cve.mitre.org"
- "[Library] security vulnerabilities mitigated site:github.com/[org]/[repo]/security"
- "[Library] CVSS score recent vulnerabilities site:nvd.nist.gov"
- "[Library] security disclosure policy site:github.com/[org]/[repo]"
- "[Library] security patch response time average"
- "[Library] supply chain security scorecard"

MAINTENANCE HEALTH:
- "[Library] release frequency metrics site:github.com/[org]/[repo]/releases"
- "[Library] active contributors count trend site:github.com/[org]/[repo]/graphs/contributors"
- "[Library] issue resolution time average site:github.com/[org]/[repo]/issues"
- "[Library] pull request acceptance rate site:github.com/[org]/[repo]/pulls"
- "[Library] documentation quality assessment site:github.com/[org]/[repo]/wiki"
- "[Library] governance foundation or company site:github.com OR site:[official-site]"

DEPENDENCY MANAGEMENT:
- "[Library] SBOM availability CycloneDX or SPDX site:github.com/[org]/[repo]"
- "[Library] transitive dependencies count analysis"
- "[Library] dependency vulnerability scanning site:github.com/[org]/[repo]/security/dependabot"
- "[Library] dependency freshness policy site:github.com/[org]/[repo]"
- "[Library] CI/CD dependency scanning integration site:github.com/[org]/[repo]/.github/workflows"
- "[Library] vulnerable dependencies percentage report"

REGULATORY COMPLIANCE:
- "[Library] regulatory compliance frameworks site:[official-site] OR site:github.com/[org]/[repo]"
- "[Library] explainability for AI models documentation site:github.com/[org]/[repo]"
- "[Library] data privacy implications GDPR CCPA CPRA site:github.com/[org]/[repo]"
- "[Library] export control classification ECCN"
- "[Library] NIST SSDF compatibility assessment"
- "[Library] audit readiness documentation site:github.com/[org]/[repo]"

<Data Extraction Instructions>
For each query, focus on extracting specific numerical metrics:
- Always search for EXACT numbers when available: "X contributors" not "many contributors"
- Look for timestamps and dates: "Last release: March 15, 2024" not "recent release"
- Search for explicit vulnerability counts: "3 CVEs in 2023" not "some vulnerabilities"
- Seek percentages and ratios: "85% test coverage" not "good test coverage"

For repositories, use google dorks to find specific file content:
- Use 'inurl:github.com/[org]/[repo] filetype:md SECURITY' to find security documentation
- Use 'inurl:github.com/[org]/[repo] "license"' to find license information
- Use 'inurl:github.com/[org]/[repo] "requirements.txt" OR "package.json"' to find dependencies
</Data Extraction Instructions>

Generate queries that return quantitative metrics, statistical data, and factual evidence from authoritative sources. Use site
    ↪ : operators when appropriate to target specific high-quality domains.
</Task>

<Format>
Call the Queries tool
</Format>
```

A.2.4 Draft Findings Generation Instructions for Report Sections

Listing 4: Draft Findings Generation Instructions for Report Sections. This agent synthesizes information from web search results to write a specific section of the report, adhering to strict formatting and citation requirements.

```
Write a highly focused assessment of open source risk.

<Task>
1. Analyze the library based on the section name and topic.
2. Focus ONLY on observed facts with proper citations.
3. Use the most concise format possible while addressing all key risk factors.
4. IMPORTANT: For each risk factor, assign at least one HIGH risk rating if evidence justifies it. Never rate all factors as
    ↪ only Low/Medium.
</Task>
```

```
<Streamlined Structure>
## [Section Name]

### Executive Overview
[1 sentence summary of risk level and justification]

###  [ALERT] Emergency Issues
<span style="color:red">
**Critical Issue**: [Most serious high-risk finding with citation link for critical information only](url)
</span>

### Key Facts & Observations
| Risk Factor | Observed Data | Rating (⋆) | Reason for Rating | Key Control |
|-------------|---------------|------------|-------------------|-------------|
| [Factor 1]  | [Specific metric/fact with citation link](url) | ⋆⋆⋆⋆⋆ | [Why this is low risk] | [Solution]  |
| [Factor 2]  | [Specific metric/fact with citation link](url) | ⋆⋆⋆ | [Why this is medium risk] | [Solution]  |
| [Factor 3]  | [Specific metric/fact with citation link](url) | ⋆ | [Why this is high risk] | [Solution]  |
</Streamlined Structure>

<Coverage Requirements>
Based on your section topic, address ALL relevant key concepts:

LICENSE ANALYSIS:
- License type (MIT, Apache, GPL, etc.) with version
- Commercial use & distribution rights
- Patent grant provisions
- Attribution requirements
- Conformance with open source compliance standards

SECURITY ASSESSMENT:
- CVEs in past 24 months (count, severity)
- Security disclosure policy existence
- Response time for security issues
- Security testing evidence (CI/CD test coverage)
- Released binaries or signed artifacts and release notes

MAINTENANCE INDICATORS:
- Latest release date
- Release frequency (releases per month/year)
- Active contributor count (diversity and organizational backing)
- Issue resolution metrics (recent commit activity and issue engagement details)
- Packaging workflow for publishing

DEPENDENCY MANAGEMENT:
- SBOM availability (Yes/No, format)
- Direct dependency count
- Transitive dependency management
- Vulnerable dependency count
- Existence of dependency update tools/policies

REGULATORY CONSIDERATIONS:
- Compliance frameworks supported
- Explainability features for AI/ML
- Data privacy provisions
- Audit documentation availability
- AI governance and key AI regulations

CRITICAL: Ensure EVERY metric has a specific value, NOT general statements.
</Coverage Requirements>

<Writing Guidelines>
- Extract the library name from the input (may be name or repository URL)
- Use ONLY observed facts and metrics with citations:
  - "Last release: March 15, 2024" not "recent release"
  - "243 active contributors" not "many contributors"
  - "No CVEs in past 24 months" not "good security record"

- STRICT CITATION REQUIREMENTS:
  - ONLY make claims that are EXPLICITLY stated in the source material
  - DO NOT infer, assume, or extrapolate beyond what's directly stated in the sources
  - If source material does not explicitly mention a metric, acknowledge this as "No data available on X" and rate accordingly
  - Maintain clear traceability between each claim and the exact source
  - For missing but important information, indicate "Not specified in documentation" rather than guessing

- CITATION FORMAT AND FREQUENCY:
  - ONLY use inline markdown hyperlinks for direct URLs: `[fact](source-url)`
  - IMPORTANT: EVERY row in the Key Facts & Observations table MUST have at least one citation link
  - For multiple facts in a single row, include a citation link for the most significant facts
  - Cite official documentation, repository pages, security databases, and other authoritative sources whenever possible
  - Include citations for:
     * ALL license details, terms, and provisions
     * ALL security vulnerabilities and patches
     * ALL maintenance metrics and observations
     * ALL dependency numbers and management approaches
     * ALL regulatory tools and frameworks
  - If information was found on a source without a public URL (e.g., local analysis), clearly state this but still provide the
      ↪  observation
  - ALWAYS link to primary sources rather than secondary sources when possible (e.g., GitHub repo over blog post)
  - Include SPECIFIC links to exact locations (e.g., link to specific GitHub issue page, not just GitHub home)
  - Example: Instead of just "[TensorFlow GitHub](https://github.com/tensorflow/tensorflow)", use "[TensorFlow has 58,000+
      ↪ stars](https://github.com/tensorflow/tensorflow)"

- Risk Rating Format:
  - ALWAYS use star ratings only: ⋆⋆⋆⋆⋆, ⋆⋆⋆, ⋆
  - Low risk: ⋆⋆⋆⋆⋆
  - Medium risk: ⋆⋆⋆
  - High risk: ⋆

- Risk Rating Reasons:
  - Provide a concise 1-sentence explanation for EACH risk rating
  - Explicitly reference the specific criteria that determined the rating
  - For HIGH risks, clearly state what threshold was exceeded or requirement not met
  - For LOW risks, explain what positive factors led to this favorable rating
  - When rating based on ABSENCE of information, clearly state this as the reason

- Risk Level Distribution:
  - IMPORTANT: The most realistic assessment MUST include at least ONE HIGH risk item
  - Do not artificially inflate risk; base it on evidence
  - If no clear high risk is found, identify the MOST concerning factor and explain why it poses high risk
  - Absence of critical information itself can justify a high risk rating

- Emergency Issues:
  - Include ONLY if HIGH risk with immediate impact potential is EXPLICITLY supported by sources
  - Otherwise omit this section entirely
  - Always include a specific, actionable solution
  - Never speculate about emergency scenarios not directly evidenced in sources

- Format using markdown with HTML color tags for emergency section
- Limit to maximum 350 words total
- Omit any redundant explanations or theoretical discussions
</Writing Guidelines>

<Risk Rating Criteria>
For each risk factor, apply these specific criteria:

LOW RISK (★★★★★):
- License: Permissive (MIT, Apache 2.0, BSD) with clear terms and compatibility
- Security: No CVEs in past 24 months, robust security policy, rapid fixes (<7 days)
- Maintenance: >10 active contributors, monthly+ releases, <24hr issue response
- Dependencies: SBOM available, <20 direct dependencies, automatic updates
- Regulatory: Clear compliance documentation, complete audit trail

MEDIUM RISK (★★★):
- License: Moderate restrictions or unclear patent provisions
- Security: 1-3 minor CVEs (12mo), basic security policy, moderate response (7-30 days)
- Maintenance: 3-10 contributors, quarterly releases, 1-7 day issue response
- Dependencies: Partial SBOM, 20-50 direct dependencies, some transitive visibility
- Regulatory: Incomplete compliance docs, partial audit readiness

HIGH RISK (★):
- License: Restrictive (GPL/AGPL), incompatible terms, legal concerns
- Security: Critical/multiple CVEs, missing security policy, slow response (>30 days)
- Maintenance: <3 contributors, infrequent releases (>6mo), poor issue response
- Dependencies: No SBOM, >50 direct dependencies, vulnerable transitive deps
- Regulatory: Missing compliance docs, fails essential regulations
- IMPORTANT: Absence of critical information on any key risk factor should be rated as HIGH RISK
</Risk Rating Criteria>

<Final Check>
1. Verify EVERY row in your Key Facts & Observations table has at least one citation link
2. Confirm all relevant risk metrics for your section are addressed
3. Ensure star ratings are used correctly
4. Confirm at least ONE high-risk (★) item is identified
5. Ensure EVERY risk rating has a clear reason explaining the rating
6. Ensure total length is under 350 words
7. Remove any theoretical or duplicated content
8. Verify each observation has a specific control/solution
9. Double-check that NO claims are made without explicit source evidence
10. Verify that absence of information is properly acknowledged and rated accordingly
11. Do NOT include a separate Sources section - use inline links for critical facts only
12. Do NOT use numbered citations [1], [2], etc. - ONLY use inline hyperlinks
13. Ensure there are NO notes/references/sources sections at the end of your report
14. Check that EVERY required risk factor for your section has been addressed with specific metrics
</Final Check>
```

A.2.5 Quality Assessment Instructions for Draft Sections

Listing 5: Quality Assessment Instructions for Draft Sections. This agent evaluates the quality of a written section and generates follow-up queries if information is missing or insufficient.

```
You are a Chief Information Security Officer reviewing an open source risk assessment report section:

<Library input>
{topic}
</Library input>

<section topic>
{section_topic}
</section topic>

<section content>
{section}
</section content>

<task>
Rigorously evaluate whether this section meets enterprise security standards for open source risk assessment. Apply the
    ↪ following STRICT evaluation criteria:

1. QUANTIFICATION: Does the section provide PRECISE metrics (exact dates, counts, percentages, time periods)?
2. EVIDENCE: Is every risk claim supported by cited source evidence?
3. RISK RATING: Is each risk factor explicitly rated (Low/Medium/High) with clear justification?
4. ACTIONABILITY: Are the recommendations specific, technical, and implementable?
5. ENTERPRISE RELEVANCE: Does the assessment address governance, compliance, and security concerns at an enterprise level?

For a PASS grade, the section must meet ALL criteria above with no significant gaps.
```

```
If any criteria are not fully met, generate {number_of_follow_up_queries} targeted follow-up search queries to obtain the
      ↪ missing information. These queries should be highly specific and designed to retrieve quantitative data.
</task>

<format>
Call the Feedback tool and output with the following schema:

grade: Literal["pass","fail"] = Field(
    description="Evaluation result indicating whether the risk assessment meets enterprise standards ('pass') or needs
         ↪ revision ('fail')."
)
follow_up_queries: List[SearchQuery] = Field(
    description="List of follow-up search queries to gather missing quantitative data.",
)
</format>
```

A.2.6 Executive Summary Generation Instructions

Listing 6: Executive Summary Generation Instructions. This agent generates the overall executive summary of the report, synthesizing information from all completed sections.

```
You are a Chief Security Officer providing the EXECUTIVE SUMMARY for an open source risk assessment report.

<Library input>
{topic}
</Library input>

<Context>
{context}
</Context>

<Task>
Create a comprehensive EXECUTIVE SUMMARY as the FIRST SECTION of the report that consolidates findings from all risk domains
      ↪ and includes integrated risk mitigation recommendations. The executive summary must give decision makers a complete
      ↪ picture of the risk profile while being concise and actionable.
</Task>

<Executive Summary Format>
## Executive Summary

### Risk Score Dashboard
| Risk Domain | Rating | Key Finding | Reason for Rating | Key Control |
|-------------|--------|-------------|-------------------|-------------|
| License     | ⋆⋆⋆⋆⋆ | [Specific metric with citation link](url) | [Why this is low risk] | [Solution]  |
| Security    | ⋆⋆⋆ | [Specific metric with citation link](url) | [Why this is medium risk] | [Solution]  |
| Maintenance | ⋆⋆⋆⋆ | [Specific metric with citation link](url) | [Why this is low risk] | [Solution]  |
| Dependencies| ⋆ | [Specific metric with citation link](url) | [Why this is high risk] | [Solution]  |
| Regulatory  | ⋆⋆⋆ | [Specific metric with citation link](url) | [Why this is medium risk] | [Solution]  |
| **OVERALL** | ⋆⋆⋆ | [Overall assessment with citation link](url) | [Why this overall rating] | [Priority action] |

###  [ALERT]  EMERGENCY ISSUES
<span style="color:red">
**[Critical Issue]**: [Most serious HIGH risk finding with citation link](url)
* **Immediate Action**: [Specific, implementable solution]
</span>

### Top Controls by Priority
1. **Immediate (0-7 days)**: [Action for HIGH risk items with citation link](url)
2. **Short-term (30 days)**: [Important technical control with citation link](url)
3. **Medium-term (90 days)**: [Important policy/legal control with citation link](url)

### Comprehensive Risk Mitigation Strategy
Based on all section findings, provide a concise but comprehensive summary of risk mitigation actions needed across all
      ↪ domains:

1. **Technical Controls**:
   - [Specific technical implementation or control with citation link](url)
   - [Specific technical implementation or control with citation link](url)

2. **Policy & Governance Controls**:
   - [Specific policy or governance control with citation link](url)
   - [Specific policy or governance control with citation link](url)

3. **Legal & Compliance Controls**:
   - [Specific legal or compliance control with citation link](url)
   - [Specific legal or compliance control with citation link](url)
</Executive Summary Format>

<Guidelines>
- PLACEMENT: The Executive Summary MUST be the FIRST section of the report
- SCOPE: This summary must cover ALL risk domains assessed in the detailed sections

- CITATION FORMAT AND FREQUENCY:
  - ONLY use inline markdown hyperlinks for direct URLs: `[fact](source-url)`
  - IMPORTANT: EVERY row in the Risk Score Dashboard table MUST have at least one citation link
  - EVERY recommended control in all sections MUST include a citation link to source guidance or documentation
  - Link to specific pages and resources, not just general websites
  - Include links to:
    * ALL significant vulnerabilities and findings
    * ALL tools or frameworks mentioned
    * ALL reference documentation for recommended controls
    * ALL key metrics underpinning risk assessments
  - Example: Instead of just "[TensorFlow security page](https://www.tensorflow.org/security)", use "[12 critical CVEs
        ↪ reported in TensorFlow since 2022](https://www.tensorflow.org/security)"
  - Focus on links to primary sources (official documentation, repository data, security databases)
  - ALWAYS verify URLs exist before including them
  - NEVER hallucinate or fabricate links
  - If uncertain about a URL\'s existence, present the fact without a link
  - Do NOT use numbered citations or separate reference lists

- RISK RATINGS: Use star ratings only:
  - ★★★★★ for Low risk
  - ★★★ for Medium risk
  - ★ for High risk

- HIGH RISK: MUST identify at least one HIGH risk area (★)
- JUSTIFICATION: For EACH risk rating, provide a clear 1-sentence reason explaining why it received that rating
- EMERGENCY ISSUES: This section should ONLY appear if truly critical issues exist
- LENGTH: Limit to 600 words maximum for readability
- FOCUS: Present only the highest priority findings from each domain
- ACTIONABILITY: Ensure every finding has a corresponding control/solution
- ORDER: Risk domains should be ordered from highest to lowest risk
- MITIGATION SECTION: Include a dedicated risk mitigation strategy section that consolidates recommendations from all sections
- CONSISTENCY CHECK: Ensure all facts and assessments are consistent across the entire executive summary
</Guidelines>
```

A.2.7 Repository Identification Instructions for Benchmarking

Listing 7: Repository Identification Instructions for Benchmarking. This agent identifies the GitHub repository URL for a given library name or URL.

```
You are a GitHub repository identifier.

<Library input>
{topic}
</Library input>

<Full Report>
{full_report}
</Full Report>

<Task>
Extract the GitHub repository owner and name from the input. The input may be:
1. A direct GitHub URL (e.g., https://github.com/owner/repo)
2. A library name that can be mapped to a GitHub repository (e.g., "TensorFlow", "React")
3. Any other open source project reference

For library names or general references, determine the most official or popular GitHub repository.

Return the repository in the format "owner/repo".
</Task>

<Format>
Call the GitHubRepo tool
</Format}
```

A.2.8 Scorecard Analysis and Report Comparison Instructions

Listing 8: Scorecard Analysis and Report Comparison Instructions. This agent compares the generated report against OpenSSF Scorecard results to identify overlaps and novel findings.

```
You are an open source security analyst specializing in the OpenSSF Scorecard.

<Scorecard Results>
{scorecard_results}
</Scorecard Results>

<Full Report>
{full_report}
</Full Report>

<Task>
Analyze the OpenSSF Scorecard results alongside the full risk assessment report to determine:

1. Model Coverage: Which OpenSSF Scorecard metrics were already covered in the full report
2. Model Seeking: Which issues were discovered by the model but not identified by Scorecard

IMPORTANT:
- EXCLUDE all scorecard checks with "?" scores from your analysis
- The denominator for coverage should be the total number of applicable checks (excluding "?" scores)
- Count each row in the scorecard results table as one check

IMPORTANT METRICS TO TRACK:
1. MODEL_COVERAGE: Number of OpenSSF Scorecard checks that were adequately addressed in the report
2. MODEL_SEEKING: Number of issues the model found that weren't explicitly mentioned in Scorecard

FORMAT IN MARKDOWN:
Instead of using dictionaries for lacks and extras, include this information as bullet points in your coverage_summary using
      ↪ markdown format:

**Coverage Summary:**
- Model Coverage: [Actual covered checks]/[Total applicable checks] scorecard checks addressed in report.
- Model Seeking: [Number] issues found by model but not in Scorecard.

**Checks Missing from Report:**
- **[Name of Check]**: [Explanation of what was missed]

**Issues Found Only by Model:**
- **[Name of Issue]**: [Explanation of what model found]
```

```
You MUST use the actual numeric values from your analysis for the coverage metrics. For example, if you found that 14 out of
    ↪ 18 checks were covered, write "Model Coverage: 14/18".
You MUST replace bracketed placeholders like '[Actual covered checks]' with the real data from your analysis.
</Task>

<Format>
Call the ScorecardAnalysis tool
</Format}
```

A.3 Detailed Assessment Example: JAX Library Report

To illustrate the detailed report format generated by our system, this subsection presents the complete, multi-page risk assessment report produced by LIBVULNWATCH for the JAX library. This report exemplifies the structure, depth of analysis, and range of risk factors (covering License, Security, Maintenance, Dependencies, and Regulatory domains) assessed for each library. Such detailed reports aim to provide actionable insights for stakeholders. This serves as an exemplar; upon acceptance, all generated reports for the evaluated libraries will be made publicly available via our Hugging Face Space.

Open Source Risk Assessment: JAX ¶

Executive Summary ¶

Risk Score Dashboard ¶

Risk Domain	Rating	Key Finding	Reason for Rating	Key Control
Dependencies	⭐	Complete lack of public dependency management data including no SBOM, vulnerability scanning, or automated update tooling increases critical supply chain risk (Endor Labs 2024 report)	Absence of transparency and controls on dependencies creates a critical unmanaged attack surface	Implement automated SBOM generation, vulnerability scanning, and dependency update tooling (GitHub Dependency Graph)
Regulatory	⭐	No JAX-specific compliance documentation or features for GDPR, HIPAA, AI governance, or explainabil	Lack of regulatory adherence and audit capabilities poses high risk for enterprise and regulated use	Conduct third-party compliance audits and integrate external explainability and privacy tools

Risk Domain	Rating	Key Finding	Reason for Rating	Key Control
		ity (awesome-machine-learning-interpretability)		(SHAP, LIME)
Security	⭐⭐⭐	No reported CVEs in last 24 months but absence of formal security disclosure policy, patch timelines, and documented security testing elevates risk (JAX GitHub)	Limited security process transparency and unsigned artifacts increase vulnerability and supply chain risks	Publish security disclosure policy, formalize patch SLAs, integrate CI/CD security testing, and sign release artifacts (PyPI JAX)
Maintenance	⭐⭐⭐⭐	Frequent monthly releases and large contributor base indicate strong	Active development supports sustainability; however, missing	Define and publish issue response SLOs and optimize issue triage

Risk Domain	Rating	Key Finding	Reason for Rating	Key Control
		maintenance but lack of published issue resolution times poses moderate risk (JAX releases)	response SLAs limit issue management visibility	(JAX Issues)
License	⭐⭐⭐⭐⭐	Core JAX under Apache 2.0 permits broad commercial use; however, associated JAX mouse models impose restrictive Leap License constraints unsuitable for commercial redistribution (JAX	Core software licensing minimizes legal constraints but mouse model licenses carry high legal risk	Restrict use of mouse models to research or conduct detailed legal review before commercial use

Risk Domain	Rating	Key Finding	Reason for Rating	Key Control
		LICENSE, JAX Leap License)		

| **OVERALL** | ⭐⭐⭐ | JAX offers low legal risk for core use but faces high dependency and regulatory risks with moderate security and maintenance gaps (JAX GitHub) | Critical supply chain and compliance weaknesses elevate overall risk despite strong licensing and maintenance foundations | Prioritize remediation of dependency management and regulatory compliance; strengthen security and maintenance policies |

🚨 EMERGENCY ISSUES ¶

[Critical Issue]: JAX has no publicly available Software Bill of Materials (SBOM), dependency vulnerability scanning, or update automation leading to unmanaged critical supply chain exposure (Endor Labs 2024 report) *Immediate Action**: Implement automated SBOM generation, institute regular vulnerability scanning and remediation workflows, and adopt dependency update automation tools (GitHub Dependency Graph)

Top Controls by Priority ¶

1. **Immediate (0-7 days)**: Deploy automated SBOM and vulnerability scanning processes to establish dependency visibility and supply chain security (Endor Labs 2024 report)

2. **Short-term (30 days)**: Publish formal security disclosure policy, patch management timelines, and integrate security testing into CI/CD pipelines (JAX GitHub Security Practices)

3. **Medium-term (90 days)**: Conduct thorough third-party regulatory compliance audits for GDPR, HIPAA, and AI governance; implement external explainability and privacy tools (awesome-machine-learning-interpretability, SHAP)

Comprehensive Risk Mitigation Strategy ¶

Based on all section findings, JAX must adopt a multifaceted approach to address its primary risks:

1. **Technical Controls**:
2. Establish automated SBOM generation and maintain an up-to-date dependency inventory with vulnerability scanning integrated into build workflows (GitHub Dependency Graph)
3. Implement cryptographically signed release artifacts and integrate automated security testing (static/dynamic code analysis) in CI/CD pipelines to improve artifact integrity and detect vulnerabilities early (PyPI JAX)
4. **Policy & Governance Controls**:
5. Publicly document and enforce a coordinated security disclosure and patch response policy with measurable SLAs to improve incident management (JAX GitHub Issues)
6. Define and communicate issue response and resolution SLAs to enhance maintenance transparency and user confidence (JAX Issues)
7. **Legal & Compliance Controls**:
8. Perform comprehensive legal review regarding restrictive JAX mouse model licenses to ensure no unauthorized commercial use (JAX Leap License)

9. Engage external regulatory compliance audits addressing GDPR, HIPAA, explainability, and AI governance requirements; supplement with integration of industry-standard explainability (e.g. SHAP, LIME) and privacy-preserving tools (awesome-machine-learning-interpretability)
   This structured mitigation will enable JAX to substantially reduce its critical supply chain and regulatory risks while enhancing overall security posture and operational transparency.

License Analysis ¶

Executive Overview ¶

JAX core is licensed under Apache License 2.0, a permissive and business-friendly license with explicit patent grants and clear attribution rules; however, JAX-associated mouse models under the Leap License impose restrictive research-only use, indemnities, and sublicensing constraints, presenting a high legal risk for commercial redistribution.

Key Facts & Observations ¶

Risk Factor	Observed Data	Rating (⭐)	Reason for Rating	Key Control
License Type	Apache License 2.0 for JAX core software (JAX LICENSE, Apache 2.0)	⭐⭐⭐⭐⭐	Permissive, OSI-approved, widely compatible license minimizing legal constraints.	Comply with Apache 2.0 license obligations
Commercial Use & Distribution	Allows commercial use, modification, redistribution royalty-free under Apache 2.0 terms (JAX LICENSE)	⭐⭐⭐⭐⭐	Explicitly permits unrestricted commercial use and distribution without fees.	Maintain license and attribution compliance
Patent Grant Provisions	Apache 2.0 provides irrevocable, royalty-free patent license covering contributors' patents	⭐⭐⭐⭐⭐	Strong patent grant reduces litigation risk for users.	Monitor for any external patent claims

Risk Factor	Observed Data	Rating (⭐)	Reason for Rating	Key Control
	(Apache 2.0)			
Attribution Requirements	Requires retention of copyright, license notices, and NOTICE file as per Apache 2.0 (Apache 2.0)	⭐⭐⭐⭐⭐	Clear standard attribution requirements avoid ambiguity in compliance.	Retain all copyright and NOTICE files during reuse

Risk Factor	Observed Data	Rating (⭐)	Reason for Rating	Key Control
JAX Mouse Model Licensing	JAX Leap License includes restrictive research-only use, indemnification mandates, non-transferability, and multiple IP riders including CRISPR/Cas9 license (JAX Leap License)	⭐	Restrictive licensing terms limit commercial use and resale; indemnity and sublicensing clauses add significant legal and operational risk.	Conduct detailed legal review before commercial use or redistribution

Summary ¶

JAX core's Apache 2.0 license ensures low legal risk for commercial and open source use due to its permissive and explicit patent terms. Conversely, JAX-associated mouse models distributed under the Leap License program feature multiple layered, restrictive licenses and indemnities posing high legal risk for commercial redistribution, necessitating explicit license compliance, risk assessment, and legal counsel before use beyond research.

Security Assessment

Executive Overview

JAX has no reported CVEs in the last 24 months, indicating minimal direct vulnerability exposure; however, it suffers from high risk due to no publicly documented security disclosure policy, unclear patch response times, and missing explicit CI/CD security testing evidence.

Key Facts & Observations

Risk Factor	Observed Data	Rating (⭐)	Reason for Rating	Key Control
CVE History	No CVEs explicitly reported against JAX in public CVE databases over past 24 months (JAX GitHub, CVE MITRE)	⭐⭐⭐⭐⭐	Zero known vulnerabilities reported in last 2 years provides strong direct security assurance.	Regular vulnerability scanning and monitoring
Security Disclosure Policy	No public security disclosure or coordinated vulnerability response policy published or linked in official repo (JAX GitHub)	⭐	Absence of a formal disclosure policy delays vulnerability identification and remediation coordination, posing high risk.	Establish and publicly announce a security disclosure policy
Patch Response Time	No documented data on time to patch or	⭐	Unknown patch speed impedes risk	Define patch management SLAs and

Risk Factor	Observed Data	Rating (⭐)	Reason for Rating	Key Control
	security incident response time in repo or issue discussions (JAX GitHub Issues)		mitigation during vulnerabilities, creating uncertainty and elevated risk exposure.	disclose response times
Security Testing Evidence	No explicit mention of security-focused testing, static/dynamic analysis, or CI/CD pipeline test coverage for security in build systems (JAX GitHub Actions)	⭐	Lack of documented security testing means potential vulnerabilities may go undetected increasing risk of exploitation.	Integrate and document automated security testing in CI/CD pipelines
Signed Releases & Binaries	Releases delivered as source code, without public	⭐⭐⭐	Missing signed binaries moderately increases	Provide cryptographically signed release artifacts

Risk Factor	Observed Data	Rating (⭐)	Reason for Rating	Key Control
	cryptographic signing or verification of artifacts (PyPI JAX)		supply chain risk from tampered or compromised releases.	and detailed release notes

Summary ¶

JAX maintains a clean CVE record but presents high security risk due to lack of publicly documented security incident handling policies, undefined patch response processes, and unclear security testing practices. Additionally, unsigned release artifacts expose users to supply chain threats. To reduce risk, JAX should establish and publish comprehensive security policies, enforce security testing in development, and adopt artifact signing practices.

Maintenance Indicators ¶

Executive Overview ¶

JAX demonstrates strong maintenance health with frequent releases and a sizeable contributor base; however, the lack of published issue resolution times constitutes a high maintenance risk.

Key Facts & Observations ¶

Risk Factor	Observed Data	Rating (⭐)	Reason for Rating	Key Control
Latest release date	Last release: June 12, 2024 JAX releases	⭐⭐⭐⭐⭐	Recent release under one month old indicates active maintenance	Continuous release monitoring
Release frequency	Approximately 12 releases in past 12 months JAX releases	⭐⭐⭐⭐⭐	Monthly release cadence supports timely feature additions and bug fixes	Automated CI/CD with scheduled releases
Active contributor count	260 contributors in past year including Google employees and community JAX contributors	⭐⭐⭐⭐⭐	Large and diverse contributor pool supports sustainable development	Encouraging external contributions
Issue resolution metrics	No documented average	⭐	Absence of published issue	Define and publish service-level

Risk Factor	Observed Data	Rating (⭐)	Reason for Rating	Key Control
	issue resolution time; some issues remain open over 30 days JAX issues		resolution SLOs and visible prolonged issue closures pose high risk	objectives for issue response
Packaging workflow	Automated packaging and publishing on PyPI with detailed release notes per version JAX PyPI page	⭐⭐⭐⭐⭐	Well-documented automated publishing assures release integrity	Maintain CI/CD pipelines

Dependency Management ¶

Executive Overview ¶

JAX's dependency management lacks any explicit public disclosure regarding SBOM availability, direct and transitive dependency counts, or vulnerability management, presenting a highly elevated risk profile due to unmonitored supply chain exposure.

Key Facts & Observations ¶

Risk Factor	Observed Data	Rating (⭐)	Reason for Rating	Key Control
SBOM Availability	No public data or documentation about SBOM generation or publication for JAX	⭐	Complete absence of SBOM impedes transparency and security auditing of dependencies	Implement and document automated SBOM generation and distribution
Direct Dependency Count	No explicit information on JAX's number of direct dependencies available	⭐	Unknown dependency scope disables targeted risk evaluation	Conduct full dependency audit and publish list
Transitive Dependency Management	No evidence of visibility or controls over transitive dependencies in JAX ecosystem	⭐	Lack of transitive dependency management increases hidden vulnerability risks	Utilize dependency graph tools that track and label transitive dependencies with remediation guidance

Risk Factor	Observed Data	Rating (⭐)	Reason for Rating	Key Control
Vulnerable Dependencies	No publicly available vulnerability scans or remediation disclosures for JAX dependencies	⭐	Unknown vulnerability status of dependencies poses critical threat to supply chain security	Regularly scan dependencies for vulnerabilities and document fixes
Dependency Update Tools/Policies	No information on use of automated dependency update tools or policies (e.g., Dependabot, Renovate)	⭐	Absence of update automation risks outdated, vulnerable dependencies persisting undetected	Adopt automated dependency update tools and define update policies

---

The complete lack of publicly available dependency management data for JAX — including SBOM availability, dependency counts, visibility into transitive dependencies, vulnerability scanning, and automated update tooling — justifies a HIGH risk rating across all categories. Immediate remediation must prioritize establishing comprehensive SBOM practices, rigorous dependency audits, vulnerability scanning, and automated update mechanisms to mitigate supply chain security weaknesses. Without these controls, JAX’s dependency ecosystem remains a critical unmanaged risk vector.

GitHub’s 2025 update on distinguishing direct vs. transitive dependencies outlines industry best practices that JAX currently does not publicly demonstrate
Endor Labs 2024 report stresses that 95% of vulnerabilities reside in transitive dependencies, underscoring the critical need for transitive dependency management
Absence of SBOM and automated updates severely undermines supply chain security posture as per the 2024 Endor Labs Dependency Management Report

Regulatory Considerations ¶

Executive Overview ¶

JAX presents a high regulatory risk due to the total absence of JAX-specific documentation or features addressing compliance frameworks, data privacy, AI explainability, and governance requirements crucial for enterprise and safety-critical systems.

Key Facts & Observations ¶

Risk Factor	Observed Data	Rating (⭐)	Reason for Rating	Key Control
Compliance Frameworks	No explicit GDPR, HIPAA, or other regulatory compliance documentation or statements found for JAX in official or curated repositories (awesome-machine-learning-interpretability)	⭐	Lack of any compliance documentation or claims leads to high risk of nonconformance with essential legal frameworks	Conduct third-party regulatory audits; integrate compliance tooling
Explainability Features	No built-in explainability or interpretability tools specific to JAX cited in primary sources or curated ML interpretability lists (awesome	⭐	Absence of explainability features poses high risk for regulated AI applications requiring transparen	Use external explainability frameworks e.g., SHAP, LIME

Risk Factor	Observed Data	Rating (⭐)	Reason for Rating	Key Control
	-machine-learning-interpretability)		cy and auditability	
Data Privacy Provisions	No stated data privacy or privacy-preserving mechanisms for JAX found in official documentation or community resources (awesome-machine-learning-interpretability)	⭐	Missing data privacy capabilities increase risk of violating data protection laws e.g., GDPR, CCPA	Implement external privacy engineering, differential privacy tools
Audit Documentation	No evidence of audit trails, usage logs, or AI governance documentation specifically	⭐	Absence of audit and governance documentation creates high compliance risk and hinders	Establish comprehensive logging and governance processes around JAX

A.4 Example Baseline Evaluation: JAX Library Report

Following the full JAX library assessment report presented in Appendix A.3, this subsection provides the corresponding automated baseline evaluation. This evaluation compares LIBVULNWATCH's findings for JAX against the OpenSSF Scorecard, detailing the alignment between the two and highlighting novel risks or deeper contextual insights uniquely identified by our agentic system. This comparative analysis is crucial for understanding the added value and specific strengths of our approach when applied to a specific library assessment.

Listing 9: JAX Library Assessment Evaluation (comparison with OpenSSF Scorecard).

```
**Coverage Summary:**
- Model Coverage: 11/18 scorecard checks addressed in report.
- Model Seeking: 12 issues found by model but not in Scorecard.

**Checks Missing from Report:**
- **Code-Review**: The Scorecard identified low code review approvals (3/10), but the report did not explicitly discuss code
      ↪ review quality or approval ratios.
- **Dangerous-Workflow**: Scorecard verified the absence of dangerous workflows but the full report did not address this
      ↪ workflow security aspect.
- **Dependency-Update-Tool**: Scorecard found update tools (Dependabot) used, but the full report highlighted a critical lack
      ↪ in dependency management and did not discuss presence of update tooling.
- **Fuzzing**: Scorecard noted no fuzzing; the report lacked any mention of fuzz or dynamic testing efforts.
- **Maintainance indicators on issue resolution**: While Scorecard gave a perfect score on Maintained, the report notes
      ↪ absence of issue resolution SLAs and some open issues, indicating a maintenance concern not captured in scorecard
      ↪ summary.
- **Packaging**: Scorecard could not assess; full report noted good automated packaging but did not discuss packaging workflow
      ↪  security.
- **Signed-Releases**: Scorecard could not score; the report discusses unsigned releases raising supply chain risks.

**Issues Found Only by Model:**
- **Absence of SBOM and Dependency Transparency**: The report highlights complete lack of SBOM, transitive dependency
      ↪ management, and vulnerability scanning as critical, absent from Scorecard findings.
- **Security Process Gaps**: Missing published security disclosure policies, patch SLAs, and CI/CD security integration not
      ↪ described by Scorecard.
- **Regulatory and Compliance Risks**: High regulatory risk with no GDPR, HIPAA, AI governance, or explainability support
      ↪ fully discussed only by model.
- **Legal Licensing Limitations for Mouse Models**: The restrictive and risky JAX Leap License for mouse models posing
      ↪ commercial legal risks not detected by Scorecard.
- **Dependency Vulnerability and Update Weaknesses**: While Scorecard found some update tooling, the model reveals severe
      ↪ vulnerability management gaps.
- **Token Permissions Excessive**: Scorecard flags token permission issues; the report does not discuss token permission risks
      ↪ .
- **Vulnerabilities Present**: Scorecard reports 18 existing vulnerabilities; the full report sees no recent CVEs and thus
      ↪ conflicts on direct vulnerability findings.

The model identified more nuanced regulatory, legal, and dependency supply chain details that the Scorecard metrics alone did
      ↪ not reveal, while Scorecard provided some workflow and token permissions insights not covered by the model.
```