Aegis: Towards Agentic Support for Software Artifact Reviews

Konstantin Kirchheim[0000−0001−5819−7692], Sebastian Nielebock[0000−0002−0147−3526], and Frank Ortmeier[0000−0001−6186−4142]

Otto-von-Guericke University Magdeburg, Germany
{firstname}.{lastname}@ovgu.de

Abstract. Researchers frequently supplement their paper with software artifacts, either as a means to obtain their results or as a research outcome itself. However, assessing the validity and reusability of software artifacts – either from an (internal) author or an (external) reviewer perspective – remains a largely manual and time-consuming task. Reviewers must inspect artifacts, draw connections to the corresponding paper, and determine whether the provided artifacts are sufficiently documented, accessible, and executable. In this work, we propose Aegis, a system that aims to assist software artifact reviews with large language models (LLMs). We cast artifact review as a process of evidence generation and interpretation. Starting from a scientific paper, Aegis extracts referenced artifacts and collects heterogeneous evidence from both the paper and the artifacts by using 1) human-specified functions, 2) semantic analysis via LLMs, and 3) software agents interacting with the artifacts in an isolated execution environment. The collected evidence is aggregated into a structured, multidimensional assessment of aspects of artifact quality that reviewers can then interpret. We implement Aegis as a prototype with extensible, plugin-based analysis components and demonstrate its feasibility through three case studies of repositories accompanying pattern recognition papers. Our code is publicly available.[1]

Keywords: artifact review · reproducibility · large language models · research software

1 Introduction

The reproducibility of results is a major pillar of science. According to the *Association for Computing Machinery* (ACM)[2], the term *reproducibility*[3] denotes the independent validation of research results by different researchers using the

[1] https://github.com/kkirchheim/aegis

[2] https://www.acm.org/publications/policies/artifact-review-and-badging-current accessed 2026/04/20

[3] *Replicability*, instead, refers to the independent validation by a different research team using a different experimental setup. In the past, the interpretation of both terms was often swapped. We stick to the above definition, if not stated otherwise.

same experimental setup, data, and tools. The popular article by Baker [3] reports a trend towards a *reproducibility crisis*, evidenced by failed attempts to reproduce many research results across a variety of disciplines. This crisis also affects computer science [6, 10], and artificial intelligence research [7].

As a particularity of the field, computer science research often produces *software artifacts*, either as a means to conduct experiments, or as a research outcome *per se*. Sometimes, scientists aim to make their artifacts reproducible by making them *reusable* by other researchers. This enables other parties to independently validate the artifacts – and thereby the research results – as well as to reuse these artifacts for subsequent research [10]. Moreover, software artifacts also constitute an integral part of transferring research into industry [11, 14].

Some conferences and journals validate the reproducibility of such software artifacts by conducting so-called *software artifact reviews*. In addition to peer review of the papers, this evaluates the associated artifacts with respect to different criteria of reusability and reproducibility [10, 40]. Both the production of reusable software and the review of software artifacts entail additional effort for authors and reviewers alike. However, expending this effort is only weakly incentivized for early-career scientists (in contrast to, e.g., the quantity of papers and citations [40]). Thus, as an incentive, artifact reviews can award badges, such as ACM badges.[2] Badges indicate certain reproducibility characteristics based on the manual assessment following guidelines and best practices.

Artifact reviews constitute an effective means for identifying reproducibility characteristics. Winter et al. [40] found that papers rewarded with the ACM Available badge also more frequently reference their artifacts ($\approx 98\%$) and more referenced artifacts are accessible ($\approx 99\%$) compared to papers without that badge ($\approx 55\%$ are referenced and $\approx 90\%$ of referenced artifacts are accessible).

Yet, an artifact review remains a time- and resource-consuming manual task [37]. The diversity of software artifacts (in terms of programming language, execution environment, required hardware, code structure, or application domain) limits the automated support of reviews [19]. Furthermore, some reproducibility criteria (cf. Sec. 2.1) are vague and more subjective.

With recent advances in large language models (LLMs) and their integration with tools to inspect, edit, and execute files [43] in execution environments [21] – so-called *agents* – these previous obstacles could be overcome. We hypothesize that LLMs help to integrate human-written review criteria, paper content, research artifact, code repository, and interpretable results.

In this work, we first summarize existing approaches for automated support of artifact reviews. Based on our findings, we propose AEGIS (***A**rtifact **E**vidence **G**eneration and **I**nterpretation **S**ystem*), a framework for supporting artifact reviews via (agentic) LLMs.

Furthermore, we present a proof-of-concept implementation as a plugin-based, hybrid tool for artifact review that leverages LLM- and agent-based approaches alongside conventional automation. We assess its applicability based on three case studies from previous RRPR workshops, comparing the evidence collected by the system with that of a human reviewer. Our main goal is to ini-

tiate a discussion about aspects of software artifact review, some of which could potentially be addressed in subsequent iterations of AEGIS.

In general, AEGIS aims to support external reviewers in validating the quality of software artifacts and their ability to produce the reported research results. It also supports the authors in preparing the dissemination of their artifact and in assessing its reusability beforehand.

To summarize, we make the following contributions:

- an *overview of guidelines and existing automation approaches for artifact reviews* (Sec. 2).
- *proposal of AEGIS*, a framework for evidence-centered support of software artifact reviews that combines deterministic repository inspection, LLM-based semantic analysis, and agent-driven experiment exploration without reducing the review to a single score or badge.
- *demonstration of the applicability of AEGIS* in three case studies (Sec. 4) and a *discussion of the main results* (Sec. 5).
- *provision of AEGIS' source code* for independent reproduction and reuse[1].

2 Background and Related Work

In this section, we introduce an overview of the criteria of *good* software artifacts and guidelines for artifact reviews ensuring these criteria (Sec. 2.1), as well as an overview of techniques and datasets to automate artifact reviews (Sec. 2.2).

2.1 Review Criteria and Guidelines

There exists a diverse set of criteria on what contributes to a *good* software research artifact. Many criteria are based on the FAIR (i.e., **F**indable, **A**ccessible, **I**nteroperable, **R**eusable) principles for scientific data [38]. In previous work, Heumüller *et al.* [18] discussed six guidelines of artifact review based on different journals (i.e., ESE [26] and JOSS [22]/JORS[4]) and conferences (i.e., TACAS[5] and CAV[6] both in 2019), institutions (i.e., ACM[2] [6] and NASA[7]), and a publication by Wilson *et al.* [39]. This way, 14 criteria were derived, namely

1. providing a *runnable software*
2. providing a sufficient *documentation* of the artifact
3. labeling the artifact with a *unique identifier or name*
4. providing the artifact in an *accessible repository*
5. storing the artifact in a *persistent* way
6. providing the artifact as a *special distribution* (e.g., for automated validation)

[4] `https://openresearchsoftware.metajnl.com/about/editorialpolicies` accessed 2026-05-28

[5] `https://conf.researchr.org/track/etaps-2019/tacas-2019-papers#Artifact-Evaluation` accessed 2026-05-12

[6] Website not available anymore as of 2026-05-12

[7] `https://code.nasa.gov/` accessed 2026-05-12

7. providing a *software license*
8. providing the *experimental data*
9. providing a *small test data set* (e.g., for fast validation)
10. providing a list of *open issues* of the artifact
11. complying with common *software engineering practices*
12. complying with *law and regulations*
13. providing a *list of contributors and contact information*
14. having certain *other requirements* (e.g., security regulations)

These criteria were also suggested in other publications, for instance, runnable software [32], documentation [35], unique identifiers [15, 35], accessible repository [10,15,36], and licenses [32,36]. In addition, the related work also suggests the following criteria:

- referencing *the artifact and data* in the publication [15, 36]
- making the *artifact and data free and accessible* [10, 15, 23, 35, 36]
- using a transparent *workflow tracking* [23, 36]
- using *citation standards for software* [35, 36]
- enabling researchers to *efficiently reuse* artifact code [10]
- producing the reported results [10, 15] in a *deterministic* way [23]
- clear *connection between written manuscript and software artifact* [23]

In a preprint of practices to reproduce LLM-based experiments in the domain of software engineering, Siddiq *et al.* [32] introduced the *Reproducibility maturity model (RMM)* consisting of four maturity levels, RMM-0 to RMM-3, each with increasing maturity. They assess five aspects, namely, *accessibility*, *environment specification*, *versioning*, *executability*, and *legal aspects*. These aspects are provided as a checklist consisting of four questions for each maturity level. We use the RMM as criteria in our case studies (cf. Sec. 4) since the simple questionnaire design makes it particularly suitable for our cases. This serves as starting point of Aegis since it main goal is to adapt to various applied criteria.

We observe that criteria range between very strict and easily decidable factors (e.g., repository contains license file, unique identifier), to more subjective ones (e.g., efficiently usable artifact, clear connection between paper and artifact), as well as executable-dependent (e.g., produces reported results) and workflow-related (e.g., transparent workflow tracking) criteria. Answering the questions requires different resources in various forms, such as the written paper (consisting of text, figures, and tables), the code repository, execution files (e.g., logs and stack traces), and additional information (e.g., license files).

2.2 Automating Artifact Reviews

Although automating artifact reviews is challenging and currently rarely implemented in artifact evaluation tracks, some attempts have been made to automate these steps. Automating artifact reviews, aiming to automate reproduction, requires using the same artifact to produce the research results. However, techniques automating replicability are also related to this topic, since they use

information from a paper to independently recreate the artifact. We reflect on these attempts by distinguishing between *conventional automation* (i.e., without LLM- or machine learning-based support) and *LLM-based automation*. We also discuss existing *benchmarks* for automated artifact reviews.

Conventional Automation We only found a limited amount of research that automates artifact reviews in a conventional (i.e., non-LLM-based) way. *ReproZip* [9] is a tool that tracks actions performed on an evaluation processing machine and generates a script with the execution environment as a self-contained package. Fursin et al. [13] provided a framework, denoted as *Collective Knowledge (CK)*, that converts scientific data and code into reusable and publishable artifacts. However, at the time of writing, their framework is no longer accessible.

LLM-based Automation A growing body of work studies LLMs for reproduction and replicability. For artifact reproduction, *Reproscreener* [4] uses GPT-4 and prior reproducibility metrics [17,23] to assess a paper's LaTeX source and artifact repository through data collection, evaluation, and reporting. Its paper analysis combines LLM-based and keyword-based methods, while repository analysis focuses on structural features. Heye et al. [19] propose a similar three-stage approach for security-related publications, assessing reproducibility, source-code quality, and methodological pitfalls.

Recent preprints further explore agentic reproduction. *Artisan* [2] uses the paper, extracted table values, related URLs, and the artifact to generate a reproduction script. *ArtifactCoPilot* [41] uses agents to reconstruct repository workflows, unify execution environments, and assess artifacts in terms of ACM badges.

Other work targets replicability rather than reproduction. Liang *et al.* [24] evaluate GPT-4 for replicating empirical software-engineering studies by extracting assumptions and generating analysis code, finding reasonable high-level code but weaker low-level software-engineering quality. *PaperCoder* [31] generates code repositories via planning, implementation-detail analysis, and code generation. *Deep-Reproducer* [8] summarizes a paper, retrieves related papers and repositories, and evolves the most relevant repository into an implementation of the analyzed paper.

Benchmarks Table 1 provides an overview of benchmarks for evaluating the automation of artifact reviews, along with their sizes (i.e., the number of publications and/or tasks). In general, benchmarks can be categorized into two types: The first type [1, 2, 4, 20, 28, 31] assess the *general reproducibility of the artifact* using various criteria (e.g., existing badges, reproducibility metrics, and code flaws). The second type [2, 5, 33, 42] defines certain reproducibility tasks – such as reproducing table values in a paper, successfully building an artifact, *etc.* – and assess which tasks an automated review can successfully complete.

For one publication [41], the benchmark was not referenced in the pre-print manuscript at the time of writing. For all but two publications [5,33], the benchmark is limited to a single research domain.

Table 1. Overview of benchmarks for automating artifact reviews. Gray entries refer to non-peer-reviewed pre-prints

Dataset	Domain	#Publications	#Tasks	Reference publication	Reference artifact repository	Contains tasks	Reproducibility assessment	Task assessment
Arp et al. [1]	Security with ML	30	-	✗	✗	✗	✓	✗
Olszewski et al. [28]	Security with ML	744	-	✓	✓	✗	✓	✗
SUPER [5]	General	N/A	45+152+604	✗	✓	✓	✗	✓
Reproscreener Dataset [4]	ML	50	-	✓	✓	✗	✓	✗
CORE-Bench [33]	Computer Science, Social Sciences, Medicine	90	270	✗	✓	✓	✗	✓
REPRO-Bench [20]	Social Sciences	112	-	✓	✓	✗	✓	✗
SciReplicate-Bench [42]	NLP	36	100	✓	✓	✓	✗	✓
Paper2CodeBench [31]	ML	90	-	✓	✓	✗	✓	✗
PaperBench & PaperBench Code-Dev [34]	ML	20	-	✓	✓	✗	✓	✗
Artisan-Bench [2]	Software Engineering	23	60	✓	✓	✓	✓	✓
Wu et al. [41]	Software Engineering	48	-	N/A	N/A	-	N/A	-

The reviewed systems and benchmarks suggest two dominant framings of automated artifact review. The first treats review as an artifact-level assessment problem, for example by predicting reproducibility indicators, identifying code-quality issues, or assigning badge-like judgments. The second treats review as a task-completion problem, for example by reproducing selected numerical results, building an artifact, or generating a reproduction script. Both framings are valuable, but they only partially match the practical needs of artifact reviewers. Reviewers often require not only a final judgment, but also auditable evidence: which dependency failed, whether the documentation is sufficient, whether a repository matches the paper, which assumptions were required during execution, and where uncertainty remains.

Aegis is positioned at this evidence layer. Rather than optimizing for a single badge, score, or benchmark task, it collects heterogeneous evidence from deterministic checks, semantic LLM-based analyses, and agentic interaction with the artifact, and organizes this evidence into a structured review context. We therefore evaluate Aegis qualitatively on real artifact-review cases using the RMM checklist as an interpretable criterion set, rather than as a complete ground-truth benchmark.

To avoid extensive manual reassessment of the benchmarks, we apply Aegis to single case studies, thereby obtaining a qualitative evaluation (cf. Sec. 4). This qualitative evaluation demonstrates the applicability of Aegis to real-world artifacts and highlights practical limitations for future work.

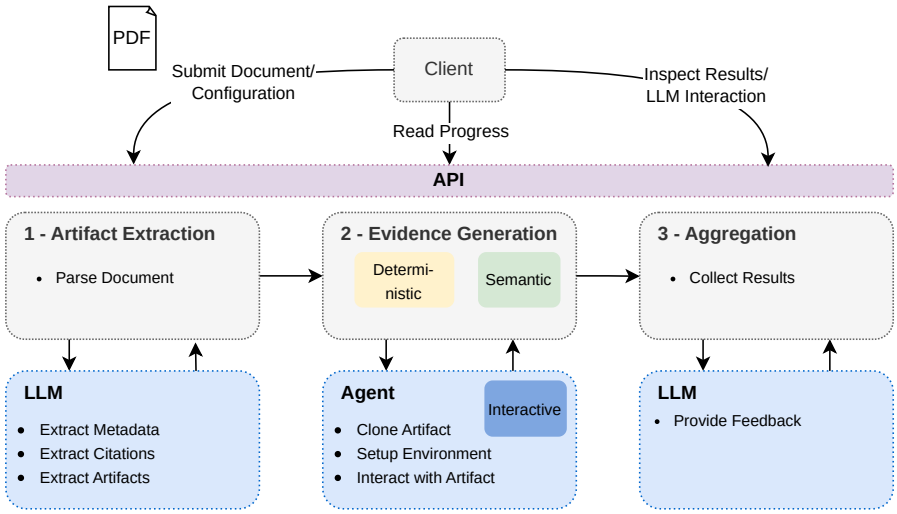

Fig. 1. Overview of the architecture for the proposed implementation of AEGIS. A publication is first parsed to extract metadata, including references to artifacts. Evidence is then collected from the paper and the referenced artifacts using complementary deterministic, semantic, and interactive analysis components. The resulting evidence is aggregated into a structured review context that supports human and LLM-assisted artifact assessment.

3 AEGIS: Agentic Artifact Review

In this work, we frame automated artifact review as a process of collecting and interpreting evidence about research artifacts referenced in a scientific paper. Rather than reducing reproducibility to a binary outcome, the goal is to provide supporting signals that capture reproducibility criteria (cf. Sec. 2.1).

Scientific papers typically reference external artifacts such as code repositories, datasets, or pretrained models. The review process, therefore, begins by identifying these artifacts and linking them to the paper's content. AEGIS uses this linkage to collect heterogeneous evidence from both paper and artifact, including repository structure, dependencies, documentation, and execution traces.

Different review criteria require different forms of analysis. Thus, AEGIS combines complementary analysis forms, ranging from static inspection via deterministic scripts to semantic evaluation via LLMs, and interactive exploration via agents. To assist human reviewers, the resulting evidence is not collapsed into a single score but organized into a structured report for further inspection.

Because much of this evidence is textual, LLMs provide a flexible interface for aggregating and interpreting it. They enable flexible querying of the collected evidence and support forming context-aware judgments about artifact quality.

3.1 Architecture

Aegis follows a three-stage pipeline: artifact extraction, evidence generation, and aggregation (cf. Fig. 1).

Extraction The process begins with a paper document and a specification of the available execution context. The paper is converted into machine-readable text and analyzed to extract metadata and references to relevant artifacts. This includes bibliographic information, cited sources, references to own and other software artifacts such as code repositories, and additional resources such as datasets or external services.

Evidence Generation The second stage collects evidence about the extracted artifacts using three complementary analysis mechanisms. *Deterministic evidence generators* evaluate explicitly specified properties, such as repository structure, configuration files, dependency specifications, and the presence of documentation or licenses. *Semantic evidence generators* address aspects that cannot be easily captured by human-specified programs. Implemented as prompt-based plugins, they use LLMs to assess properties such as documentation clarity, inferred resource requirements, and the correspondence between the repository and the method described in the paper. *Interactive evidence* is obtained through direct engagement with the artifact. An agentic LLM explores the repository in a controlled and isolated environment, following setup instructions and executing commands to adhere to the intended workflow. This produces execution traces and observations that complement static and semantic analyses.

Aggregation All collected evidence is consolidated into a structured report that integrates metadata, analysis results, and interaction traces. The resulting report can be inspected directly or queried through an LLM-based interface, enabling targeted analysis and follow-up questions. Because much of the evidence is textual, this representation remains both human-interpretable and well-suited for language-model-based reasoning.

3.2 Implementation

We implement the proposed framework as a modular prototype Aegis using mainly AI-driven software engineering. Papers are parsed from PDF into machine-readable text, after which an LLM extracts structured metadata and identifies referenced artifacts. While the design is general, Aegis focuses on software artifacts hosted in Git repositories.

We realized evidence generation through a plugin architecture combined with agentic interaction. We have two types of plugins – one for deterministic and one for semantic analyses: the former encode explicit repository checks, while the latter define prompt-based evaluation criteria for LLM-based assessment. Each plugin produces structured outputs, including results and execution logs.

Table 2. Assessment of three case studies against the RMM criteria. Symbols denote AEGIS' criterion-level judgment: ✓= satisfied, ✗= not satisfied, ? = inconclusive. Cell colors denote manual validation: blue = correct, red = incorrect, gray = inconclusive.

	RMM-0				RMM-1				RMM-2				RMM-3			
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
	Artifacts missing?	Env. details insufficient?	Versions unspecified?	Legal ambiguity?	Install brittle?	Versions not fully pinned?	Pipeline not automated?	Licenses unclear?	Container provided?	Dependencies fully pinned?	Data/models persistently hosted?	Licenses complete?	Exact reproduction possible?	Independent reproduction?	Outputs consistent?	Clean environment success?
Case Study 1 [12]	✓	✗	✗	✓	✓	✗	✓	?	✗	✗	✗	?	✗	?	?	✗
Case Study 2 [29]	✓	✓	✗	✓	✓	✗	✓	✓	✗	✗	✓	✓	✓	✓	?	✓
Case Study 3 [30]	✗	✗	✗	?	✓	✗	✓	✗	✗	✗	✓	✗	✗	✓	✗	✓

Interactive evidence is obtained through an agent that explores artifacts in a controlled environment. The agent is invoked once per artifact to reproduce its intended usage. For example, it sets up dependencies, executes the provided scripts, and tracks the resulting traces.

All outputs are consolidated into a unified, structured representation. An LLM-based interface can inspect or query this representation as review context.

From a software engineering perspective, AEGIS follows a service-oriented design with an OpenAPI-compatible interface, enabling integration with external workflows. Our implementation provides both a command-line interface for batch execution and a web-based interface for interactive inspection. The plugin architecture allows new analyses to be added without modifying the core system. Although AEGIS targets Git-based artifacts in computer vision and pattern recognition, the design is intended to generalize to other domains.

4 Case Studies

In this section, we evaluate AEGIS on scientific papers and their accompanying software artifacts. AEGIS assessed reproducibility using the Reproducibility Maturity Model (RMM) [32]. For both semantic analysis and interactive exploration, we employed *Claude Haiku 4.5*, with a budget of 30 interaction steps per artifact.

For validation, we selected three papers from the Workshop on Reproducible Research in Pattern Recognition (RRPR) with manually pre-assessed artifacts. This allows us to compare AEGIS' judgments against artifacts deemed reproducible through prior manual inspection. When repositories were not explicitly linked in the paper, we manually provided the corresponding URLs. Execution environments were selected based on prior inspection.

Table 2 summarizes Aegis' RMM assessments and their manual validation. The first author manually validated the RMM criteria for selected submissions that had passed the RRPR reproducibility assessment. This validation is intended as a qualitative plausibility check rather than an independent benchmark annotation. We distinguish between evidence generated autonomously by Aegis and evidence obtained after limited reviewer-provided environment hints; the latter reflects a reviewer-in-the-loop usage mode rather than fully autonomous reproduction.

4.1 Case Study 1: Ferreira et al. [12]

We analyzed the artifact by Ferreira et al. [12]. Aegis reported that 4 out of 16 RMM checks passed, while 4 were marked as unclear. The agent could execute parts of the code, but encountered an exception in one of the scripts. This was caused by loose version constraints (e.g., `numpy>=1.8.1`), which led to incompatibilities when the newest available version is installed. To resolve this, we manually selected a Python 3.6 environment and provided the agent with explicit instructions to install the correct NumPy version. In this execution context, the agent could execute the full pipeline and correctly identified several issues, such as underspecified versions of the execution environment.

4.2 Case Study 2: Oskarsson [29]

For Oskarsson's artifact [29], Aegis reported 11 out of 16 RMM checks as satisfied, with 1 marked as unclear. The agent successfully executed the code and produced the plots and figures from the paper; however, the text-only agent could not cross-check the produced figures against those in the paper, which it correctly reported. Aegis correctly flagged that the OS, libraries, and MATLAB version used had not been fully specified.

4.3 Case Study 3: Qin et al. [30]

We further analyzed the artifact by Qin et al. [30]. Aegis reported that 4 of the 16 RMM checks were satisfied, with 1 being marked as unclear. The agent could run the provided script successfully. It correctly identified that the repository was missing some of the test images referenced in the paper, lacks a license file, and underspecifies dependencies (e.g., OS and MATLAB version).

5 Discussion

Across the case studies, the agent executed substantial parts of the provided code despite incomplete or underspecified environments. It also identified several legitimate reproducibility concerns in line with the RMM, particularly regarding dependency specification, environment configuration, and documentation.

However, full reproduction sometimes required limited manual intervention, such as selecting a suitable execution environment or resolving implicit assumptions not documented in the artifact. Thus, Aegis does not eliminate human involvement, but can reduce the effort needed to diagnose and address reproducibility issues. We next discuss broader implications and limitations.

5.1 Ephemeral Reproducibility

As observed in Section 4.1, reproducibility is not a static property of an artifact, but depends on an execution context that may evolve over time. Changes in dependencies, external services, or hardware environments can render previously executable artifacts unusable. This is particularly evident when dependencies are not precisely pinned or become unavailable, allowing future installations to resolve to incompatible versions.

Reproducibility assessments should therefore be understood as time-dependent and context-specific rather than permanent certifications. In this sense, artifact badges or reproducibility claims capture the state of an artifact under a particular configuration at a particular time.

Aegis can make such dependencies explicit by inspecting environment specifications and detecting sources of instability, such as unpinned requirements. By exposing these properties as evidence, it supports more robust and forward-looking assessments of artifact quality.

5.2 Interpreting Agent Failures

When an automated agent cannot execute an artifact, the cause is often ambiguous. Failures may stem from agent limitations, insufficient documentation, or genuine artifact issues, and distinguishing these factors is inherently difficult.

Nevertheless, agentic execution offers a useful perspective on usability. An artifact that can be processed by an agent following general instructions exhibits procedural clarity likely beneficial for human users as well. Conversely, agent failures can indicate missing documentation, implicit assumptions, or fragile setup procedures.

Agent-based evaluation therefore does not replace human judgment, but provides a lower bound on artifact usability under standardized conditions. By recording execution traces and failure modes, Aegis allows reviewers to interpret such outcomes in context.

5.3 Interaction Modalities and Limitations

Some artifacts require graphical user interfaces (GUIs), visual inspection of plots, or other modalities not yet fully supported by current agent-based systems, as illustrated in Section 4.2. This limits automated assessment of artifacts that rely on interactive or visualization-heavy workflows.

Recent work on vision–language agents shows the feasibility of GUI control through screenshot-based perception and action generation [25, 27]. However,

such agents are not yet robust enough for general artifact evaluation; Aegis therefore focuses on command-line and script-based interaction.

5.4 Robustness and Adversarial Considerations

Using LLMs in artifact review introduces vulnerabilities, including prompt injection [16, 44] and other adversarial manipulation. Such attacks may try to influence automated assessments through misleading instructions embedded in artifacts or documentation.

This underscores the importance of controlled execution environments, transparent logging, and preserving intermediate evidence. Aegis retains generated evidence, including prompts and execution traces, in the review context, enabling manual post-hoc inspection and reducing reliance on opaque model outputs.

6 Conclusion & Outlook

We presented Aegis, a modular prototype for supporting software artifact assessment through structured evidence collection. It combines deterministic analyses, LLM-based semantic evaluation, and agent-based interaction, and was evaluated on three pattern-recognition papers.

The results indicate that Aegis can surface relevant reproducibility issues while partially automating artifact inspection. However, human oversight remains necessary, especially for resolving implicit assumptions and selecting suitable execution environments. Aegis should therefore be understood as supporting, not replacing, artifact review.

Such approaches can enable more scalable review processes, for example, in conference settings. We aim to improve Aegis by adapting agents to this task and extending the framework to support automated feedback and artifact repair.

References

1. Arp, D., Quiring, E., Pendlebury, F., Warnecke, A., Pierazzi, F., Wressnegger, C., Cavallaro, L., Rieck, K.: Dos and Don'ts of Machine Learning in Computer Security. In: Proc. 31st USENIX Secur. Symp. (USENIX). pp. 3971–3988 (2022), https://www.usenix.org/conference/usenixsecurity22/presentation/arp
2. Baek, D., Pradel, M.: Artisan: Agentic Artifact Evaluation (2026), https://arxiv.org/abs/2602.10046
3. Baker, M.: 1,500 scientists lift the lid on reproducibility. Nature **533**, 452–454 (May 2016). https://doi.org/https://doi.org/10.1038/533452a
4. Bhaskar, A., Stodden, V.: Reproscreener: Leveraging LLMs for Assessing Computational Reproducibility of Machine Learning Pipelines. In: Proc. 2nd ACM Conf. Reprod. Replica. (REP). p. 101–109 (2024). https://doi.org/10.1145/3641525.3663629
5. Bogin, B., Yang, K., Gupta, S., Richardson, K., Bransom, E., Clark, P., Sabharwal, A., Khot, T.: SUPER: Evaluating Agents on Setting Up and Executing Tasks from Research Repositories. In: Proc. 2024 Conf. Empir. Methods Nat. Lang. Process.

(EMNLP). pp. 12622–12645. ACL (Nov 2024). https://doi.org/10.18653/v1/2024.emnlp-main.702
6. Boisvert, R.F.: Incentivizing reproducibility. Commun. ACM **59**(10), 5 (Sep 2016). https://doi.org/10.1145/2994031
7. Bouthillier, X., Laurent, C., Vincent, P.: Unreproducible Research is Reproducible. In: Proc. 36th Int. Conf. Mach. Learn. (PMLR). p. 725–734 (2019), https://proceedings.mlr.press/v97/bouthillier19a.html
8. Chen, P., Yan, N., Zhao, Z., Lin, Y., Chen, H., Hu, Y., Bai, Q., Li, X., Mortazavi, M.S.: Deep-Reproducer: From Paper Understanding to Code Generation. In: NeurIPS 2025 4th Work. Deep Learn. Code (2025), https://openreview.net/forum?id=zw2DpSxnXn
9. Chirigati, F., Rampin, R., Shasha, D., Freire, J.: ReproZip: Computational Reproducibility With Ease. In: Proc. 2016 Int. Conf. Manag. Data (SIGMOD/PODS). p. 2085–2088 (2016). https://doi.org/10.1145/2882903.2899401
10. Collberg, C., Proebsting, T.A.: Repeatability in Computer Systems Research. Commun. ACM **59**(3), 62–69 (Feb 2016). https://doi.org/10.1145/2812803
11. Diebold, P., Vetrò, A.: Bridging the Gap: SE Technology Transfer into Practice: Study Design and Preliminary Results. In: Proc. 8th ACM/IEEE Int. Symp. Empir. Softw. Eng. Meas. (ESEM) (2014). https://doi.org/10.1145/2652524.2652552
12. Ferreira, Á.R., de Rosa, G.H., Papa, J.P., Carneiro, G., Faria, F.A.: Creating classifier ensembles through meta-heuristic algorithms for aerial scene classification. In: Proc. 25th Int. Conf. Pattern Recognit. (ICPR). pp. 415–422. IEEE (2021). https://doi.org/10.1109/ICPR48806.2021.9412938
13. Fursin, G., Lokhmotov, A., Plowman, E.: Collective Knowledge: Towards R&D sustainability. In: Proc. 2016 Des., Autom. & Test Eur. Conf. & Exhib. (DATE). pp. 864–869 (2016), https://ieeexplore.ieee.org/abstract/document/7459430
14. Garousi, V., Petersen, K., Ozkan, B.: Challenges and best practices in industry-academia collaborations in software engineering: A systematic literature review. Elsevier J. Inf. Softw. Technol. **79**, 106–127 (2016). https://doi.org/10.1016/j.infsof.2016.07.006
15. Gonzalez-Barahona, J.M., Robles, G.: Revisiting the reproducibility of empirical software engineering studies based on data retrieved from development repositories. Elsevier J. Inf. Softw. Technol. **164**, 107318 (2023). https://doi.org/10.1016/j.infsof.2023.107318
16. Greshake, K., Abdelnabi, S., Mishra, S., Endres, C., Holz, T., Fritz, M.: Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. In: Proc. 16th ACM Work. Artif. Intell. Secur. (AISec). pp. 79–90 (2023). https://doi.org/10.1145/3605764.3623985
17. Gundersen, O.E., Kjensmo, S.: State of the Art: Reproducibility in Artificial Intelligence. Proc. 32nd AAAI Conf. Artif. Intell. (AAAI) **32**(1) (Apr 2018). https://doi.org/10.1609/aaai.v32i1.11503
18. Heumüller, R., Nielebock, S., Krüger, J., Ortmeier, F.: Publish or perish, but do not forget your software artifacts. Springer Empir. Softw. Eng. pp. 1–32 (2020). https://doi.org/10.1007/s10664-020-09851-6
19. Heye, D., Kindermann, K., Decker, R., Lohmöller, J., Belova, A., Geisler, S., Wehrle, K., Pennekamp, J.: Supporting Artifact Evaluation with LLMs: A Study with Published Security Research Papers. In: Proc. 13th IEEE Int. Conf. Big Data (BigData). pp. 5077–5085 (2025). https://doi.org/10.1109/BigData66926.2025.11401815

20. Hu, C., Zhang, L., Lim, Y., Wadhwani, A., Peters, A., Kang, D.: REPRO-Bench: Can Agentic AI Systems Assess the Reproducibility of Social Science Research? In: Find. Assoc. Comput. Linguist.: ACL 2025. pp. 23616–23626 (Jul 2025). https://doi.org/10.18653/v1/2025.findings-acl.1210
21. Hu, R., Peng, C., XinchenWang, Xu, J., Gao, C.: Repo2Run: Automated Building Executable Environment for Code Repository at Scale. In: Adv. Neural Inf. Process. Syst. (NeurIPS). vol. 38, pp. 32679–32718. Curran Associates, Inc. (2025), https://proceedings.neurips.cc/paper_files/paper/2025/hash/2f2b1d6bbd50865eca40e2774a057eef-Abstract-Conference.html
22. Katz, D.S., Niemeyer, K.E., Smith, A.M.: Publish Your Software: Introducing the Journal of Open Source Software (JOSS). Comput. Sci. & Eng. **20**(3), 84–88 (2018). https://doi.org/10.1109/MCSE.2018.03221930
23. Krafczyk, M.S., Shi, A., Bhaskar, A., Marinov, D., Stodden, V.: Learning from reproducing computational results: introducing three principles and the Reproduction Package. Philos. Trans. Royal Soc. A: Math., Phys. Eng. Sci. **379**(2197), 1–28 (Mar 2021). https://doi.org/10.1098/rsta.2020.0069
24. Liang, J.T., Badea, C., Bird, C., DeLine, R., Ford, D., Forsgren, N., Zimmermann, T.: Can GPT-4 Replicate Empirical Software Engineering Research? Proc. ACM Soft. Eng. **1**(FSE) (Jul 2024). https://doi.org/10.1145/3660767
25. Lin, K.Q., Li, L., Gao, D., Yang, Z., Wu, S., Bai, Z., Lei, S.W., Wang, L., Shou, M.Z.: ShowUi: One Vision-Language-Action Model for GUI Visual Agent. In: 2025 IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR). pp. 19498–19508 (2025), https://openaccess.thecvf.com/content/CVPR2025/html/Lin_ShowUI_One_Vision-Language-Action_Model_for_GUI_Visual_Agent_CVPR_2025_paper.html
26. Méndez Fernández, D., Monperrus, M., Feldt, R., Zimmermann, T.: The Open Science Initiative of the Empirical Software Engineering Journal. Springer Empir. Softw. Eng. **24**(3), 1057–1060 (2019). https://doi.org/10.1007/s10664-019-09712-x
27. Niu, R., Li, J., Wang, S., Fu, Y., Hu, X., Leng, X., Kong, H., Chang, Y., Wang, Q.: ScreenAgent: A Vision Language Model-driven Computer Control Agent. In: Proc. 33rd Int. Jt. Conf. Artif. Intell. (IJCAI). pp. 6433–6441 (8 2024). https://doi.org/10.24963/ijcai.2024/711
28. Olszewski, D., Lu, A., Stillman, C., Warren, K., Kitroser, C., Pascual, A., Ukirde, D., Butler, K., Traynor, P.: "Get in Researchers; We're Measuring Reproducibility": A Reproducibility Study of Machine Learning Papers in Tier 1 Security Conferences. In: Proc. 2023 ACM SIGSAC Conf. Comput. Commun. Secur. (CCS). p. 3433–3459 (2023). https://doi.org/10.1145/3576915.3623130
29. Oskarsson, M.: Characterizing the structure tensor using gamma distributions. In: 2016 23rd Int. Conf. Pattern Recognit. (ICPR). pp. 763–768. IEEE (2016). https://doi.org/10.1109/ICPR.2016.7900154
30. Qin, X., Jagersand, M., Yang, X., Wang, J.: Building facade recognition from aerial images using Delaunay Triangulation induced feature perceptual grouping. In: 2016 23rd Int. Conf. Pattern Recognit. (ICPR). pp. 3368–3373. IEEE (2016). https://doi.org/10.1109/ICPR.2016.7900154
31. Seo, M., Baek, J., Lee, S., Hwang, S.J.: Paper2Code: Automating Code Generation from Scientific Papers in Machine Learning. In: NeurIPS 2025 4th Work. Deep Learn. Code (2025), https://openreview.net/forum?id=sehFd0idWo
32. Siddiq, M.L., Islam-Gomes, A., Sekerak, N., Santos, J.C.S.: Large language models for software engineering: A reproducibility crisis (2025), https://arxiv.org/abs/2512.00651

33. Siegel, Z.S., Kapoor, S., Nadgir, N., Stroebl, B., Narayanan, A.: CORE-Bench: Fostering the Credibility of Published Research Through a Computational Reproducibility Agent Benchmark. Trans. Mach. Learn. Res. (2024), https://openreview.net/forum?id=BsMMc4MEGS
34. Starace, G., Jaffe, O., Sherburn, D., Aung, J., Chan, J.S., Maksin, L., Dias, R., Mays, E., Kinsella, B., Thompson, W., Heidecke, J., Glaese, A., Patwardhan, T.: PaperBench: Evaluating AI's Ability to Replicate AI Research (2025), https://arxiv.org/abs/2504.01848
35. Stodden, V., McNutt, M., Bailey, D.H., Deelman, E., Gil, Y., Hanson, B., Heroux, M.A., Ioannidis, J.P., Taufer, M.: Enhancing reproducibility for computational methods. Science **354**(6317), 1240–1241 (2016). https://doi.org/10.1126/science.aah6168
36. Stodden, V., Miguez, S.: Best Practices for Computational Science: Software Infrastructure and Environments for Reproducible and Extensible Research. J. Open Res. Soft. **2**(1) (2014), https://openresearchsoftware.metajnl.com/articles/10.5334/jors.ay, collection: Working towards Sustainable Software for Science: Practice and Experiences
37. Timperley, C.S., Herckis, L., Le Goues, C., Hilton, M.: Understanding and improving artifact sharing in software engineering research. Springer Empir. Softw. Eng. **26**(4), 67 (2021). https://doi.org/10.1007/S10664-021-09973-5
38. Wilkinson, M.D., Dumontier, M., Aalbersberg, I.J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.W., da Silva Santos, L.B., Bourne, P.E., et al.: The FAIR Guiding Principles for scientific data management and stewardship. Sci. Dat. **3**(1), 1–9 (2016). https://doi.org/10.1038/sdata.2016.18
39. Wilson, G., Bryan, J., Cranston, K., Kitzes, J., Nederbragt, L., Teal, T.K.: Good Enough Practices in Scientific Computing. PLOS Comput. Biol. **13**(6), 1–20 (2017). https://doi.org/10.1371/journal.pcbi.1005510
40. Winter, S., Timperley, C.S., Hermann, B., Cito, J., Bell, J., Hilton, M., Beyer, D.: A retrospective study of one decade of artifact evaluations. In: Proc. 30th Eur. Softw. Eng. Conf./Found. Softw. Eng. (ESEC/FSE). p. 145–156 (2022). https://doi.org/10.1145/3540250.3549172
41. Wu, Z., Zhao, Y., Chen, Z., Wang, Z., Wang, H.: Agent-based software artifact evaluation (2026), https://arxiv.org/abs/2602.02235
42. Xiang, Y., Yan, H., Ouyang, S., Gui, L., He, Y.: SciReplicate-Bench: Benchmarking LLMs in Agent-driven Algorithmic Reproduction from Research Papers. Proc. 2nd Conf. Lang. Model. (COLM) (Jul 2025), https://openreview.net/forum?id=8LoPjpvWde#discussion
43. Xu, F.F., Alon, U., Neubig, G., Hellendoorn, V.J.: A Systematic Evaluation of Large Language Models of Code. In: Proc. 6th ACM SIGPLAN Int. Symp. Mach. Program. (MAPS). p. 1–10 (2022). https://doi.org/10.1145/3520312.3534862
44. Yi, J., Xie, Y., Zhu, B., Kiciman, E., Sun, G., Xie, X., Wu, F.: Benchmarking and defending against indirect prompt injection attacks on large language models. In: Proc. 31st ACM SIGKDD Conf. Knowl. Discov Data Min. (KDD) V.1. pp. 1809–1820 (2025). https://doi.org/10.1145/3690624.3709179