
UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

# UNIVERSITY OF MODENA AND REGGIO EMILIA

"Enzo Ferrari" Department of Engineering

Master's Degree in Artificial Intelligence Engineering (LM-32)

## Paper-snitch: A Practical Tool for Evidence-Based Reproducibility Assessment

**Supervisor:**
Prof. Federico Bolelli

**Candidate:**
Davide Santoli
Student ID 195193

---

**ACADEMIC YEAR 2024/2025**

# *Abstract*

**Paper-snitch: A Practical Tool for Evidence-Based Reproducibility Assessment**

Reproducibility policies aim to make medical-imaging research "checkable" at review time, yet many submissions still ship artifacts that are missing, incomplete, or not practically verifiable under peer-review constraints. In a longitudinal analysis of 3,722 MICCAI papers, the fraction linking code increases from 51.8% to 72.5%, but ∼13% of linked repositories are inaccessible or empty, and many others lack the concrete information needed to audit claims without executing untrusted code.

This thesis introduces PAPER-SNITCH, a reviewer-facing decision support for evidence-based reproducibility screening. PAPER-SNITCH parses conference metadata and PDFs, resolves and sanity-checks linked repositories, and applies a policy-aware checklist aligned with MICCAI-style guidance. Rather than attempting full reproduction, it performs bounded, inspectable checks (e.g., artifact presence, documentation completeness, environment specification, and claim-to-command traceability) and produces an auditable report that links each criterion outcome to concrete evidence excerpts and repository artifacts. Criterion outcomes are aggregated deterministically into interpretable component scores and a global verifiability summary.

We evaluate PAPER-SNITCH against human assessment on a sample of MICCAI papers and analyze practical deployment considerations, including per-paper cost, caching, and observability. Finally, we discuss key limitations—most notably incomplete recall under bounded retrieval, susceptibility to strategically written text, and the non-determinism and drift of LLM-backed components—and outline directions for robustness hardening and reviewer-centered validation.

**Keywords:** Reproducibility screening, evidence-grounded auditing, artifact verification, peer-review decision support, policy-aware checklists, provenance, large language models.

# Contents

# List of Figures

# List of Code

# 1. Introduction

## 1.1 Context and Motivation

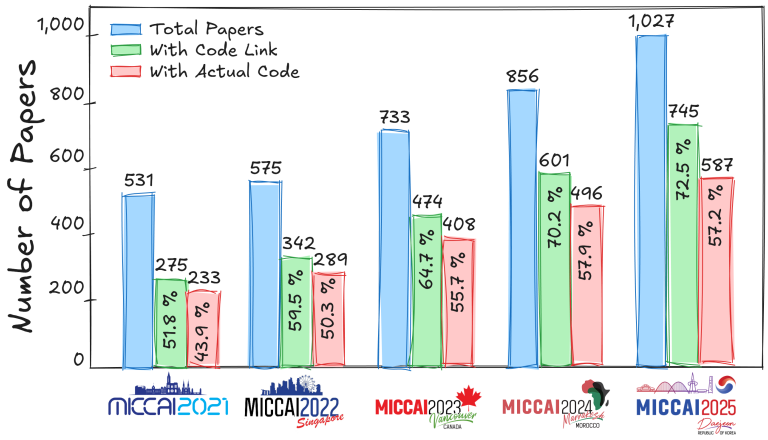
The field of medical image analysis has witnessed unprecedented advances, largely driven by novel architectures and increasingly sophisticated training methodologies. However, this rapid progress presents a less glamorous but increasingly urgent problem for the scientific community: how to keep research checkable when both the volume and complexity of scientific production are accelerating at once. Although significant emphasis is regularly placed on achieving state-of-the-art predictive performance, establishing the robust verifiability of these underlying methodologies remains a fundamental, yet frequently under-addressed, necessity.

This challenge is not only technical but also shaped by structural aspects of the current academic environment. The strong emphasis on rapid publication and visibility can, in practice, incentivize speed over thorough validation, making it more difficult to consistently prioritize reproducibility. As a result, methodological transparency and careful verification are sometimes constrained by time and resource pressures, with downstream effects on the overall reliability of published work.

The current wave of medical imaging research is further shaped by two concurrent trends: a rapid increase in pipeline complexity and the widespread adoption of large language models (LLMs) and automated code-generation tools. Modern research workflows involve numerous interdependent components, including data curation, preprocessing, hyperparameter tuning, evaluation protocols, and hardware considerations. At the same time, LLMs have significantly reduced the friction associated with writing manuscripts and producing supporting code artifacts.

Technological advances that could help reproducibility are instead frequently repurposed for throughput. Public code and data platforms (e.g., GitHub, OpenNeuro/BIDS), containerization (Docker, Singularity/Apptainer), workflow managers (Snakemake, Nextflow), and experiment-tracking tools (DVC, MLflow, W&B) can materially improve verifiability when used to archive data, fix computational environments, and record experiments. However, these tools are too often used to produce artifacts that are superficially complete but insufficiently validated, limiting their effectiveness in supporting reproducible research.

Due to this combination of increasing complexity, accelerated production, and partial adoption of reproducibility practices, the academic review process is under significant strain. Reviewers are tasked with evaluating technical novelty, clinical relevance, experimental rigor, and ethical considerations within strict time constraints, while reproducibility requirements introduce an additional demanding dimension. This thesis situates its contribution precisely at the intersection of growing methodological complexity, systemic pressures, and the need for effective verification mechanisms.




(a) Code availability in MICCAI proceedings


(b) Findings

Figure 1.1: Code availability trends in MICCAI openly available proceedings (2021–2025). (a) Longitudinal analysis across 3 722 papers: for each year, the plot reports the total number of accepted papers, the number of papers that provide a GitHub repository link, and the number of papers whose linked repository is verified to be accessible and to contain implementation code. While the fraction of papers providing a code link increases over time, a persistent gap remains between linked and verifiably accessible repositories, reflecting links that resolve to inaccessible (e.g., 404) or effectively empty repositories. (b) Example of the PAPER-SNITCH web interface showing an evidence-highlighted paper view with criterion-linked excerpts (see Section 1.3).

## 1.2 The Role and Limitations of Reproducibility Policies

To address these challenges, major scientific venues have established formal reproducibility policies. For example, the MICCAI society provides explicit guidance detailing the artifacts required to

support reproducible research, including codebases, trained models, data documentation, and evaluation protocols. Similarly, other computer vision and pattern recognition communities increasingly employ artifact evaluation tracks and badge systems to recognize verifiable releases and incentivize compliance. The fundamental intent of these initiatives is to shift the scientific paradigm from assumed trust to verifiable evidence.

Despite these clear directives, the practical bottleneck lies within the review workflow. A significant proportion of submissions either fail to provide code entirely or promise a post-acceptance release that frequently fails to materialize. Even when repository links are provided, they are often empty, private, or lack critical components beyond a rudimentary placeholder. Furthermore, even with accessible artifacts, reviewers operating under strict time constraints cannot realistically execute arbitrary code, resolve complex dependencies, or reproduce computationally intensive GPU training regimens. Consequently, accurately assessing policy compliance becomes exceedingly difficult, blurring the boundary between artifacts that are not verifiable within review constraints and those that are genuinely insufficient.

Crucially, a substantial portion of this verification burden is entirely mechanical and therefore amenable to automation. Many of the checks required during the review phase are objective verifications rather than nuanced scientific judgments. For example, determining whether a repository exists, contains executable code, includes an explicit license, or provides specific environment configurations are automatable tasks. Similarly, verifying the presence of pretrained weights, data access instructions, and the traceability of paper claims to concrete execution commands are bounded checks.

## 1.3 Proposed Solution: PAPER-SNITCH

This observation motivates the core proposition of this thesis: modern artificial intelligence tooling can be leveraged to perform these bounded checks, acting as a decision-support tool to reduce the burden of mechanical verification while supporting human judgment in reproducibility assessments. The objective is not to create an automated gatekeeper to accept or reject papers, nor to execute untrusted code during peer review. Instead, the goal is to produce a transparent and evidence-based report that assists reviewers in rapidly identifying which reproducibility elements are present, which

are missing, and which require human interpretation.

To this end, this thesis introduces PAPER-SNITCH, a reviewer-facing decision support designed for the review-time screening of reproducibility signals in medical imaging submissions. By integrating lightweight repository inspection, structured text analysis of the manuscript, and policy-aware checklists aligned with MICCAI guidelines, the system generates a comprehensive verifiability score. This score is decomposed into interpretable sub-scores, alongside criterion-linked findings and excerpts that reviewers can directly inspect.

The workflow is designed to be highly scalable, allowing it to seamlessly integrate into a conference review pipeline as an auxiliary report. It is explicitly tailored to the time and safety constraints of double-blind peer review. The system never executes untrusted code or attempts heavy reproduction; it strictly focuses on verifiable checks based on the provided artifacts. Although this research focuses on the MICCAI conference due to the complex nature of medical imaging reproducibility, the underlying architecture is modular and adaptable to other computer vision venues.

## 1.4 Research Questions and Hypotheses

Building on the need for scalable, review-time verification of reproducibility artifacts, this thesis investigates how policy requirements can be translated into an automated, evidence-based screening workflow. Rather than attempting full reproduction, the focus is on bounded auditable checks that align with the practical constraints of peer review. The research is guided by the following questions:

- **RQ1 (Operationalizing Policy into Verifiable Criteria):** How can reproducibility guidelines defined by venues such as MICCAI be systematically translated into a structured set of verifiable review-time criteria?
  - *Hypothesis 1:* Decomposing high-level policy requirements into granular checklist-based criteria enables the construction of a deterministic and auditable scoring framework that reflects artifact completeness and traceability.
- **RQ2 (Automated Evidence Extraction for Bounded Verification):** To what extent is it possible to automatically analyze the manuscript text and linked repositories to extract the

evidence needed for reproducibility screening without running the code?

  - *Hypothesis 2:* A hybrid pipeline that combines structured text parsing, repository inspection, and targeted retrieval mechanisms can reliably recover sufficient evidence to support bounded reproducibility checks under peer-review constraints.

- **RQ3 (Agreement with Human Assessment):** How closely do decisions at the criterion-level and the aggregate scores produced by PAPER-SNITCH align with the human expert evaluations of the reproducibility artifacts?

  - *Hypothesis 3:* The system will achieve substantial agreement with human reviewers on objective, evidence-based criteria, supporting its validity as a reproducibility screening tool.

- **RQ4 (Utility in the Review Workflow):** What is the practical value of providing reviewers with decomposed verifiability scores and evidence based on criteria during peer review?

  - *Hypothesis 4:* Presenting structured scores alongside traceable evidence reduces the burden of mechanical verification and improves the efficiency and clarity of reproducibility assessment without replacing human judgment.

# 2. Background and Literature Review

This chapter establishes the conceptual foundations of the thesis and directly supports the research questions introduced in Chapter 1. First, it clarifies terminology and frames reproducibility as an operational property of a complete research artifact rather than a purely rhetorical claim. It then reviews how conferences translate this objective into checklists, badges, and artifact-review procedures, highlighting that different communities operationalize "reproducibility" through different mechanisms (e.g., mandatory checklists, optional artifact badges, or venue-specific guidance). The chapter subsequently discusses why medical image analysis poses unusually strong reproducibility constraints and finally introduces the technical rationale for bounded LLM-assisted verification under peer-review constraints. This background motivates the system design choices detailed in Chapter 3.

## 2.1 From Reproducibility Claim to Verifiable Artifact

In computational research, reproducibility is increasingly understood as a property that must be demonstrated through artifacts. A paper alone can communicate ideas and empirical trends, but it is usually insufficient to independently re-obtain reported results. The practical unit of scientific contribution is therefore better represented as a computational compendium: manuscript, code, data access instructions, environment specification, and execution logic.

This view has been repeatedly emphasized in the reproducible computational science literature, where the artifact package—not only the narrative—is treated as the primary substrate for verification and reuse [36, 8].

### 2.1.1 Terminology and Scope

The reproducibility landscape contains partially overlapping terms that are sometimes used inconsistently across fields.

In this thesis, we adopt a pragmatic distinction aligned with how conferences operationalize requirements in checklists: *computational reproducibility* refers to re-obtaining the reported results

given the authors' code, claimed configuration, and the same (or functionally equivalent) data access; *replicability* refers to obtaining consistent conclusions under an independent implementation and potentially different but comparable data; and *robustness* refers to stability of findings under reasonable variations (e.g., alternative preprocessing choices, different random seeds, or different sites).

These distinctions matter because review-time policies rarely require full replication; instead, they ask authors to provide enough evidence that computational reproducibility is plausible and that methodological reporting is sufficiently detailed to enable downstream replication studies.

This shift is particularly relevant for contemporary AI workflows, where results depend on many tightly coupled implementation decisions. Even minor details–for example, preprocessing errors, random seeds, evaluation scripts, or checkpoint selection criteria–can materially alter the results [8]. As a consequence, reproducibility is not only about making code public; it is about exposing enough operational detail to make claims auditable.

However, at review time, full reproduction is rarely feasible. Reviewers operate under strict deadlines, limited computational budgets, and security constraints that discourage the execution of arbitrary external code. For this reason, a pragmatic distinction emerges between *full reproduction* and *bounded verifiability*. Full reproduction asks whether the complete pipeline can be rerun end-to-end; bounded verifiability asks whether the submission provides the minimal evidence expected by policy to make that reproduction plausible. This thesis explicitly targets the second objective and treats it as a well-defined, policy-scoped screening task.

## 2.2 Institutional Responses: Checklists, Badges, and Policy Formalization

Major conferences and scientific societies have progressively formalized reproducibility requirements to reduce ambiguity in the review process. Two trends are especially important.

First, venues introduced structured checklists to standardize what authors must report (e.g., environment setup, data splits, training details, and evaluation protocols). Notable examples include the NeurIPS reproducibility program and its checklist, which formalize reporting expectations and document lessons learned from large-scale deployment [33, 25].

Second, communities adopted artifact evaluation and badging mechanisms to explicitly recognize releases that satisfy predefined criteria such as availability, functionality, and reusability. The ACM artifact review and badge taxonomy is a widely referenced example that separates the existence of an artifact from its evaluated functionality and reusability, thereby reducing ambiguity around what a "released" repository actually means [1].

These mechanisms serve both normative and operational roles. Normatively, they communicate that reproducibility is an expected part of scientific quality. Operationally, they provide reviewers with a shared rubric that reduces ad-hoc judgment. However, in practice, compliance quality remains heterogeneous: some submissions provide complete and testable artifacts, while others provide partial repositories, inaccessible resources, or underspecified instructions.

A useful way to interpret this heterogeneity is to distinguish between *self-reported compliance* and *verified artifacts*. Checklists are primarily self-reported and therefore improve coverage and standardization of what is described, but do not guarantee that artifacts are reachable, complete, or runnable. Badging and artifact evaluation introduce a verification step, but they are costlier and typically occur outside the main review loop or after acceptance. In this thesis, the goal is not to replace either mechanism, but to support checklist-oriented review-time screening by producing evidence-based signals that help reviewers focus their limited time on items that are missing, inconsistent, or ambiguous.

An additional practical tension is that the reproducibility expectations in review-time interact with double-blind constraints. Some venues discourage or regulate external links during submission to protect anonymity, which can limit what reviewers can verify in practice and push artifact checks later in the workflow [4]. This tension reinforces the need to clearly separate what can be verified during peer review from what can only be validated post-acceptance.

Within medical imaging and pattern recognition, this institutional evolution is visible in initiatives such as reproducible-research tracks and venue-specific guidance [10]. For this thesis, MICCAI is the key reference context because its guidance explicitly enumerates artifact- and reporting-level expectations that are tightly coupled to medical imaging constraints (restricted data access, governance, and heterogeneous acquisition) [22].

Importantly, the present thesis does not attempt to generalize the emphasis of policy in research areas or conferences. It uses MICCAI as a concrete reference point because its publicly stated

guidelines provide an explicit, itemized policy surface that can be operationalized into review-time criteria [22].

Other communities have adopted different, but equally formal mechanisms. For example, the major general ML venues have used mandatory reproducibility checklists that are included in the submission PDF and visible to reviewers [33, 25]. The practical difference is therefore not that one research area inherently "cares more," but that venues differ in *where* they place emphasis and *how* they enforce it: MICCAI's guidance foregrounds dataset provenance and conditional access constraints alongside artifact expectations, whereas checklist-driven venues foreground standardized reporting and structured self-assessment at submission time.

In all of these settings, peer review remains constrained by double-blind procedures, limited time, and a high volume of submissions. This shared tension motivates automation that assists verification without replacing reviewer authority.

The next section summarizes the MICCAI guidance and clarifies how its qualitative requirements can be mapped into concrete review-time verifiability criteria.

## 2.3 MICCAI Guidelines for Reproducible Research

The MICCAI reproducible research guidance explicitly frames reproducibility as a combination of artifact availability and methodological transparency, encouraging authors to strengthen (i) data transparency, (ii) open implementations, and (iii) appropriate evaluation design and reporting [22]. Authors are strongly encouraged to provide sufficient technical evidence for reviewers and Area Chairs to assess not only novelty but also the practical verifiability of the contribution.

The guidelines can be organized into four complementary requirement blocks.

**1) Model and algorithm description.** For each method, authors are asked to report: (i) the mathematical setting and algorithmic/model formulation, (ii) the assumptions under which the method is expected to work, and (iii) the software framework with exact version information.

Here, *mathematical setting* refers to the precise formalization of the task and variables (e.g., the input domain and modality, the output space, the learning objective, and the notation used for data, labels, and model parameters). *Algorithmic/model formulation* specifies what is actu-

ally implemented (e.g., architecture components, loss functions, optimization procedure, and any postprocessing), as opposed to a high-level conceptual description. *Assumptions* make explicit the conditions under which the proposed method and evaluation are valid (e.g., i.i.d. sampling vs. site-dependent shifts, availability and quality of annotations, preprocessing requirements, or constraints tied to a specific imaging protocol). Finally, *framework and version information* should identify the exact software stack that materially affects results (e.g., PyTorch/TensorFlow versions, CUDA/cuDNN, and key libraries), enabling reviewers and future users to reproduce the environment rather than approximating it.

This block is essential because many apparent performance differences arise from the implementation context rather than the conceptual novelty alone. In reproducibility terms, framework and version declarations reduce ambiguity and make environment reconstruction feasible.

**2) Dataset transparency and provenance.** For datasets, MICCAI requests descriptive statistics (e.g., number of examples), cohort characterization, references for existing datasets, and detailed acquisition documentation for newly collected data (including setup, devices, imaging parameters, annotator instructions, and quality-control procedures). The guideline also asks for public download links when available and explicit statements on ethics approval requirements.

These terms have specific operational meaning. *Descriptive statistics* summarize the dataset at a minimum granularity sufficient to contextualize performance claims (e.g., sample counts by split, class prevalence, missing data rates, and image resolution ranges). *Cohort characterization* describes how subjects/cases were selected and what population the data represent (e.g., inclusion/exclusion criteria, demographics when appropriate, pathology distribution, sites/centers, and time ranges), which is essential to understand generalization. *Provenance* captures the dataset's origin and processing history (e.g., where the data came from, how it was curated, how labels were obtained, what transformations were applied, and what governance constraints apply). For *acquisition documentation* in medical imaging, "setup" and "devices" concretely include scanner/-manufacturer, field strength, sequence/protocol identifiers, reconstruction parameters, and calibration steps; *imaging parameters* include resolution, slice thickness, echo/repetition times, and other modality-specific settings that can change the signal distribution. *Annotator instructions* specify the labeling protocol (what was labeled, according to which rules, by whom, and with what tooling), and

*quality-control procedures* document how corrupted, low-quality, or out-of-distribution cases were detected and handled. *Ethics approval requirements* clarify whether institutional review/consent applies, and what restrictions it imposes on sharing, re-use, or external access.

This block is particularly critical in medical imaging, where access constraints, cohort composition, and acquisition protocols strongly affect external validity. Even when data cannot be openly released, detailed provenance and governance information remain necessary to interpret and compare results.

**3) Code and artifact completeness.** For code artifacts (already available or planned upon acceptance), MICCAI asks for dependency specification, training code, evaluation code, pretrained models, dataset access information, and a README containing a result table plus precise commands to reproduce those results.

In practice, *dependency specification* means a machine-actionable declaration of the runtime environment (e.g., pinned package versions via `requirements.txt`/`environment.yml`, containers, and any non-Python system dependencies) so that the code can be installed consistently. *Training code* should cover end-to-end model fitting (including configuration, random seeds, checkpointing, and logging), while *evaluation code* should cover inference and metric computation in a way that maps clearly to what is reported in the paper. *Pretrained models* (when shareable) are the actual weight files or model checkpoints needed to validate reported results without re-training. *Dataset access information* goes beyond a URL: it includes credentials or request procedures for restricted datasets, expected directory structure, preprocessing scripts, and any required licenses. Finally, a *README with a result table and precise commands* should provide a traceable bridge between claims and execution (i.e., the paper's main numbers correspond to specific commands/configurations that reproduce them under stated conditions).

This requirement operationalizes the idea of a computational compendium: the repository should be executable as a documented workflow, not merely a placeholder. From a review perspective, this is the minimum structure needed to assess whether claims are practically checkable.

**4) Experimental reporting quality.** For experimental evidence, the guideline requires reporting of hyperparameter search ranges and selection strategy, sensitivity analyses, number of runs, base-

line implementation/tuning details, train/validation/test split definitions, metric definitions, central tendency and variability, statistical significance analysis, runtime or energy estimates, memory footprint, failure-case analysis, computing infrastructure, and discussion of clinical significance.

For clarity, these items specify the minimum information needed to interpret empirical results as *evidence* rather than as a single-point claim. *Hyperparameter search ranges and selection strategy* describe which configurations were explored (e.g., learning rates, regularization, augmentation strengths) and how the final model was chosen (e.g., grid/random search, Bayesian optimization, early stopping, and validation-based selection). *Sensitivity analyses* assess whether conclusions are stable when key choices change (e.g., different preprocessing variants, different thresholds, or different training schedules). *Number of runs* and *variability* report robustness to randomness (e.g., multiple seeds, confidence intervals, standard deviations), while *central tendency* specifies how performance is summarized (e.g., mean vs. median). *Baseline implementation/tuning details* document whether comparisons are fair (e.g., baselines re-implemented vs. taken from prior code, tuned with the same budget, and evaluated under the same splits). *Split definitions* should state the unit of splitting (e.g., patient-level vs. slice-level), leakage-avoidance rules, and any cross-site or temporal separation. *Statistical significance analysis* clarifies whether observed differences exceed expected noise (e.g., paired tests, bootstrap intervals, or non-parametric tests, depending on the setting). *Runtime/energy*, *memory footprint*, and *computing infrastructure* make the compute envelope explicit (e.g., GPU model, VRAM, CPU/RAM, software stack), which affects feasibility and reproducibility. *Failure-case analysis* requires describing where and how the method fails, and *clinical significance* contextualizes whether measured improvements are meaningful for downstream medical decision-making rather than merely statistically detectable.

This final block addresses a recurrent issue in AI papers: reproducibility can fail even when code is released if evaluation design and reporting are underspecified. MICCAI therefore emphasizes that reproducibility is both *artifact availability* and *methodological transparency*.

In general, these recommendations provide a concrete policy framework that aligns directly with the goals of this thesis. In Chapter 3 and Chapter 4, they are translated into operational criteria for bounded review-time checks, allowing automated support of the mechanical verification process while preserving human scientific judgment.

From an operational perspective, the four blocks also suggest a natural decomposition for

automated screening. Items about model/algorithm description and experimental reporting are largely *manuscript-based* (they can be verified by extracting and checking whether required details are stated). In contrast, code/artifact completeness and parts of dataset transparency are often *artifact-grounded* (they require checking linked repositories, supplementary materials, and data-access statements). This split motivates the separation used later in this thesis between paper-side criteria and external artifact-side checks, as well as the need to keep output traceable to either PDF evidence or repository evidence.

## 2.4 Why Medical Image Analysis is Reproducibility-Critical

Medical image analysis presents additional constraints compared to general computer vision. Data are often sensitive, access-controlled, and subject to legal or institutional restrictions, which can prevent direct release of raw images. At the same time, acquisition heterogeneity (scanner types, protocols, sites, and populations) introduces distributional variability that directly affects model behavior.

In many cases, reproducibility must therefore be interpreted as *conditional* rather than universal: authors may be unable to publish raw data, but they can still provide precise access procedures (e.g., data-use agreements), cohort and acquisition descriptions, and explicit preprocessing and split logic. In other words, when open data are infeasible, the reproducibility target shifts from downloading the dataset to being able to audit what was done and, when permitted, to re-run the analysis under comparable conditions.

Methodological pipelines are also highly composite. Typical workflows combine curation, quality control, preprocessing, resampling, region selection, model training, postprocessing, and metric computation. Each stage introduces degrees of freedom that may be weakly documented in short conference papers. When these decisions are not explicitly stated, even a public repository may fail to support meaningful verification.

Related work in the medical imaging community has highlighted that incomplete reporting and heterogeneous evaluation design can make results difficult to interpret and reproduce, motivating structured reporting standards and checklists (e.g., in challenge settings) as a corrective mechanism [20].

For this reason, reproducibility assessment in medical imaging must go beyond binary checks such as "code available / not available." Useful assessment requires a criterion-level inspection of whether the artifact package documents key choices about data provenance, split strategy, preprocessing logic, and evaluation definitions. This requirement directly informs the decomposition of the criteria used later in the proposed scoring framework.

## 2.5 Technical Foundations for Automated Reproducibility Screening

### 2.5.1 Bounded Information Extraction

A central observation of this thesis is that many review-time reproducibility checks are fundamentally extraction and matching tasks. Examples include verifying whether repository links are reachable, whether an environment file is declared, whether training or inference scripts are identifiable, and whether manuscript claims are supported by concrete artifact references.

These tasks are objective enough to benefit from automation, provided that the system is constrained and auditable. In this perspective, AI is not used to produce unrestricted scientific judgments; it is used to map evidence fragments to predefined criteria. This design directly aligns with the thesis scope and research questions: automation targets the mechanical burden of verification while leaving scientific interpretation and final decisions to humans.

### 2.5.2 Design Constraints Under Peer Review

Automated screening is only useful if it respects the same constraints that make manual verification difficult. Three constraints are particularly important.

First, *safety*: review-time workflows cannot assume it is acceptable to execute untrusted code, download arbitrary binaries, or run opaque install scripts. Second, *privacy and governance*: submissions may contain unpublished material, and artifacts may reference sensitive datasets or restricted-access resources. Third, *budget and latency*: a screening tool must scale to hundreds or thousands of submissions, and therefore must bound its computational footprint and its dependence on external

services.

These constraints motivate a conservative architecture: static inspection of artifacts, retrieval-bounded context for LLM calls, and deterministic aggregation rules. The remainder of this chapter introduces the main technical ingredient used to satisfy these constraints: retrieval-augmented, criterion-first evaluation.

### 2.5.3 Retrieval-Augmented Generation in this Context

Because manuscript text and repository contents can be long and structurally noisy, direct end-to-end prompting is brittle. Retrieval-Augmented Generation (RAG) offers a more robust pattern: first retrieve the most relevant passages for a criterion, and then ask the model to evaluate only on that bounded evidence [12, 15].

Dense retrieval is particularly useful when lexical overlap is weak (for example, when implementation details are paraphrased). By indexing paper and repository chunks in an embedding space, the system can surface semantically related context even when exact keywords differ. This improves recall of relevant evidence and reduces prompt overload.

At the same time, retrieval introduces design trade-offs: chunk size, overlap strategy, and top-$k$ selection directly influence evidence quality. Too little context may omit decisive details; too much context may reintroduce noise and hallucination risk. This is particularly relevant for long documents, where models can underweight crucial middle spans when context windows are large [18]. Therefore, the retrieval configuration is not an implementation detail, but part of the methodological design space addressed in this thesis.

### 2.5.4 From LLM Outputs to Deterministic Scores

To preserve interpretability, the system separates two layers: evidence extraction and score computation. The LLM layer proposes criterion-level findings with traceable support, while the final score is computed deterministically through explicit arithmetic rules. This separation reduces subjectivity in the aggregation and aligns with the research objective of policy operationalization.

In other words, the model supports *the evidence that is present*; the scoring logic defines *how that evidence is weighted*. This design choice is essential for auditability and for adapting the

framework across submission types without changing the underlying evaluation philosophy.

## 2.6 LLMs in Peer Review: Opportunity, Risk, and Human Oversight

The growing interest in generative AI for peer review reflects a practical need: submission volume is increasing faster than reviewer capacity, and many reviewers report severe time constraints. In this setting, LLMs can provide useful support for triage, consistency checks, and evidence organization, especially when tasks are repetitive and criteria are explicit [9, 24].

Recent progress in long-text modeling further increases the attractiveness of these tools, because models can process lengthy submissions and supplementary material in a single pass. In the best case, this enables semi-automated support for structured checks such as verifying that a checklist item is addressed, detecting missing declarations (e.g., data availability, ethics statements, or environment specification), and flagging inconsistencies between claims, tables, and artifact pointers. At the same time, long contexts do not automatically translate into reliable reasoning: models can still under-attend to crucial passages and can exhibit context-position effects, which makes careful evidence selection and traceability important even when token budgets are large [18].

However, these benefits are coupled with non-trivial risks regarding comprehension, calibration, and bias. Although proficient in linguistic and summarization tasks, current LLMs may lack deep scientific understanding in specialized domains and can struggle with nuanced statistical reporting or cautious academic phrasing (e.g., distinguishing exploratory from confirmatory claims, or recognizing when limitations materially weaken a conclusion). Failure modes include hallucinations (fabricating evidence, numbers, or references), overconfident misclassification, and biased language patterns. Empirically, LLM-generated reviews have been found to skew toward overly positive phrasing and to assign higher acceptance recommendations than human experts, raising concerns about bias amplification if outputs are used without oversight [9, 24].

Integration of LLMs also introduces privacy and security vulnerabilities. Peer review regularly involves potentially sensitive and unpublished intellectual property, and routing this material through third-party cloud APIs can create exposure risks depending on provider policies, institutional rules, and data-handling practices. Moreover, systems that ingest external links, PDFs, and repositories can

become targets for adversarial content. In particular, *indirect prompt injection* can be implemented by embedding hidden instructions in a submission (e.g., white text in a PDF) or in an artifact (e.g., a repository README), attempting to manipulate the model into ignoring flaws, fabricating citations, or inflating compliance outcomes. Such attacks are especially relevant for reproducibility screening because the system is explicitly tasked with reading untrusted artifacts.

For these reasons, a safe design must enforce bounded prompts, criterion-specific outputs, explicit refusal to infer missing evidence, and strict human-in-the-loop validation. In practice, these safeguards translate into architectural constraints: retrieval-bounded context, structured output, deterministic aggregation, and provenance tracking for every evidence fragment used in scoring. These layers are intended to keep outputs anchored to auditable evidence, reduce the attack surface for generative manipulation, and make failures diagnosable rather than opaque.

The approach adopted in this thesis follows that principle. The automated system is framed as a decision-support system, not an autonomous reviewer: it organizes and checks the verifiable surface of submissions under review-time constraints, while the final scientific judgment and accountability remain with human reviewers and chairs [9, 24].

## 2.7 Synthesis and Link to Methodology

This chapter provided the background necessary to interpret the methodological choices in the next chapter. Reproducibility has been reframed as a verifiable artifact property; terminology and scope were clarified to motivate bounded screening; institutional policies were positioned as operational constraints; medical imaging challenges motivated the granularity of the criterion; and RAG-based bounded LLM assistance was introduced as a practical mechanism for scalable screening.

Building on this foundation, Chapter 3 details how these principles are instantiated in the architecture of Paper-snitch, including data ingestion, retrieval design, criterion evaluation, and deterministic score aggregation.

# 3. Methodology and System Design

This chapter describes the methodology used to operationalize the conference reproducibility guidance into a review-time screening pipeline based on evidence and the corresponding system design choices behind PAPER-SNITCH. Building on the policy and conceptual framing introduced in Chapter 2, the goal here is not to "reproduce" results end-to-end, but to perform *bounded verifiability checks*: objective, inspectable validations that can be executed within peer-review constraints.

The methodology is shaped by three non-negotiable constraints. First, *evidence grounding*: all model-assisted judgments must be anchored to explicit artifacts (paper excerpts, repository files, URLs) to reduce hallucinations, provide explainability, and keep generation focused on the relevant text rather than on surrounding noise. Second, *safety*: the primary inputs (PDFs, repositories, supplementary files, and external URLs) are untrusted and potentially adversarial, therefore, the system must avoid risky actions such as executing third-party code. Third, *scalability and auditability*: the pipeline must support conference-scale usage (potentially thousands of submissions triggered with minimal user effort) while keeping outputs stable, inspectable, and reproducible through durable storage of intermediate artifacts.

## 3.1 Design Goals and Threat Model

The design of PAPER-SNITCH is guided by four goals that repeatedly inform architectural decisions.

**G1: Boundedness and feasibility.** The system must prioritize checks that are achievable during peer review: availability of artifacts, clarity of documentation, presence of environment declarations, traceability of claims to scripts and commands, and consistency between paper statements and repository contents. It explicitly avoids GPU-intensive reproduction and avoids any dependence on private infrastructure.

**G2: Safety against untrusted artifacts.** Repositories and PDFs may contain malicious payloads or instructions aimed at manipulating the analysis. Consequently, PAPER-SNITCH treats all ingested

text as untrusted and follows a strict *non-execution* principle: it never executes third-party code and relies on static inspection and bounded retrieval.

**G3: Auditability and provenance.** Every score and intermediate decision must be traceable to concrete artifacts (paper excerpts, file paths, URLs, parsed metadata). Provenance is essential both to reduce hallucination risk and to enable human verification.

**G4: Modularity and partial completion.** The pipeline is not an atomic operation, but a composition of heterogeneous steps. The methodology therefore favors a node-based workflow that supports conditional routing, fault isolation, caching, and parallelization.

## 3.2 Workflow Architecture as a Versioned State Graph

### 3.2.1 Node-based Pipeline with Conditional Branches

PAPER-SNITCH implements the analysis as a graph of nodes, where each node performs a focused transformation (e.g., PDF parsing, paper-type classification, embedding construction, criterion retrieval, criterion evaluation, repository inspection). This choice is motivated by the structure of the task: reproducibility screening requires heterogeneous operations, some deterministic (parsing, indexing, similarity search, arithmetic aggregation) and some probabilistic (LLM-based classification and extraction). Treating each operation as an explicit node yields several practical benefits.

**Partial completion.** In real-world inputs, failures are common (broken links, unsupported PDFs, rate limits, missing files). A node-based architecture allows the pipeline to return partial results even when some nodes fail.

**Fault isolation and resource savings.** When a node fails, the previously computed artifacts remain valid and reusable. This prevents expensive recomputation (e.g., embeddings or repository digests) and isolates failures to a localized component.

**Modularity and multiple workflows.** Different submission types require different checks. Conditional routing supports specialized workflows (e.g., dataset-focused branches) without forcing every paper through every step.

**Auditability.** The graph structure provides an explicit "why" chain: scores can be traced back through node outputs to the underlying evidence. This is a prerequisite for reviewer-facing decision support.

**Parallelization.** Independent nodes (e.g., criterion evaluations) can execute concurrently, improving throughput when analyzing multiple criteria and/or multiple models.

In the implementation, this abstraction is expressed as a versioned state graph (LangGraph) where the nodes operate in a shared persistent state and produce structured artifacts suitable for storage and inspection [13, 14].

### 3.2.2 Database-backed Workflow Engine and Execution Semantics

To make the graph operational in a production web service, PAPER-SNITCH implements a database-backed workflow engine. The workflow engine treats the analysis as a directed acyclic graph (DAG) whose state is persisted in a relational database, enabling reliability, restartability, and auditability. Conceptually, the engine is organized around the following entities:

- **Workflow definition:** a reusable template that describes nodes, edges, retry limits, and handlers.
- **Workflow run:** a concrete instance of the workflow applied to a specific paper, with run-level input parameters (e.g., model choice, policy version, forced reprocessing flags).
- **Workflow node:** an executable unit within a run, with explicit dependencies, status, timestamps, and error information.
- **Node artifacts and logs:** durable records of intermediate outputs and execution traces (e.g., extracted TEI, retrieved evidence chunks, JSON-structured LLM outputs, token usage).

- **LangGraph checkpoints:** persistent state snapshots for graph-based LLM components, enabling resumability and post-hoc inspection.

This persistence layer enables multiple runs per paper without conflicts, supports comparisons across reruns (e.g., different LLMs or policy versions), and provides the system-of-record required for reviewer-facing explainability.

**Distributed claiming and fault tolerance.** Nodes transition through explicit states (e.g., `READY`, `RUNNING`, `SUCCESS`, `FAILED`, `SKIPPED`), and a periodic scheduler claims runnable nodes and dispatches them to asynchronous workers. To support multi-worker execution safely, claiming relies on database-level concurrency control (row-level locking with `SELECT ... FOR UPDATE SKIP LOCKED`). This prevents duplicate work when multiple workers compete for tasks, while allowing horizontal scaling.

Operationally, a workflow run progresses through a repeated scheduling loop:

1. create a *workflow run* for a paper and initialize all nodes from the selected workflow definition;
2. mark nodes with no unsatisfied dependencies as `READY`;
3. a scheduler periodically claims `READY` nodes and dispatches them for execution;
4. upon node completion, downstream nodes whose dependencies are satisfied transition to `READY`;
5. the loop continues until all nodes reach a terminal state.

To avoid deadlocks and ensure progress under worker failures, claims are time-bounded, and stale claims can be released by periodic cleanup tasks.

**Idempotency and retries.** Because external calls (network downloads, LLM APIs) can fail transiently, the nodes are designed to be idempotent and safe to retry. Retry limits are encoded at the node level, and the workflow engine can re-attempt a failed node without corrupting upstream artifacts. This is consistent with the methodological objective of partial completion: upstream results remain available even when some branches fail.

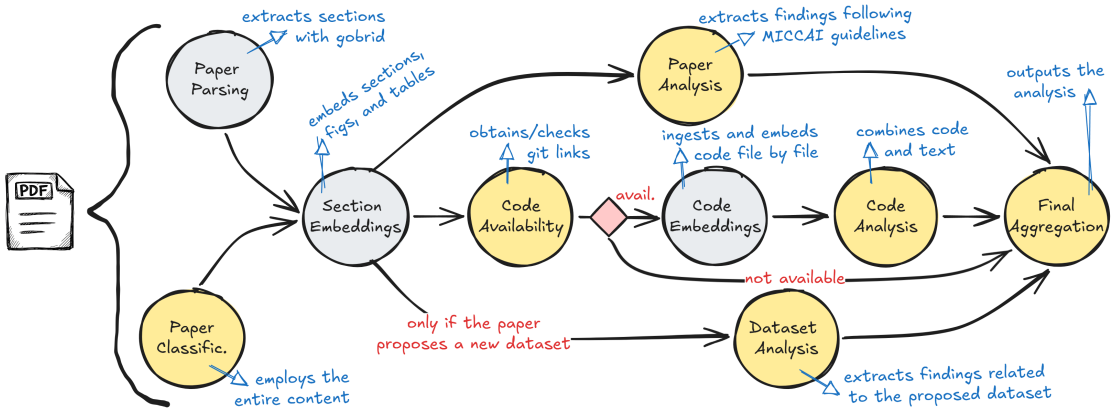



Figure 3.1: End-to-end workflow of PAPER-SNITCH. The diagram distinguishes model-assisted steps (LLM calls in yellow) from deterministic processing, and shows the conditionally executed branches (e.g., dataset and code analysis) through explicit routing markers (diamonds and branch labels).

**Artifacts as first-class outputs.** Every node is expected to produce structured outputs and artifacts that can be inspected. This supports evidence grounding (by linking findings to stored excerpts and repository files) and supports auditability (by exposing how each score was produced).

### 3.2.3 Concrete Analysis Graph

In its current version, the analysis is implemented as an 9-node DAG that combines paper-side, code-side, and dataset-side checks with a deterministic aggregation step. At a high level, the workflow is summarized in Figure 3.1

The responsibilities of each node align with the methodological decomposition introduced in the remainder of this chapter:

- **Paper Parsing:** Paper parsed into sections using GROBID
- **Paper Classification:** Paper-type classification used for conditional routing and weighting.
- **Section Embeddings:** Embedding construction over section-wise chunks to support retrieval.
- **Paper Analysis:** Paper-side reproducibility checklist evaluation via retrieval and structured outputs.

- **Dataset Analysis:** Dataset documentation checklist, conditionally executed when a dataset contribution is detected.
- **Code availability:** Extraction and validation of repository links.
- **Code Embeddings:** Repository ingestion, file selection, chunking, and embedding for code retrieval.
- **Code Analysis:** Code-side analysis over retrieved repository artifacts, grounded against paper claims.
- **Final Aggregation:** Deterministic aggregation into component and overall scores, plus a constrained qualitative synthesis.

### 3.2.4 Explicit `Skip` Semantics for Inapplicable Branches

A central methodological requirement is to distinguish between (i) a node that could not be executed due to an error, and (ii) a node that is deliberately inapplicable for a given input. For example, a dataset documentation checklist should not be executed for a purely methodological paper.

To make this distinction explicit, nodes can terminate with four outcomes: *success* (artifact produced), *failed* (artifact missing due to error),*cancelled* (artifacts didn't run due to previous failed node) or *skipped* (node was not required). This `skip` semantics is critical for interpretability of downstream aggregation: a skipped branch should not penalize the paper, whereas a failed branch indicates that relevant checks could not be completed. Explicit skipping also improves reporting: reviewers can see that an aspect was intentionally not evaluated (e.g., "dataset checklist skipped: no dataset contribution detected"), rather than conflating it with missing evidence.

### 3.2.5 Non-execution Principle: Static Inspection and Bounded Retrieval

Executing arbitrary third-party repositories is unsafe and infeasible during peer review. It introduces security risks (malware, credential exfiltration, prompt injection via build logs), nondeterminism (dependency resolution, network variability), and substantial compute costs (especially for GPU workloads). Accordingly, PAPER-SNITCH follows a strict non-execution principle: it never runs untrusted code.

Instead, it relies on static inspection and bounded retrieval over verifiable surfaces such as:

- the presence and contents of key files (e.g., licenses, environment specifications, entry-point scripts);
- documented commands and configuration instructions in READMEs and scripts;
- availability of evaluation scripts and pretrained artifacts, when claimed;
- consistency between paper claims and repository contents.

This methodology aligns with the thesis scope: review-time screening should support human judgment by surfacing objective signals and concrete evidence, rather than attempting full reproduction.

### 3.2.6 Provenance Tracking and Adversarial Robustness

LLM-based components are restricted to bounded subtasks (classification, evidence extraction, criterion-level judgment) and are coupled with provenance requirements. Every criterion evaluation is expected to return: (i) a structured decision, (ii) a short explanation, and (iii) a set of evidence pointers that reference the stored artifacts (paper section/chunk identifiers, repository file paths, or URLs). This provenance layer serves two purposes.

First, it reduces hallucination risk by requiring claims to be supported by retrievable text spans. Second, it enables post-hoc audits and comparisons across reruns (e.g., different model versions or updated policies).

Because inputs are not trusted, the methodology also assumes an adversarial setting. PDFs and repositories can contain instructions intended to manipulate model behavior. To limit this attack surface, PAPER-SNITCH uses bounded prompts, criterion-specific tasks, restricted context windows (retrieval instead of full-document prompts), and deterministic aggregation logic. This reduces the "blast radius" of malicious instructions and makes suspicious behaviors easier to detect during inspection.

## 3.3 Paper Processing Pipeline

Before PDF parsing and criterion evaluation can take place, PAPER-SNITCH must acquire a consistent snapshot of paper-side artifacts: submission metadata, the PDF, and (when available) supplementary material and review artifacts. Because MICCAI metadata is not exposed through a public API, early prototypes explored generic website-to-markdown extraction. In practice, this approach was not sufficiently complete for the project objectives (e.g., meta-review information was not reliably captured), which motivated a transition to a configurable crawler. The final acquisition process follows a two-stage strategy: (i) parse the conference index page to collect paper identifiers, titles, authors, and links; (ii) visit each paper page to extract richer fields such as the abstract, code and dataset links, DOI, supplementary material, and review-related content. These artifacts are stored explicitly so that downstream evaluations operate on stable, inspectable inputs.

Concretely, an initial approach based on a generic web extraction service (`jina.ai`) did not reliably capture all review-side metadata. This failure case motivated the adoption of a configurable crawling library (`crawl4ai`), which provides finer control over navigation and extraction.

### 3.3.1 Parsing PDFs into Structured TEI/XML with GROBID

PDFs are layout-centric artifacts that are difficult to analyze directly: reading order is implicit, text is fragmented, and structural elements (sections, captions, references) are not reliably encoded. To obtain a representation more suitable for downstream retrieval and evidence linking, PAPER-SNITCH parses PDFs into a structured TEI/XML format using GROBID [7, 19].

Early prototypes attempted direct text extraction from PDFs (e.g., lightweight parsing libraries such as `PyPDF2`) and also explored passing the PDF itself to LLMs. These approaches were adequate for preliminary experiments, but they did not provide the stable section-wise structure required for reliable retrieval and provenance, and they made it difficult to verify whether scores were grounded in the extracted text. The transition to a TEI/XML representation therefore served a methodological goal: making evidence localization and reviewer inspection feasible under bounded contexts.

The TEI representation preserves the document hierarchy (e.g., section boundaries) while remaining plain text at the content level, which facilitates chunking, indexing, and citation of evidence. In practice, this step improves robustness relative to treating the PDF as a single flat text stream.

### 3.3.2 Section-wise Storage and Caption Extraction

Rather than storing the entire paper as a monolithic blob, PAPER-SNITCH stores section-wise text. When feasible, figure and table captions are also extracted and stored separately. This design has two methodological advantages.

First, it improves efficiency: many criteria are localized (e.g., dataset provenance, training protocol, evaluation details), and searching within relevant sections reduces wasted retrieval and token budget. Second, it improves traceability: reviewers can be shown exactly *where* in the manuscript a supporting statement appears, and evidence locators can reference stable identifiers such as (section, paragraph, chunk).

### 3.3.3 Paper-type Classification for Content-aware Routing

Different contribution types imply different reproducibility obligations. For example, a dataset paper has orthogonal requirements (collection protocol, annotation process, access restrictions) compared to a methods paper. To avoid diluting checks into a generic rubric, PAPER-SNITCH classifies each paper into a small set of types (dataset / method / both / theoretical) using an LLM.

This classification acts as a lightweight, content-aware routing mechanism. It enables conditional branches (e.g., running the dataset checklist only when a dataset contribution is detected) and supports paper-type-dependent weighting during aggregation. The methodology favors this approach because it provides flexibility at low engineering cost compared to building a full, rule-based discourse parser.

### 3.3.4 Chunking into Short, Overlapping Spans

Reproducibility evidence is often sparse and localized. Feeding long contexts to LLMs risks diluting relevant statements and exacerbates known long-context failure modes [18]. Therefore, after section-wise extraction, PAPER-SNITCH chunks the paper into short, overlapping spans (approximately 2,000 characters per chunk, with a 20% overlap).

Chunking yields fixed-size units that are suitable for embedding, retrieval, and bounded prompting. The overlap reduces boundary effects where a key sentence might otherwise be split across two chunks.

### 3.3.5 Dense Embeddings and Reusable Vector Index

To retrieve candidate evidence for each criterion, PAPER-SNITCH builds dense embeddings for all chunks and stores them in a reusable vector index. Dense retrieval supports semantic matching even when the criterion wording does not directly match the paper phrasing, consistent with retrieval-augmented generation practices [12, 15].

The index is cached and reused across criteria and reruns, amortizing cost. This is particularly important because the same paper is evaluated against many criteria, and reruns are common during development (e.g., ablations, model comparisons). Embeddings are computed using a fixed embedding model interface [29].

### 3.3.6 Checklist Definition and Criterion Representations

Criteria are defined explicitly as a checklist aligned with MICCAI reproducibility guidance [22]. This provides "guardrails" that reduce subjectivity and encourages consistent, decomposable scoring. Each criterion includes a name, a description, and (when useful) examples of what would constitute acceptable evidence.

To further stabilize retrieval behavior, PAPER-SNITCH precomputes a vector representation for each criterion (e.g., embedding over name + description + examples). Because criteria are fixed for a given policy version, this avoids repeated computation and supports versioned policies.

### 3.3.7 Retrieval of Top-$k$ Spans with Conservative Thresholds

For a given criterion, PAPER-SNITCH retrieves the top-$k$ candidate chunks by cosine similarity between the criterion representation and the paper chunk embeddings. The methodology is intentionally conservative: retrieval operates under strict token budgets, and similarity thresholds are used to prefer "evidence not found" over overconfident matches. This bounds the context exposed to the LLM and reduces the chance of hallucination by limiting irrelevant text.

### 3.3.8 Criterion-level Evaluation with Structured Outputs

Each criterion is evaluated independently. Instead of a single global prompt asking the model to judge the entire paper, the system performs multiple focused calls: one criterion at a time, grounded on the retrieved candidate spans. This decomposition reduces the cognitive load on the model and yields more interpretable outputs.

This design was validated through iterative testing with multiple LLMs and prompt variants. Early experiments highlighted both output variance and sensitivity to prompt structure when evaluating broad criteria. Moving to criterion-specific calls, combined with explicit schemas, improved relevance and interpretability, and reduced the risk that the model "fills in" missing information from prior knowledge rather than from paper evidence.

All LLM responses are required to follow a structured schema (e.g., a categorical decision, confidence/strength indicator, evidence pointers, and a short rationale). In the implementation, these schemas are validated before storage (e.g., using Pydantic-style data models), so that downstream aggregation can rely on well-formed, machine-readable outputs. Structured outputs reduce downstream parsing ambiguity and enable deterministic aggregation without additional post-processing steps.

### 3.3.9 Separating LLM Extraction from Deterministic Scoring

In PAPER-SNITCH, LLMs are used to extract and assess evidence at the criterion level, but numeric scoring is computed deterministically. This separation is methodologically important: arithmetic and aggregation are stable and reproducible, while LLM outputs can vary across runs. By constraining generative variability to evidence identification and bounded judgments, and by keeping scoring logic deterministic, the system reduces "score drift" and improves auditability.

An intermediate prototype separated evidence extraction and scoring into two sequential LLM calls. While conceptually appealing, this doubled end-to-end latency in the tested setup without a commensurate quality gain, and it increased the number of failure points. This failure-driven refinement motivated the final approach: a single criterion-level call that returns both an extractive evidence set and a bounded judgment, followed by deterministic scoring.

This design also supports explicit provenance. Each criterion score is linked to a small set

of evidence locators (chunk identifiers, section names, and quoted spans) that can be displayed to reviewers.

## 3.4 Code Repository Processing

### 3.4.1 Repository Resolution and Sanity Checks

Code processing begins with lightweight, objective checks. Given a repository link extracted from the manuscript (or provided explicitly), PAPER-SNITCH verifies reachability and performs basic sanity checks (e.g., non-empty repository, non-placeholder content). These checks provide immediate signals and prevent wasting compute on missing artifacts.

In practice, papers may contain multiple candidate repositories, multiple PDFs, or multiple dataset references. To handle this ambiguity robustly, the pipeline was extended with disambiguation steps that enumerate candidates, preserve them in provenance logs, and apply bounded heuristics (and, where necessary, model assistance) to select the most plausible target for deeper analysis. This refinement emerged from repeated failure cases during development, where naive "first link wins" selection produced unstable or misleading evaluations.

### 3.4.2 Repository Ingestion via Digest Representation

Repositories can be large and heterogeneous. To standardize ingestion and remain within strict token budgets, PAPER-SNITCH uses a repository-to-text digest tool (GitIngest) to obtain a prompt-friendly textual snapshot of repository structure and key files [6]. This representation reduces implementation complexity, provides a consistent substrate for retrieval, and supports caching.

In addition, the methodology assumes that only a subset of files is typically relevant for reproducibility checks (e.g., installation instructions, environment declarations, training/evaluation entry points, configuration files). Accordingly, code analysis is preceded by lightweight preprocessing that prioritizes high-signal files and directories. When repositories are too large to be processed holistically, PAPER-SNITCH uses bounded model assistance to select candidate files for deeper inspection, and then chunks selected files into short, overlapping spans using the same principle adopted for paper text. This yields fixed-size units that can be indexed and retrieved under controlled

budgets.

Importantly, the digest is treated as an *inspection artifact*, not as an executable environment. It is suitable for detecting the presence of documentation, dependency declarations, entry points, and other verifiable signals.

### 3.4.3 Targeted Components of Code Analysis

Rather than prompting an LLM with the entire repository and asking for a single global judgment, PAPER-SNITCH decomposes code analysis into targeted components aligned with how reviewers reason about artifacts. Examples include:

- methodology and experiment workflow detection (e.g., what the repository appears to implement and how it is intended to be used);
- dependency and environment specification (e.g., `requirements.txt`, `environment.yml`, container files);
- entry points and execution commands (scripts, CLI instructions, experiment runners);
- reproducibility artifacts (e.g., configuration files, evaluation scripts, pretrained weights, checkpoints);
- reporting of splits, seeds, and other stability-critical settings;
- documentation quality (README completeness, installation steps, expected inputs/outputs).

Component-wise analysis yields interpretable outputs and enables paper-type-dependent expectations. For instance, a theoretical paper may legitimately omit training scripts, while a methods paper claiming a full pipeline should provide runnable entry points and evaluation procedures.

### 3.4.4 Joint Grounding on Repository Evidence and Paper Claims

Reproducibility evidence is split across the paper and the repository. Moreover, mismatches between them are informative (e.g., the paper claims that scripts are provided, but the repository lacks entry points). Therefore, PAPER-SNITCH conditions code analysis on both sources: the model is asked to evaluate repository artifacts in light of the paper's claims, and to explicitly flag inconsistencies.

This joint grounding transforms the task from "summarize the repository" into a bounded consistency check that produces actionable findings.

## 3.5 Dataset Branch and Conditional Evaluation

Dataset contributions impose obligations that are orthogonal to code availability. When paper-type classification indicates a dataset component, PAPER-SNITCH activates a dedicated dataset documentation checklist aligned with MICCAI guidance [22].

This checklist focuses on verifiable reporting elements such as:

- cohort characterization and descriptive statistics;
- acquisition protocol and annotation procedures;
- access constraints, licensing, and ethics statements;
- train/validation/test splits and evaluation protocols.

When no dataset contribution is detected, this branch is explicitly `skipped`, preventing cost and avoiding penalizing non-dataset papers.

## 3.6 Aggregation and Reviewer-Facing Reporting

### 3.6.1 Exposing Intermediate Scores and Artifacts

PAPER-SNITCH is designed as decision support. Therefore, the report always exposes intermediate component scores and criterion-level artifacts. This decomposition allows reviewers and chairs to focus on specific shortcomings, identify sources of uncertainty, and verify evidence directly.

A practical lesson from development is that provenance must remain verifiable at the PDF surface. Even with structured parsing, extracted text can diverge from the visual rendering due to layout artifacts (e.g., formulas, headers/footers, hyphenation), and models may paraphrase instead of quoting. To mitigate this, the reporting layer emphasizes extractive evidence locators and supports PDF-facing verification (e.g., by highlighting retrieved spans) so that users can directly inspect what the system relied upon.

### 3.6.2 Weighted Aggregation with Paper-type-dependent Weights

The overall verifiability score is computed as a weighted average of component scores. Weights depend on the classified paper type. For example, dataset papers place more emphasis on dataset documentation and access transparency, while method papers place more emphasis on code availability and runnable evaluation artifacts.

This weighting encodes a pragmatic review rationale and can be tuned per venue or policy version. Because aggregation is deterministic, changes in weights are explicit and auditable.

### 3.6.3 LLM Use Restricted to Qualitative Synthesis

While numeric scoring is deterministic, a short narrative summary can improve usability. In PAPER-SNITCH, an LLM may be used to synthesize a qualitative summary *only* from already-computed artifacts and criterion outcomes. The model is not allowed to generate or adjust numeric scores. This constraint prevents score manipulation and reduces the impact of generative failures.

## 3.7 Infrastructure and Operational Considerations

The methodological requirements above (boundedness, safety, auditability, partial completion) translate into concrete infrastructure decisions.

### 3.7.1 System Architecture and Service Boundaries

PAPER-SNITCH is implemented as a web service with explicit separation between the user-facing application and the long-running analysis pipeline. At a high level, the deployed system includes: a web backend (Django), asynchronous workers (Celery), a relational database (MySQL/MariaDB), a message broker (Redis), a dedicated PDF parsing service (GROBID), and external LLM/embedding APIs. This separation isolates failures (e.g., parser errors or API rate limits) and simplifies scaling by increasing worker capacity without modifying the web tier.

In a typical deployment, a reverse proxy (e.g., NGINX) fronts the web application to handle HTTP termination and static file delivery, while the analysis pipeline runs out-of-band through the worker pool.

**Containerized deployment.** The system is deployed as a containerized web service. Containerization provides self-contained environments, reduces incompatibility risks across updates, and isolates failures to individual services.

In the development environment, dependency management also evolved iteratively: issues with `pipenv` locking motivated a transition to `uv` and corresponding container updates, improving build reproducibility.

In practice, PDF highlighting was implemented using `PyMuPDF`, producing reviewer-facing artifacts that make evidence verification faster than manual searching.

**Web backend and asynchronous execution.** The user-facing interface is served by a Django backend. Analyses are long-running and I/O-heavy (PDF parsing, repository ingestion, embedding, LLM calls), so they are executed asynchronously via a worker queue (Celery). This prevents request timeouts, supports concurrency, and enables robust progress tracking.

In practice, conference-scale screening requires that a large batch of papers can be triggered with minimal user interaction. Asynchronous execution, combined with a persistent workflow state, allows the system to schedule work across a pool of workers and to show progress and partial results as nodes complete.

**Database as system-of-record.** Auditability requires durable storage of inputs, node outputs, errors, and token usage. Accordingly, a persistent database stores analysis runs and their artifacts. This design also supports caching and reuse: expensive intermediate artifacts (e.g., embeddings for a fixed paper snapshot) can be reused across reruns.

The workflow engine also uses the database as a coordination mechanism for distributed execution: workers claim runnable nodes through transactional locking, ensuring that each node is executed at most once at a time even under high concurrency.

**Scheduling and housekeeping.** Workflow execution relies on periodic scheduling. In practice, a beat scheduler triggers (i) a workflow scheduler task that claims `READY` nodes and dispatches them, and (ii) a cleanup task that releases stale claims when workers crash or exceed expected execution windows. These background tasks are essential to guarantee liveness at scale.

**Administrative inspection.** Because the system is intended for auditability, the implementation exposes workflow runs, node states, logs, and stored artifacts through an administrative interface. This makes it possible to inspect how a result was produced, debug failed nodes, and compare reruns for the same paper.

**Dedicated parsing service.** GROBID is deployed as a dedicated service/container. PDF parsing is a specialized dependency; isolating it improves operational stability and prevents parsing failures from coupling tightly with the rest of the application.

**Versioned workflow logic.** Analysis logic is expressed as a versioned state graph (LangGraph) [13, 14]. Versioning supports scientific accountability: when the pipeline changes, the system can record which graph/policy version produced a given score and report.

**Observability and cost accounting.** To support scalability and experimentation, the system records operational metadata such as execution durations and token usage per node and per model. This makes it possible to reason about throughput and costs (e.g., how many papers can be processed under a fixed budget), and to run controlled comparisons between models and pipeline variants.

This instrumentation was not purely an engineering convenience: without explicit token accounting, model comparison and iterative prompt refinement became difficult to control. Treating token usage and latency as first-class artifacts informed conservative retrieval budgets, caching strategies, and the decision to keep scoring deterministic.

# 4. Implementation Details

This chapter describes the concrete engineering choices used to operationalize the methodology in Chapter 3. The emphasis is on the system components that turn heterogeneous, untrusted inputs (conference websites, PDFs, and external links) into structured, database-backed artifacts suitable for downstream evidence retrieval and reproducibility scoring.

## 4.1 Conference Crawling and Metadata Acquisition

A prerequisite for any reproducibility screening pipeline is access to a consistent snapshot of paper-side artifacts and metadata. For the MICCAI conference website, this includes (at minimum) the paper title, authors, and the URL of the paper details page; in addition, the project targets review-time signals that are only present in the paper details page, such as the abstract, code and dataset links, DOI, supplementary material, and reviewer-facing content (reviews, author feedback, and meta-reviews).

### 4.1.1 Crawling vs. Scraping

Although the term *web scraping* is often used as an umbrella label, the MICCAI paper portal exposes the relevant information as static, server-rendered pages with a stable HTML structure. As a consequence, the acquisition task does not require browser automation workflows that emulate user interactions (e.g., clicking, scrolling, waiting for dynamic DOM updates). In this setting, a *crawler* that fetches pages and extracts structured fields is sufficient. This constraint informed two key decisions.

In early prototypes, we tested generic website-to-markdown extraction services (e.g., `jina.ai`) to reduce implementation effort. However, these approaches did not reliably capture all review-side fields required by the downstream evaluation (most notably meta-review information), which motivated the transition to a configurable crawler.

First, heavy-weight browser automation tools such as Selenium [35] were considered unnecessary overhead: they add complexity, increase operational brittleness (e.g., timing issues and

headless browser compatibility), and complicate deployment inside Dockerized workers. Second, fully LLM-driven extraction at run time (where each page is sent to an LLM to infer the desired fields) was deliberately avoided. While LLM-based scrapers can adapt to diverse websites, they are costly at scale and introduce additional variance: the MICCAI portal contains thousands of papers, and a per-page LLM call would make acquisition tightly coupled to an external model both in latency and in price.

### 4.1.2 Schema-Driven Extraction with Crawl4AI

The implemented solution uses Crawl4AI [3] and, in particular, its JSON-CSS extraction approach. The central idea is to use an LLM only once to infer an extraction schema from a representative HTML sample, and then apply that schema deterministically across all pages that share the same structure.

At runtime, the crawler relies on Crawl4AI's asynchronous execution model to fetch pages and produce both raw HTML and a markdown rendering. The crawler can be used either as an asynchronous context manager or through an explicit lifecycle, which is useful in long-running services. Internally, Crawl4AI defaults to a Playwright-based crawling strategy, so starting the crawler corresponds to initializing the headless browser resources needed to retrieve and render pages.

Each fetch is executed via `arun(url, config=. . . )`, where the `CrawlerRunConfig` object controls crawl behavior (e.g., caching mode, thresholds, and the extraction strategy). The output is a structured `CrawlResult` that exposes multiple representations of the same page: the rendered HTML (when available), a markdown projection (used in this project for section parsing), and an `extracted_content` field filled by the configured extraction strategy. This design matches the development rationale documented in the Crawl4AI tutorial transcript: when the HTML pattern is repetitive across many pages, an LLM can be used once to derive a schema, after which extraction is performed deterministically without further model calls.

When no cached schema is available, the scraper fetches an HTML sample from the conference index page, truncates it to a bounded size (50 KB) to remain within typical token limits. The resulting schema is persisted for reuse in two places: (i) as a JSON file, and (ii) optionally inside the database as part of the `Conference` model. This dual persistence ensures that schema generation is

amortized across runs while remaining inspectable and versionable.

### 4.1.3 Extraction Pipeline and Normalization

After obtaining a schema, acquisition proceeds in two stages. First, the crawler downloads the conference landing page and applies the schema to extract a list of paper entries, each containing a title, an author list (as either a string or a nested list of author objects), and a link to a paper details page. Second, for each extracted paper URL, the service fetches the details page and relies on Crawl4AI's markdown conversion to obtain a simplified textual representation. The implementation then performs a deterministic segmentation step by extracting all top-level sections via regular expressions, producing a dictionary that maps normalized section names to their corresponding content spans. This representation is convenient for downstream parsing and for auditing, because it retains the website's explicit information architecture.

Even within a single conference portal, page templates may evolve across years (e.g., MICCAI 2022 vs. later formats) and fields may appear in slightly different textual forms. The crawler therefore performs a lightweight normalization layer that converts raw markdown sections into canonical database fields.

Examples include: (i) normalizing author lists into a single comma-separated string, as the extracted schema may return either a list of objects or a plain string; (ii) extracting dataset links by parsing "name: <URL>" patterns into a dictionary; (iii) extracting code repository links from an angle-bracket representation (e.g., `<https://github.com/...>`); and (iv) extracting DOI and PDF URLs with multiple patterns to cover both newer and legacy page layouts.

To avoid silently losing information when an unexpected field is present, the database ingestion step computes the difference between the scraped keys and the expected model fields. Any unexpected keys are preserved in a dedicated `metadata` JSON blob attached to the `Paper` record. This design supports forward compatibility: when the portal adds a new field, the system remains stable and the information can be incorporated later without re-crawling.

### 4.1.4 Scalability and Persistence

Crawling a conference-scale website requires balancing throughput with rate limits and resource usage. The implementation uses asynchronous I/O and constrains parallelism via semaphores. Specifically, per-paper crawling is wrapped in a semaphore, limiting the number of simultaneous requests.

To prevent unbounded memory growth when handling thousands of papers, the pipeline processes the paper list in batches. Each batch is crawled concurrently, then saved to the database, and only then proceeds to additional post-processing steps. This structure yields predictable memory usage while preserving most of the benefits of asynchronous execution.

The crawler is integrated into the Django backend and persists outputs directly into the relational database. Each paper is stored as a `Paper` entry associated with a `Conference` and linked to zero or more `Dataset` objects. The save routine is wrapped in a transactional boundary. This makes crawling idempotent: rerunning the crawler updates existing entries without creating duplicates, which is essential during development and for periodic recrawls.

Finally, the acquisition step optionally downloads the PDF (and supplementary materials, if available) using a dedicated HTTP client and stores the resulting bytes in Django `FileFields`. The subsequent PDF text extraction step is executed after the database write and is rate-limited independently, so that failures or slowdowns in PDF parsing do not compromise the correctness of the website crawl.

In summary, the MICCAI acquisition component is implemented as a schema-driven crawler that (i) uses an LLM only for one-time schema generation, (ii) performs deterministic extraction and normalization on the resulting markdown sections, and (iii) persists all artifacts and metadata in an idempotent, database-backed representation. This choice matches the methodological constraints of boundedness, safety, and auditability: the crawler does not execute untrusted code, it produces inspectable intermediate artifacts (schema, section text, URLs), and it scales to conference-sized workloads through concurrency and batching.

## 4.2 Website Interface and Reviewer Workflow

The system is exposed through a web interface designed to support two complementary needs: (i) administrative acquisition and monitoring at conference scale, and (ii) reviewer-facing inspection of paper-level results with traceability to intermediate artifacts. The interface follows a progressive disclosure pattern. Users start from a conference overview, drill down into a conference-specific paper list, and finally inspect a single paper through a workflow-centric view that emphasizes execution provenance (node status, logs, and stored outputs).

The web interface is intentionally lightweight. Long-running operations (conference acquisition and paper analyses) are executed asynchronously by worker processes; the frontend therefore emphasizes feedback loops: clear confirmation steps before starting large jobs, live status indicators, and periodic refresh of progress signals without requiring manual page reloads.

### 4.2.1 Conference List View: Entry Point and Acquisition Controls

The first view presents the set of available conferences as a searchable, paginated catalogue. Each conference card summarizes the current coverage (number of papers already acquired) and, when available, aggregate usage indicators (e.g., cumulative token consumption and per-paper averages from the latest runs). These aggregates serve a practical purpose during development and evaluation: they allow a quick sanity check of computational cost and help identify outlier conferences or runs.

For privileged users this view also acts as the control surface for acquisition. A dedicated action initiates the crawling process and optionally supports a limit parameter to restrict the number of papers processed (useful for testing and incremental debugging). Because acquisition may take minutes and involves many network requests, the interface includes an integrated log viewer: rather than returning a single blocking response, the system streams progress information to a scrollable console-like panel and exposes a stop action that interrupts the acquisition while preserving already stored results.

Finally, the interface supports onboarding new conferences through a structured form that captures the conference metadata (name, year, URLs, and optional branding). The form also allows an optional extraction schema to be provided manually (either by selecting a template or pasting a custom JSON schema). This is an important engineering affordance: while schema inference can

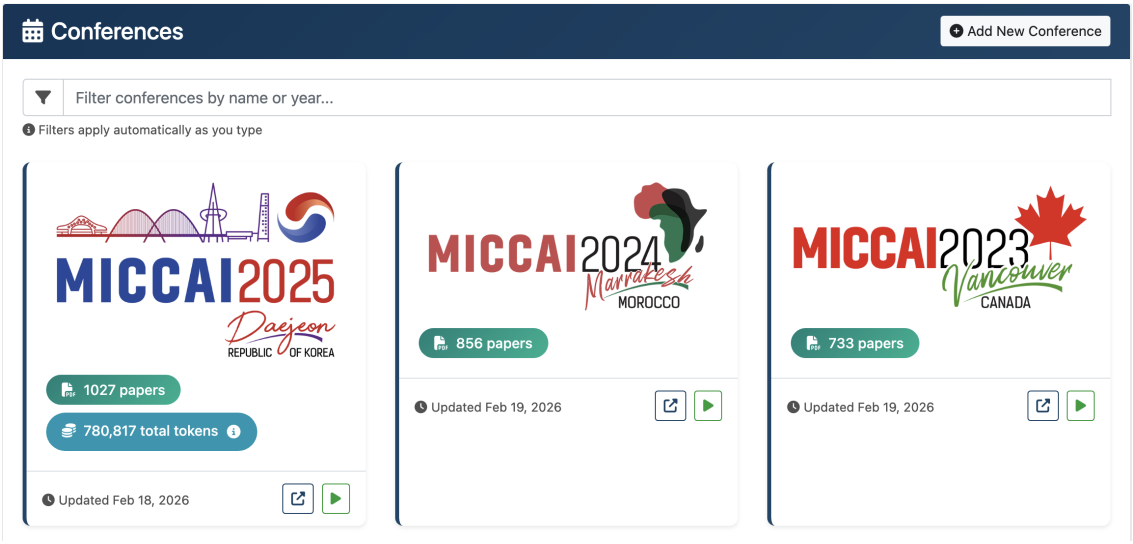



Figure 4.1: Conference overview page used as the entry point for acquisition and monitoring.

be automated, a manual override makes the acquisition pipeline more robust to website changes and enables rapid adaptation when the HTML structure deviates from expectations.

### 4.2.2 Conference Detail View: Paper Inventory and Bulk Workflow Operations

After acquisition, users interact primarily with a conference-specific view that lists all papers for that conference. The main table provides an at-a-glance inventory of each paper (title, DOI when available, and abbreviated author list) together with analysis state indicators: the status of the latest workflow run, the number of workflow runs associated with the paper, and a compact summary of token usage for the most recent completed run. These signals are designed for triage: they help identify which papers have not been analyzed, which are currently in progress, and which failed and may require intervention.

To support conference-scale operation, the interface includes bulk actions for privileged users. In particular, it allows restarting analyses for many papers at once (selecting which workflow definition to apply, optionally forcing recomputation even when cached artifacts exist, and previewing the affected papers before confirming). A complementary bulk stop action cancels running and pending workflows, which is useful for fault containment (e.g., when an upstream service becomes unavailable or a misconfiguration would otherwise enqueue a large number of failing jobs).

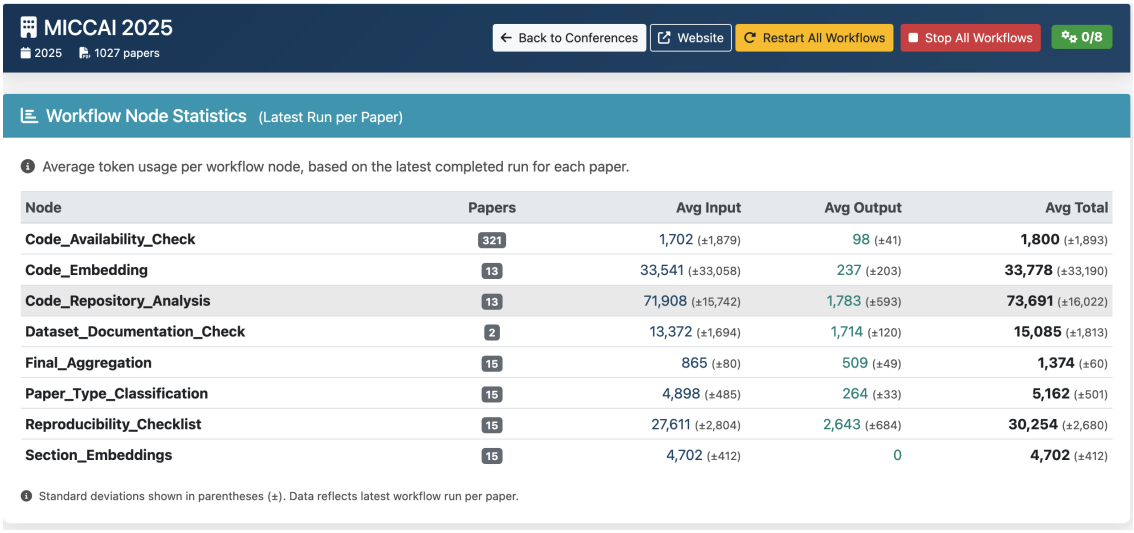



Figure 4.2: Conference detail page showing aggregate workflow-node statistics computed from the latest run for each paper.

Because both acquisition and analysis are asynchronous, the conference view implements periodic status refresh. Paper-level status badges and summary counters are updated automatically via background requests, keeping the page informative during long executions. When available, the view also exposes aggregate node-level statistics computed from the most recent runs (e.g., average token usage per workflow node). This helps diagnose where model cost concentrates and provides an empirical basis for optimization and ablation studies.

### 4.2.3 Paper Detail View: Workflow Provenance and Inspectable Artifacts

The paper-level view is centered around the idea that a reproducibility score must be auditable. Instead of showing only the final numeric outcome, the interface presents the most recent workflow run as a directed acyclic graph (DAG) where each node corresponds to a pipeline step (e.g., parsing, retrieval, criterion evaluation, aggregation). Nodes are visually annotated with execution state (completed, running, pending, failed, cancelled or skipped), enabling rapid identification of bottlenecks and failure points.

In addition to workflow-level information, the view provides direct access to the acquired paper PDF when available. This is intentionally exposed as a simple download action, consistent with the broader non-execution principle: the goal is to let the user inspect primary artifacts rather than to

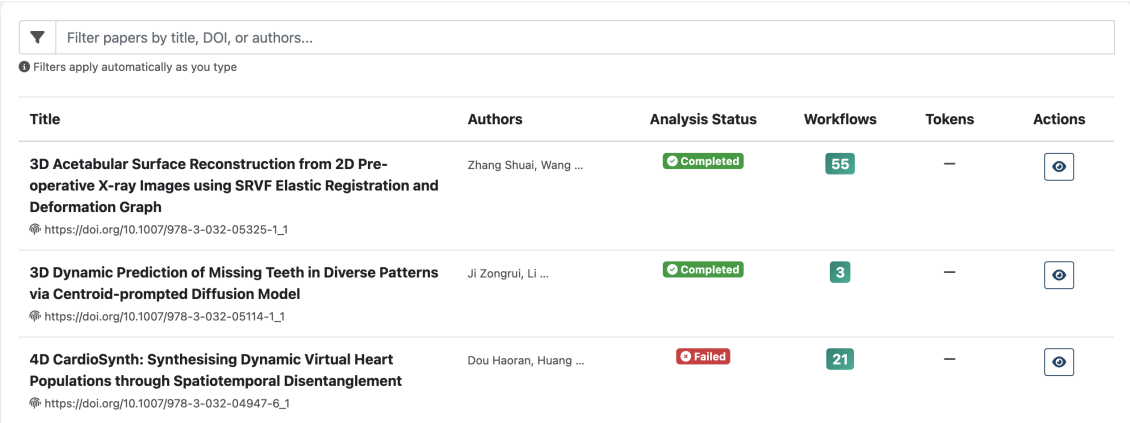



Figure 4.3: Conference detail page listing papers with workflow status and run counts for bulk monitoring and triage.

run them.

Selecting a node opens a detailed execution panel that reports structured provenance: identifiers, retry counts, timing information, and a live execution log. When node outputs are available, the panel displays the stored artifacts produced by that step, including the final aggregated assessment as well as structured criterion-level data intended for comparison with human annotation. This tight coupling between node state, logs, and stored artifacts operationalizes the methodological requirement of provenance: it allows users to verify what evidence was used and how intermediate results contributed to the final report.

To further support inspection, the node details panel includes an option to open a PDF view where relevant evidence spans can be highlighted. This bridges the gap between abstract, structured outputs (e.g., evidence pointers) and the human-readable source document, making it easier to validate whether the reported evidence is correctly localized.

The paper view also supports controlled reruns. Users can trigger a new workflow run by selecting the workflow type and the configured language model, and can optionally force recomputation to avoid reusing cached artifacts. For debugging and iterative development, rerun granularity is further refined by allowing the execution of a single node or a partial workflow restart from a chosen node onward. This capability is aligned with the system's modular design goal: it accelerates iteration while preserving the ability to inspect and compare outcomes across runs.

The automated pipeline described in Chapter 3 relies on *evidence locators*: machine- or human-

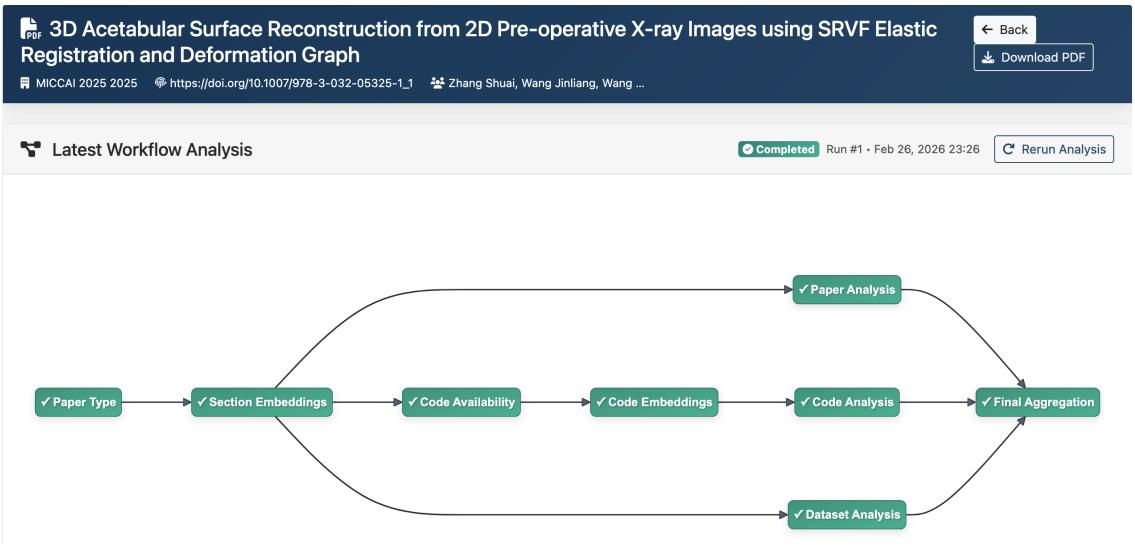



Figure 4.4: Paper detail view presenting the latest workflow run as a directed acyclic graph with node-level completion status.

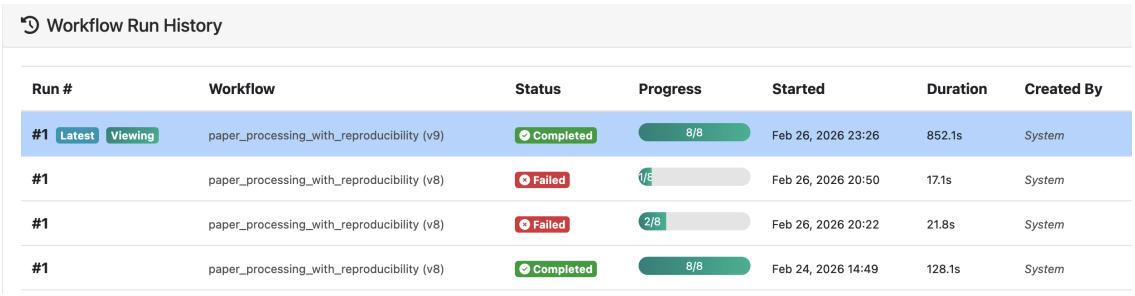



Figure 4.5: Workflow run history view supporting reruns and comparisons across workflow versions.

produced pointers that connect a claim (e.g., "code is available") to a concrete span of text in the primary artifact (the PDF). In order to (i) create a reliable human reference for evaluation and error analysis, and (ii) provide a practical way to inspect and validate the locators produced automatically, the system includes an internal web application, hereafter called the *Annotator*.

### 4.2.4 Design Goals and Workflow Rationale

The Annotator was designed around two complementary goals. First, it provides a user-friendly interface for human annotators tasked with applying the same reproducibility-oriented workflow used by the automatic system. Concretely, this means collecting labeled evidence spans for a predefined set of categories (criteria), so that the resulting annotations can serve as ground truth for

quantitative comparison and qualitative analysis.

Second, the Annotator acts as an inspection tool for evidence locators computed by the pipeline itself. A workflow-centric system can produce structured outputs, but debugging and trust calibration require the ability to rapidly confirm whether a predicted locator truly points to the relevant passage. The Annotator therefore emphasizes re-openability and persistence: once a document has been annotated (by a human or by an automated process that emits compatible locators), the interface can re-render the same highlights deterministically.

### 4.2.5 PDF-to-HTML Rendering for Interactive Selection

Direct interaction with PDF files in the browser is possible, but fine-grained selection, stable anchoring, and highlight rendering are substantially simpler on HTML. For this reason, the Annotator converts each uploaded PDF into an HTML representation and serves it through the Django application.

Each uploaded artifact is stored as a `Document` object, which persists the PDF file, the generated HTML file (if available), and a conversion status field used to make failures explicit and recoverable. The conversion step first attempts a high-fidelity rendering using `pdf2htmlEX` [32] executed in a dedicated Docker image. This choice preserves layout and typography more faithfully than plain text extraction, which is important for usability when annotators must locate short evidence spans in dense scientific documents. When Docker is unavailable or the conversion fails, the implementation falls back to a simpler conversion that extracts page text and generates a minimal HTML page structure. While this fallback loses layout fidelity, it preserves the key requirement for the annotation workflow: selectable text segments that can be anchored and revisited.

### 4.2.6 Annotation and Category Data Model

The Annotator persists both the controlled vocabulary of categories and the user-created evidence spans.

Categories are represented as `AnnotationCategory` objects. The model supports a two-level hierarchy (parent category and subcategories) to keep the interface navigable as the number of criteria increases. Each category also stores a color code used consistently across the UI and in

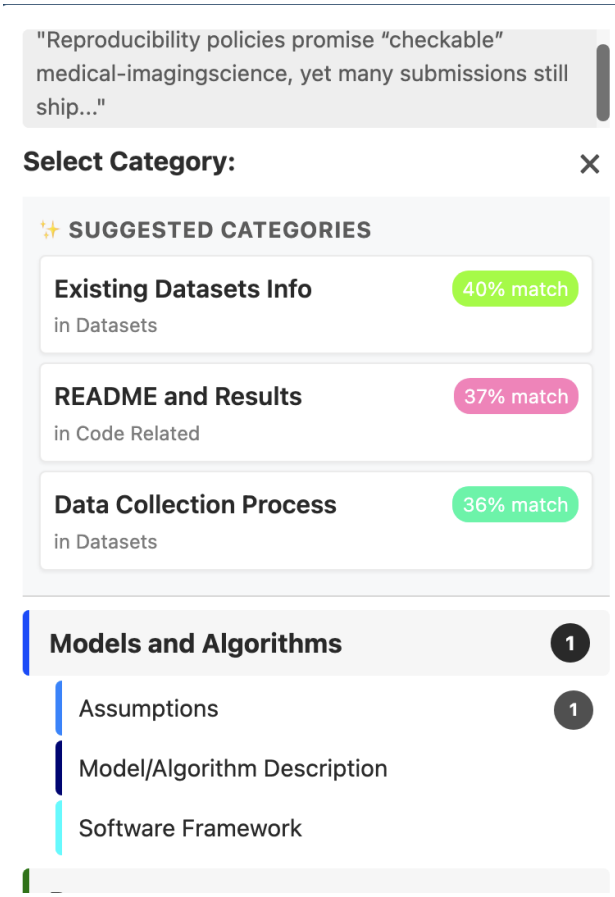



Figure 4.6: Annotator category selection dialog, including embedding-based suggested categories for the currently highlighted text span.

the rendered highlights, and a textual description used both for user guidance and for embedding-based recommendations. Finally, each category includes a stored embedding vector, which allows similarity search without recomputing the category representation at run time.

Individual annotations are represented as `Annotation` objects linked to a `Document`, a `User`, and a chosen category. Each annotation stores (i) the selected text span, (ii) a structural locator in the HTML representation (a CSS selector or XPath-like string), and (iii) a `position_data` JSON object that captures additional selection metadata required to re-identify the same span when the document is reopened. The model also optionally stores the embedding of the selected text (packed as a binary float array) and the corresponding cosine similarity to the chosen category embedding. Persisting these fields makes the interface auditable and enables post-hoc analysis of near-misses and systematic category confusions.

### 4.2.7 Embedding-Based Category Recommendation

The interface is designed for interactive, high-throughput labeling: when an annotator highlights a text span, the system immediately prompts for a category assignment. With a large number of categories, a purely manual selection would require frequent scrolling and would become a source of friction and inconsistency. To mitigate this, the Annotator includes a lightweight recommendation mechanism that surfaces the most similar categories at the top of the selector.

At selection time, the client sends the highlighted text to a server endpoint that computes an embedding vector and compares it against the precomputed category embeddings using cosine similarity. The top-$k$ categories (with $k = 3$ in the current implementation) are returned together with the embedding itself, so that the embedding can be stored alongside the annotation without requiring a second model call. The embedding model used is `text-embedding-3-small`, configured to produce 1,536-dimensional vectors.

The following code excerpt shows the two utility functions used by the endpoint: the wrapper around the embedding API and the cosine similarity computation.

```python
def get_embedding(
  text: str,
  dimensions: int = 1536,
  model: str = EMBEDDING_MODEL,
):
  if not client:
    print("OpenAI client not initialized (missing API key?)")
    return []

  text = text.replace("\n", " ")
  try:
    response = client.embeddings.create(
      input=[text], model=model, dimensions=dimensions
    )
    embedding_32 = np.array(response.data[0].embedding, dtype=np.float32)
    return embedding_32
  except Exception as e:
    print(f"Error getting embedding: {e}")
```

```
    return []
```

Code 4.1: Embedding computation function

```
def cosine_similarity(a: List[float], b: List[float]) -> float:
  """Calculates cosine similarity between two vectors."""
  if a is None or b is None:
    return 0.0

  a_arr = np.array(a)
  b_arr = np.array(b)

  norm_a = np.linalg.norm(a_arr)
  norm_b = np.linalg.norm(b_arr)

  if norm_a == 0 or norm_b == 0:
    return 0.0

  return float(np.dot(a_arr, b_arr) / (norm_a * norm_b))
```

Code 4.2: Cosine similarity function

In practice, category embeddings are computed offline from the category definition text and stored in the vector database, while text embeddings are computed on demand from the selected span only. This asymmetry is intentional: categories are stable and few, whereas the selected span is unknown in advance.

### 4.2.8 Persistence, Visualization, and Reuse for Automatic Locators

All annotations are persisted as first-class records and are scoped to the user that created them. When a document is reopened, the interface queries the stored annotations and reconstructs the visual highlights by applying the stored structural locator and selection metadata to the HTML content. This enables a consistent review experience: the same evidence spans are shown with the same category colors, and annotations can be edited (category reassignment) or deleted without reprocessing the underlying PDF.

Embeddings, by contrast, are stored and queried through a vector database. This architectural

separation is motivated by the access pattern of the recommendation and inspection tasks: both require repeated similarity queries in a high-dimensional space (here, 1,536 dimensions), where the natural operation is to retrieve the nearest neighbors under a distance metric such as cosine distance. While a relational database can store vectors as blobs or arrays, similarity search over such fields typically degenerates into linear scans or requires specialized extensions and careful index tuning. A vector database is designed specifically to support low-latency nearest-neighbor retrieval via approximate indexing structures, providing predictable performance as the number of stored vectors grows.

In the context of the Annotator, this choice improves user experience and scalability simultaneously. Suggestions can be produced interactively without enumerating all categories, and the same infrastructure supports retrieval of previously computed vectors (for example, when visualizing and comparing evidence locators generated automatically). Importantly, the relational store remains the system of record for human-interpretable artifacts (text spans, category identifiers, and locators), while the vector database acts as an acceleration layer for semantic lookup.

Beyond manual labeling, the same representation can be reused to visualize evidence locators produced automatically. When the analysis pipeline emits locators compatible with the Annotator schema (document identifier, category, and a locator expressed in terms of HTML structure and offsets), they can be persisted and rendered through the same highlighting mechanism. This unifies evaluation and inspection: human and machine evidence are comparable at the level of concrete spans, and discrepancies can be diagnosed by inspecting exactly what text was selected and how it was categorized.

## 4.3 Prompting: Structured API Usage and Prompt Engineering

Large language models are integrated into the system as bounded, inspectable components: they are invoked for tasks where natural language understanding is required (e.g., classification, targeted retrieval, and qualitative synthesis), while numerical scoring and aggregation are computed deterministically whenever possible. Two implementation choices support this design. First, model calls are structured through schema-constrained outputs in order to reduce parsing ambiguity and enforce stable data contracts. Second, prompts are treated as engineered artifacts: they encode

task definitions, constraints, and evaluation rubrics, and are iterated upon using systematic error analysis.

### 4.3.1 Structured OpenAI API Usage: `Create` vs. `Parse`

The implementation uses a mix of API patterns depending on the requirements of each node: (i) the *Responses API* is used when tool invocation is required (e.g., web search), combined with typed parsing via the `text_format` argument; (ii) the *Chat Completions API* is used for pure text reasoning and structured JSON outputs, either through explicit JSON schema enforcement (`create` with `response_format={type=json_schema}`) or through direct parsing into a Pydantic model [34] (`parse`).

The motivation for preferring `parse` when possible is twofold. First, it reduces boilerplate: instead of manually decoding raw JSON and handling validation errors, the returned payload is validated against a Pydantic schema and exposed as a typed object. Second, it improves robustness: schemas are declared with `extra="forbid"`, which makes unexpected keys explicit and prevents silent drift when prompts evolve.

**Example 1: Online Code Search with Tool Use**

The code availability pipeline includes a step that attempts to recover a missing repository link by querying the public web. This requires tool use (a web search capability), and therefore it is implemented with the Responses API. The output is constrained by a Pydantic schema, `OnlineCodeSearch`, which is passed to the API through the `text_format` parameter.

```
class OnlineCodeSearch(BaseModel):

  model_config = ConfigDict(extra="forbid")

  repository_url: Optional[str] = Field(
    description="URL to the code repository if found, null if not found"
  )
  confidence: float = Field(
    description="Confidence score between 0 and 1", ge=0.0, le=1.0
  )
```

```
    search_strategy: str = Field(
      description=(
        "How the repository was found (e.g., 'GitHub search', "
        "'author's profile', 'paper title match')"
      )
    )
    notes: str = Field(
      description=(
        "Additional notes about the search process or why repository wasn't
            found"
      )
    )
```

Code 4.3: Online code repository search class definition

```
response = client.responses.parse(
  model=model,
  message=[
    {
      "role": "system",
      "content": (
        "You are an expert at finding academic code repositories. "
        "Be conservative - only return URLs when you're confident "
        "they match the paper."
      ),
    },
    {"role": "user", "content": search_prompt},
  ],
  tools=[{"type": "web_search_preview"}],
  text_format=OnlineCodeSearch,
  reasoning={"effort": "minimal"},
)
```

Code 4.4: Tool-enabled LLM API call with structured output

Several parameters are worth highlighting. *Model selection* is workflow-configurable to support evaluation across backends. *Input messages* are provided as a short system instruction plus a user prompt that embeds the paper metadata and an explicit search strategy. *Tools* declares that the model

may call `web_search_preview`; this makes the operation closer to a bounded retrieval step rather than an unconstrained generation. *Structured parsing* is enabled by `text_format`, which requests an output that can be validated and directly materialized as a typed object. Finally, *reasoning control* is expressed through `reasoning`, which biases the model toward concise intermediate deliberation when set to `minimal`, helping to control latency and cost.

**Example 2: Paper Type Classification**

Paper type classification is implemented as a structured decision with a bounded label set (dataset / method / both / theoretical / unknown), a confidence score, and evidence quotes. In this node, a strict JSON schema is passed to the model through `response_format`. The resulting JSON is then decoded and validated against the Pydantic model.

```
response = client.chat.completions.create(
  model=state["model"],
  messages=[
    {"role": "system", "content": system_prompt},
    {"role": "user", "content": user_content},
  ],
  response_format={
    "type": "json_schema",
    "json_schema": {
      "name": "paper_type_classification",
      "strict": True,
      "schema": PaperTypeClassification.model_json_schema(),
    },
  },
  reasoning_effort="minimal",
)
```

Code 4.5: Paper type classification LLM API call with structured output

Here, `strict=True` makes the model adhere tightly to the schema, improving reliability when the output is consumed downstream by deterministic scoring logic. The `name` field assigns an explicit identifier to the schema, which is useful for logging and for differentiating multiple structured calls.

The schema itself is derived directly from the Pydantic model, ensuring that the declared data contract and the runtime validator remain synchronized.

**Example 3: Final Qualitative Synthesis**

The final aggregation node combines multiple component analyses into a final assessment. Scores and recommendations are computed programmatically, while the model is tasked only with producing qualitative narrative text (executive summary, strengths, weaknesses). This division enforces boundedness: the model cannot override computed values.

```python
class FinalQualitativeAssessment(BaseModel):
  model_config = ConfigDict(extra="forbid")

  executive_summary: str = Field(
    description=(
      "2-3 paragraph exec summary synthesizing key findings and implications"
    )
  )
  strengths: List[str] = Field(
    description="3-7 concrete reproducibility strengths across all dimensions"
  )
  weaknesses: List[str] = Field(
    description="3-7 specific gaps or areas needing improvement"
  )
```

Code 4.6: Final qualitative class definition

```python
response = client.chat.completions.parse(
  model=model,
  messages=[
    {"role": "system", "content": system_prompt},
    {"role": "user", "content": user_prompt},
  ],
  response_format=FinalQualitativeAssessment,
  reasoning_effort="minimal",
)

```

```
11  qualitative = response.choices[0].message.parsed
```

Code 4.7: Final qualitative LLM API call with structured output

Compared to the explicit JSON schema approach, `parse` eliminates an additional decoding step and yields a validated `FinalQualitativeAssessment` instance directly. Importantly, the prompt explicitly forbids the generation of scores and recommendations; only qualitative fields are accepted by the schema, reinforcing a separation between probabilistic language generation and deterministic computation.

### 4.3.2 Prompt Engineering Process and Prompt Quality

While structured schemas constrain the *shape* of model outputs, the *quality* of the content remains highly sensitive to the prompt. In this project, prompt engineering is treated as a core engineering activity because the model is asked to operate under non-trivial constraints: it must be conservative when evidence is weak, it must produce outputs aligned with a reproducibility rubric, and it must do so with predictable structure suitable for downstream aggregation and inspection.

The prompt development process followed an iterative, evidence-driven loop. First, each node prompt was derived from an explicit task specification: what decision is required, what information is available, and what failure modes are unacceptable. Second, prompts were written to include: (i) a role and task framing, (ii) an explicit label space or generation constraints, (iii) a set of operational guidelines, and (iv) a required output structure. Third, prompts were refined by examining logged outputs on representative papers and correcting systematic errors (e.g., overconfident URL attribution, ambiguous classifications, or verbose qualitative summaries that ignored the computed scores).

The three examples above illustrate recurring prompt design patterns. The following extracts show the kinds of hard constraints and decision rules encoded in prompts:

#### Online Code Search Prompt

```
You are a research code repository finder. Your task is to find the official
code repository for this paper.
```

```
Paper Information:
Title: {paper.title}
Authors: {authors_str}
Abstract: {paper.abstract or 'Not available'}

Search Strategy:
1. Search GitHub/GitLab/Bitbucket for repositories matching the paper title
2. Look for repositories from the paper's authors
3. Check for official implementations mentioned in the paper
4. Verify the repository actually corresponds to this specific paper

Platforms to search: GitHub, GitLab, Bitbucket, Gitee, Codeberg

Important:
- Only return a repository URL if you are reasonably confident it's the
    correct one
- If uncertain or no repository found, set repository_url to null and
    confidence to 0.0

Provide:
1. Repository URL (or null if not found/uncertain)
2. Confidence score (0.0 to 1.0) - only use >= 0.7 if very confident
3. Search strategy used
4. Notes about the search process
```

**Paper Type Classification Prompt**

```
Your task is to classify papers into one of these categories:
1. "dataset" - Papers primarily presenting a new dataset
2. "method" - Papers presenting a new method, model, algorithm, methodology,
    or benchmark
```

```
3. "both" - Papers presenting both a new dataset AND a new method
4. "theoretical" - Papers with purely theoretical contributions (proofs,
    mathematical frameworks, surveys, position papers) where executable
    code is not expected
5. "unknown" - Cannot determine from available information

Guidelines:
- Focus on the PRIMARY contribution of the paper
- Be confident in your assessment - use "unknown" sparingly
```

**Final Qualitative Assessment Prompt**

```
You will receive analyses and COMPUTED SCORES from multiple sources.
Your task: Generate ONLY executive summary, strengths, and weaknesses.
DO NOT generate scores or recommendations - those are computed programmatically.
```

In the online code search task, the prompt encodes a conservative policy (return a URL only when confidence is high) and a concrete search procedure, which reduces spurious matches and supports auditability through the `search_strategy` and `notes` fields. In paper type classification, the prompt includes a closed set of categories, disambiguation rules (e.g., dataset used to support a method is classified as `both`), and an explicit request for evidence quotes; this aligns the output with the broader evidence-grounding principle. In final aggregation, the system prompt makes the boundary conditions explicit ("do not generate scores or recommendations"), and the user prompt provides computed scores and component analyses as context; this encourages coherent synthesis while preventing the model from inventing quantitative claims.

Finally, prompt engineering is also a cost and latency control mechanism. By explicitly specifying what information is needed (and what is out of scope), prompts reduce irrelevant generation and make it feasible to run the pipeline at conference scale. Combined with structured parsing and conservative reasoning settings (e.g., minimal reasoning effort), this approach yields outputs that are both usable in the web interface and suitable for reproducible evaluation.

# 5. Evaluation and Results

This chapter evaluates PAPER-SNITCH as a decision-support system for review-time reproducibility screening. The evaluation focuses on two complementary questions. First, how closely do the system's structured criterion-level outputs match human judgments when both are expressed in the same rubric format? Second, what are the operational trade-offs (token usage, cost variability, and latency drivers) when running the pipeline at conference scale?

The evaluation is intentionally aligned with the methodological constraints in Chapter 3: the goal is not to reproduce results end-to-end, but to verify whether review-time signals can be extracted and summarized in a way that is evidence-grounded, auditable, and cost-feasible.

## 5.1 Experimental Setup

### 5.1.1 Paper Sample and Stratification

We consider a random sample of papers from the MICCAI 2025 proceedings, stratified by the paper-type classifier (dataset / method / both; see Section 3.3.3). The stratification is operationally important: it ensures that the evaluation includes papers where dataset- and code-side branches are expected to be active (both), in addition to the method-only cases.

In addition to this high-level protocol, the repository snapshot used to write this thesis includes a set of fully annotated, criterion-level artifacts for a representative subset of 10 papers (used for the detailed agreement analysis reported below). This subset contains papers classified as *method* and papers classified as *both* and includes the corresponding paper-side checklist outputs, and when applicable, dataset-side and code-side structured analyses.

### 5.1.2 Model Comparison Protocol

Each paper is processed by running the full workflow twice, swapping only the large language model used in the generation steps. All deterministic components are kept fixed across runs: parsing, chunking, retrieval logic, and the deterministic aggregation described in Section 3.6.

To isolate the effect of the language model on extraction and classification, we also keep the embedding model fixed. In the current implementation, the retrieval index is built using `text-embedding-3-small` with an embedding size of 1 536.

The workflow uses schema-constrained structured outputs for criterion-level decisions (Chapter 4, Section 4.3). For these structured calls, the decoding is configured to be near-deterministic (low temperatures in the 0.1–0.2 range). For synthesis tasks (final narrative summary), a slightly higher temperature (around 0.3) is used to improve readability without affecting numeric scoring.

### 5.1.3 Retrieval Depth (Top-$k$)

Many nodes rely on retrieval to ground model judgments in a small shortlist of candidate evidence spans (Section 3.3.5). We fix the retrieval depth to $k = 3$ for all retrieval-based nodes. Concretely: (i) the paper-side checklist retrieves the top-$k$ section chunks per criterion; (ii) the dataset documentation branch retrieves the top-$k$ section chunks per dataset criterion; and (iii) the code branch retrieves the top-$k$ embedded code/documentation chunks per component query, while preserving file paths and chunk indices for traceability.

A small $k$ is a deliberate bias toward boundedness and auditability: it makes the grounding set easy to inspect and limits the risk that irrelevant context dominates a criterion prompt.

## 5.2 Human Reference and Evaluation Protocol

Human reference scores are obtained from two annotators with more than 3 years of experience. Annotators fill the same rubrics exposed by Paper-snitch: (i) a paper-side checklist (20 criteria grouped into models/datasets/experiments), (ii) a dataset documentation checklist (10 criteria, conditionally executed), and (iii) the code-component checklist when a repository is available.

To reduce ambiguity-driven variance, annotators evaluate disjoint batches and discuss borderline cases to reach consensus before finalizing rubric scores. The final aggregation uses the same deterministic formulas as the automated pipeline (Section 3.6), ensuring that differences between human and automated scores are attributable to criterion-level decisions rather than to differences in arithmetic.

## 5.3 Agreement Metrics

The core comparison treats each criterion/component output field as a binary decision (present/absent). Agreement is reported at multiple granularities: (i) independently for paper, dataset, and code components; and (ii) globally, pooling all fields across components available for a paper.

We report two complementary measures.

**Percent agreement (accuracy).** Let $y_i \in \{0, 1\}$ be the human label and $\hat{y}_i \in \{0, 1\}$ the automated label for the same field. Percent agreement is the fraction of matching fields:

$$\text{Acc} = \frac{\sum_i \mathbb{1}[y_i = \hat{y}_i]}{\sum_i 1}.$$

**Matthews Correlation Coefficient (MCC).** Because many criteria are sparse (most papers do not satisfy many strict reproducibility items), accuracy alone can be misleading. We therefore also report the Matthews Correlation Coefficient (MCC) [21] computed from the pooled confusion matrix counts (TP, TN, FP, FN):

$$\text{MCC} = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}.$$

When the denominator is zero (degenerate cases where one class is missing), MCC is undefined; in those cases we report it as not applicable and exclude it from means.

## 5.4 Quantitative Results

Table 5.1 reports mean agreement on the subset of papers for which complete criterion-level human artifacts are available in the current repository snapshot. The sample size depends on branch applicability: the paper checklist is available for all $N = 10$ papers, while code and dataset branches are evaluated only when the corresponding artifacts are present.

Several observations follow. First, agreement is substantially higher in the code branch than in paper- and dataset-side criteria, reflecting that many code checks are closer to objective surface signals (e.g., the presence of environment files, documented commands, checkpoints) while paper-side criteria often require nuanced interpretation and can be confounded by PDF parsing artifacts.

Table 5.1: Mean agreement with human labels on the available fully annotated subset. "Global" pools all binary fields across paper/dataset/code components available per paper. The dataset branch is applicable only to papers classified as *dataset* or *both*.

| **Analysis Category** | **GPT-4o** | | | | **GPT-5** | | | |
|---|---|---|---|---|---|---|---|---|
| | **% Agr.** | **MCC** | **Token**$_{IN}$ | **Token**$_{OUT}$ | **% Agr.** | **MCC** | **Token**$_{IN}$ | **Token**$_{OUT}$ |
| Code | 90.70 | 0.81 | 95,370 | 1,330 | 84.80 | 0.68 | 126,750 | 2,650 |
| Paper | 77.54 | 0.57 | 37,910 | 2,810 | 66.28 | 0.38 | 37,910 | 5,780 |
| Dataset | 72.22 | 0.29 | 13,390 | 990 | 65.00 | 0.18 | 13,390 | 1,710 |
| Global | 80.40 | 0.60 | 78,800 | 3,510 | 70.89 | 0.43 | 78,800 | 7,980 |

Second, in this snapshot the GPT-4o configuration yields higher agreement than GPT-5 on the criterion-level decisions. This should not be interpreted as a universal model ranking; instead it indicates that prompt+schema configurations can interact with the model in non-trivial ways, and motivates the prompt-engineering and ablation analyses described below.

Finally, agreement varies across papers even within the same configuration. On this subset, global agreement ranges from 72.47% to 86.14% for GPT-4o, and from 59.71% to 80.79% for GPT-5. This spread motivates reporting distributions and error modes, not only means.

**Paper-type classification.** In our random pool, paper-type classification was always correct. This matters operationally because paper type gates conditional branches (dataset documentation, code analysis routing) and controls aggregation weights; a misclassification would therefore distort both cost and scoring.

## 5.5 Token Usage and Cost Accounting

A practical screening tool must be cheap enough to run routinely. PAPER-SNITCH logs token usage per node as input and output tokens and aggregates them over a workflow run. This instrumentation supports budgeting, throughput planning, and reproducibility-of-the-tool itself (a chair can identify

which nodes dominate cost).

In the evaluation protocol, the mean cost per paper is $0.23 for GPT-4o and $0.43 for GPT-5 when using current API rates for text tokens [28], including the text embedding costs. Cost variability is increased by the presence and size of code repositories: papers with accessible repositories trigger additional ingestion, embedding, and multiple criterion-level checks, while papers without code can skip the code branch entirely (Section 3.2.4).

## 5.6 Qualitative Error Analysis

Aggregate agreement hides systematic failure modes. To support targeted improvements, we analyze which criteria are most frequently disagreed upon between human and automated labels.

Across the available annotated subset, the most frequent paper-side disagreements concentrate in criteria that are both (i) semantically subtle and (ii) vulnerable to incomplete evidence extraction.

For the GPT-4o configuration, the single most frequent disagreement is `evaluation_metrics` (disagreed in all 10 papers in the annotated subset), almost always as false negatives (human marked present, model marked absent). The next most common false-negative patterns include `mathematical_description`, `baseline_implementation`, `failure_analysis`, and `clinical_significance`.

For the GPT-5 configuration, disagreements are less concentrated in a single criterion, but the same class of false negatives remains prominent. The most frequent disagreements include `mathematical_description`, `baseline_implementation`, `evaluation_metrics`, and `hyperparameters_reported`, again predominantly as false negatives.

In the dataset branch, frequent disagreement criteria include licensing/availability statements and annotation-protocol documentation (e.g., `data_license`, `data_availability`, `annotator_details`, and `quality_control`). These items are often stated in dispersed or implicit forms (footnotes, supplementary links, institutional review board statements), which makes them easy to miss under retrieval constraints.

These patterns suggest two actionable directions for improvement. First, retrieval and evidence localization should be tuned for "metadata-like" statements that may appear in captions, appendices, footnotes, or boilerplate sections. Second, prompts should explicitly disambiguate what counts as

sufficient evidence for a criterion (e.g., what qualifies as an evaluation metric reporting) and should request extractive evidence spans rather than paraphrases.

## 5.7 Planned Ablations and Sensitivity Analyses

The evaluation above motivates additional ablations that can be executed within the same framework.

**Top-$k$ sensitivity.** While this thesis uses $k = 3$ as a conservative default, a sensitivity analysis over $k \in \{1, 3, 5\}$ can quantify the trade-off between boundedness and recall. Increasing $k$ is expected to reduce false negatives for dispersed criteria (e.g., evaluation metrics and licensing statements) at the cost of higher token usage and potentially noisier prompts.

**Prompt and schema ablations.** Because structured outputs make the data contract explicit (Chapter 4), prompts can be modified without changing downstream parsing. This enables controlled prompt ablations such as: (i) stricter evidence requirements (must quote exact metric name/definition), (ii) explicit "where to look" hints per criterion (methods/experiments/appendix), and (iii) calibration constraints (e.g., conservative confidence thresholds).

**Caching and reuse.** Finally, because section and code embeddings are cached, reruns are substantially cheaper than first runs. A practical extension of the evaluation is to report the fraction of token usage attributable to non-cacheable nodes (LLM calls) versus cacheable preprocessing (embedding construction), and to quantify how much cost is reduced during iterative prompt refinement.

# 6. Conclusions

This thesis investigated whether reproducibility screening can be operationalized as a set of *bounded* and *inspectable* checks that remain feasible under peer-review constraints. The resulting system, PAPER-SNITCH, is a review-time decision-support system that produces evidence-grounded reports over heterogeneous, untrusted artifacts (conference metadata pages, PDFs, and linked repositories) without executing third-party code.

The central contribution is not a new scientific model, but a concrete engineering methodology for translating venue guidance into criterion-level checks and for connecting each check to auditable evidence. Concretely, the system (i) decomposes reproducibility guidance into explicit criteria, (ii) retrieves a small, inspectable set of evidence candidates from manuscripts and repositories, (iii) constrains model-assisted steps to structured outputs grounded in that evidence, and (iv) aggregates criterion outcomes deterministically into component scores and a global verifiability summary. This decomposition is intentionally designed to support human judgment: it localizes disagreements to individual criteria, exposes the underlying evidence, and makes it possible to contest or refine outcomes without treating the final score as an opaque verdict.

The remainder of this chapter situates these contributions relative to the research questions in Chapter 1 (Section 1.4), clarifies the claims and boundaries of the approach, and summarizes limitations and future directions.

## 6.1 What the System Does and Does Not Claim

A key design position of PAPER-SNITCH is that review-time screening should not be conflated with full reproducibility. The tool does not attempt GPU-intensive reproduction, does not validate empirical correctness, and does not execute untrusted code. Instead, it focuses on the mechanical surface layer that can be checked safely and quickly: artifact presence, documentation completeness, environment specification, claim-to-command traceability, and consistency between manuscript statements and provided artifacts.

This scope is important for responsible deployment. A high score should be interpreted as

"many verifiability signals are present and internally consistent," not as "the results are guaranteed to reproduce." Conversely, a low score may reflect genuine omissions, but can also reflect legitimate constraints (restricted clinical data, licensing restrictions, delayed releases, or double-blind constraints on links) that require contextual interpretation.

More broadly, PAPER-SNITCH is designed to complement existing policy mechanisms (checklists, badges, and artifact evaluation) rather than to replace them. Its role is to reduce the mechanical burden of verification by producing a structured, evidence-linked report that reviewers can inspect quickly and critique when necessary.

## 6.2 Research Questions Revisited

This thesis was guided by four research questions (Section 1.4). The results in Chapters 3–5 provide the following answers.

**RQ1 (Operationalizing policy into verifiable criteria).** MICCAI-style reproducibility guidance can be operationalized as a structured checklist of review-time criteria by making two separations explicit. First, criteria are written as verifiable requests for evidence rather than as open-ended assessments (e.g., "provide precise commands to reproduce results" rather than "is the code good?"). Second, evidence extraction is separated from score computation: model-assisted components propose criterion-level findings with provenance, while aggregation is computed deterministically with a transparent scoring scheme (Chapter 3). This separation makes the mapping from policy to score auditable and allows the policy profile to be versioned and tuned without changing the underlying pipeline philosophy.

**RQ2 (Automated evidence extraction for bounded verification).** It is feasible to automatically extract a bounded set of relevant evidence from manuscripts and linked repositories without executing code, provided that the pipeline uses conservative retrieval and explicit provenance. In PAPER-SNITCH, PDFs are converted into section-wise text (via a structured representation) to support chunking and retrieval; repositories are inspected through static file analysis and embedding-based retrieval over documentation/code chunks; and each criterion is evaluated only on a small top-$k$ evidence shortlist (Chapter 3, Chapter 4). While this design does not guarantee perfect recall, it

satisfies the peer-review constraints that motivated the problem statement: safety (non-execution), boundedness (small contexts), and auditability (evidence pointers).

**RQ3 (Agreement with human assessment).** Chapter 5 shows that, on the repository snapshot used for detailed analysis, criterion-level agreement with human labels is achievable but uneven across components. Agreement is systematically higher for code-oriented checks than for paper- and dataset-side criteria, reflecting that many code checks correspond to clearer surface signals (presence of environment files, documented commands, pretrained artifacts), while manuscript-side criteria often depend on dispersed or implicit statements and are sensitive to PDF parsing and retrieval coverage. The error analysis further indicates that disagreements are dominated by false negatives on "metadata-like" criteria (e.g., reporting of evaluation metrics and baseline details), suggesting that improvements should prioritize evidence localization in captions/appendices and stricter extractive prompting rather than more permissive generation.

**RQ4 (Utility in the review workflow).** The system provides reviewer-oriented affordances consistent with decision support: it persists intermediate artifacts, exposes node-level provenance and logs, and links criterion outcomes back to concrete evidence spans (Chapter 4). This supports rapid triage (what is missing, what is ambiguous, what is unverifiable under constraints) and enables targeted follow-up rather than full re-analysis. However, the thesis also highlights that reviewer utility is ultimately a human-factors question: a dedicated user study with reviewers and area chairs remains necessary to quantify time savings, calibration effects, and the risk of over-trust in automated scores.

## 6.3 Limitations and Threats to Validity

**Granularity of the rubric and uncertainty representation.** The current evaluation largely reduces criteria to binary fields (present/absent). While this choice supports clear reporting and deterministic aggregation, it under-represents common real-world cases such as partial releases, restricted-access datasets, and "verifiable only post-acceptance" situations. A more faithful representation would support graded or ordinal levels and explicit uncertainty, and would require careful validation against expert judgment and downstream reproducibility outcomes.

**Policy dependence and tunable thresholds.** The meaning of "sufficient documentation" or "acceptable artifact availability" is venue- and community-dependent. Weighting choices, criterion definitions, and thresholds therefore should not be treated as universal constants. They must be tuned to each venue's policies and validated before any high-stakes deployment. A practical implication is that the system should expose versioned policy profiles and make the chosen profile explicit in all reports.

**Adversarial robustness and artifact gaming.** Because PAPER-SNITCH consumes untrusted natural-language artifacts (e.g., README files) and routes them into model-assisted steps, it inherits known risks around indirect prompt injection and "gaming" through strategically written text. The approach adopted in this thesis mitigates these risks by design (bounded retrieval, schema-constrained outputs, conservative prompting, and deterministic cross-checks against repository structure), but these defenses are not complete. Robust deployment would require systematic adversarial testing and continued hardening, particularly because peer review creates incentives for strategic writing.

**LLM non-determinism and model drift.** Even when inference is configured conservatively (e.g., low temperature and schema-constrained outputs), LLM-driven steps are not fully deterministic: small variations in decoding, backend updates, and occasional format deviations can lead to different extracted evidence candidates or criterion-level judgments across runs. In addition, model providers may update weights, safety layers, or default behaviors over time, which can change pipeline outputs even if the surrounding code is unchanged. While PAPER-SNITCH reduces sensitivity to this variability through bounded retrieval, caching of intermediate artifacts, and deterministic aggregation of criterion outcomes, some residual run-to-run variance remains a limitation. A practical implication is that operational deployments should explicitly version model configurations and prompts, log relevant metadata, and periodically re-calibrate thresholds and rubrics when model behavior shifts.

**Evaluation scope.** The empirical evaluation reported in Chapter 5 provides evidence that criterion-level agreement with human judgments can be achieved at a practical cost, but it remains limited in scope. Scaling the human reference to larger, more diverse samples and to additional annotators is necessary to quantify inter-annotator variability, to stress-test edge cases (e.g., restricted-access data), and to evaluate generalization across conferences and years.

## 6.4 Operational Feasibility

A central practical question is whether LLM-assisted screening is affordable and operationally stable at conference scale. For this reason, PAPER-SNITCH treats observability as a first-class requirement: it logs token usage per node, stores intermediate artifacts, and supports caching to avoid repeated preprocessing. In the evaluation protocol summarized in Chapter 5, the mean per-paper costs are on the order of a few tenths of a dollar, with variability driven largely by whether a code repository is present and how large it is (Section 5.5). These costs are not negligible, but they are within a range that makes conference-scale screening plausible for targeted use (e.g., as an auxiliary report for borderline submissions or for post-submission auditing). Moreover, the architectural choices in this thesis make costs tunable: retrieval depth, caching reuse, and model choice provide explicit trade-offs between boundedness, recall, and budget.

## 6.5 Future Work

The results in this thesis suggest several high-value extensions that align with the limitations and open questions above.

**Richer scoring schemas.** Move beyond binary present/absent fields toward graded labels and uncertainty-aware aggregation, supported by prompts and interfaces that explicitly capture "partial" and "not verifiable under constraints" cases.

**Robustness hardening.** Develop and evaluate systematic defenses against prompt injection and repository gaming. This includes adversarial testing, stricter isolation of untrusted text, and additional deterministic validators that cross-check model outputs against parsed repository structure and metadata.

**Sensitivity and ablation studies.** Perform controlled analyses of retrieval depth ($k$), prompt variants, and model choices, reporting trade-offs between boundedness, agreement, and cost. Because outputs are schema-constrained and scoring is deterministic, such ablations can be executed in a reproducible manner.

**Reviewer-centered evaluation.** Finally, because the intended role of PAPER-SNITCH is decision support, an important next step is a user study with reviewers and area chairs. Beyond label agreement, such a study should measure whether the evidence-grounded artifacts reduce review time, improve consistency, and help users identify missing or unverifiable elements without over-trusting automated scores.

**Governance and responsible deployment.** An additional direction is to formalize governance constraints and reporting norms for AI-assisted verification in peer review. This includes making model provenance and policy profiles explicit, documenting failure modes and calibration guidance, and defining where in the review workflow automated screening is appropriate (e.g., pre-checks for completeness versus post-acceptance artifact auditing).

## 6.6 Concluding Remarks

In summary, PAPER-SNITCH demonstrates that reproducibility screening can be implemented as a transparent, evidence-grounded pipeline rather than as an opaque judge. By decomposing venue guidance into explicit criteria, enforcing bounded and safe analysis, and persisting provenance-rich intermediate artifacts, the system provides a concrete baseline for scalable review-time support.

The broader implication is methodological. If AI is used to assist in the evaluation of human scientific artifacts, the system must be constrained, inspectable, and policy-aware, and it must be designed to make disagreement and uncertainty explicit rather than hidden. Under these conditions, automated screening can shift reviewer effort away from repetitive mechanical checks and toward substantive scientific judgment, while preserving the reviewer as the final accountable decision-maker.

# Bibliography


[1] Association for Computing Machinery (ACM). *Artifact Review and Badging*. Website. Accessed: 2026-02-06. 2020. URL: https://www.acm.org/publications/policies/artifact-review-and-badging-current.

[2] Jiahui Chen et al. " GLM-SFNet: Global-Local Vision-Mamba with Semantic Fusion for Medical Image Segmentation ". In: *proceedings of Medical Image Computing and Computer Assisted Intervention – MICCAI 2025*. 2025.

[3] Crawl4AI. *Crawl4AI: Web crawling and extraction library*. https://github.com/unclecode/crawl4ai. Accessed: 2026-03-27. 2026.

[4] CVPR. *CVPR 2026 Author Guidelines*. Website. Accessed: 2026-02-06. 2026. URL: https://cvpr.thecvf.com/Conferences/2026/AuthorGuidelines.

[5] Pengyu Dai et al. " GoCa: Trustworthy Multi-Modal RAG with Explicit Thinking Distillation for Reliable Decision-Making in Med-LVLMs ". In: *proceedings of Medical Image Computing and Computer Assisted Intervention – MICCAI 2025*. 2025.

[6] GitIngest. *GitIngest: Turn any Git repository into a prompt-friendly text digest*. https://gitingest.com/. Accessed: 2026-03-27. 2026.

[7] *GROBID*. https://github.com/kermitt2/grobid. 2008–2026. swh: 1:dir:dab86b296e3c3216e2241968f0d63b68e8209d3c.

[8] Odd Erik Gundersen and Sigbjørn Kjensmo. "State of the Art: Reproducibility in Artificial Intelligence". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018.

[9] Robert Hoyt, Alfonso Limon, and Anthony Chang. *Generative AI and scientific manuscript peer review*. 2025.

[10] ICPR. *Reproducible Research in Pattern Recognition (RRPR) Badge*. Website. Accessed: 2026-02-06. 2025. URL: https://icpr2026.org/rrprBadges.html.

[11] Sungyoon Jung, Donghyun Lee, and Won Hwa Kim. “ MindLink: Subject-agnostic Cross-Subject Brain Decoding Framework ”. In: *proceedings of Medical Image Computing and Computer Assisted Intervention – MICCAI 2025*. 2025.

[12] Vladimir Karpukhin et al. “Dense Passage Retrieval for Open-domain Question Answering”. In: *Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2020.

[13] LangChain. *LangGraph Overview (Python) — Documentation*. `https://docs.langchain.com/oss/python/langgraph/overview`. Accessed: 2026-03-27. 2026.

[14] langchain-ai. *langgraph: Build resilient language agents as graphs*. `https://github.com/langchain-ai/langgraph`. Accessed: 2026-03-27. 2026.

[15] Patrick Lewis et al. “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks”. In: *Advances in Neural Information Processing Systems* (2020).

[16] Rui Li et al. “ Multimodal Fusion Network with Distribution-based Tumor-Marker Imputation for Multi-Origin Metastatic Cervical Lymphadenopathy Classification ”. In: *proceedings of Medical Image Computing and Computer Assisted Intervention – MICCAI 2025*. 2025.

[17] Jingsong Liu et al. “ HASD: Hierarchical Adaption for Pathology Slide-Level Domain-Shift ”. In: *proceedings of Medical Image Computing and Computer Assisted Intervention – MICCAI 2025*. 2025.

[18] Nelson F Liu et al. “Lost in the Middle: How Language Models Use Long Contexts”. In: *Transactions of the Association for Computational Linguistics* (2024).

[19] Patrice Lopez. “GROBID: Combining Automatic Bibliographic Data Recognition and Term Extraction For Scholarship Publications”. In: *International Conference on Theory and Practice of Digital Libraries*. 2009.

[20] Lena Maier-Hein et al. “BIAS: Transparent reporting of biomedical image analysis challenges”. In: *Medical Image Analysis* 66 (2020).

[21] Brian W Matthews. “Comparison of the predicted and observed secondary structure of T4 phage lysozyme”. In: *Biochimica et Biophysica Acta (BBA)-Protein Structure* 405.2 (1975).

[22] MICCAI. *MICCAI 2025 Paper Submission Guidelines: Reproducible Research*. Website. Accessed: 2026-02-06. 2025. URL: https://conferences.miccai.org/2025/en/PAPER-SUBMISSION-GUIDELINES.html#reproducibleresearch.

[23] Microsoft. *Playwright for Python*. https://github.com/Microsoft/playwright-python. Accessed: 2026-03-27. 2026.

[24] Miryam Naddaf. “AI is transforming peer review—and many scientists are worried”. In: *Nature* 639.8056 (2025).

[25] NeurIPS. *NeurIPS Paper Checklist Guidelines*. Website. Accessed: 2026-02-06. 2025. URL: https://neurips.cc/public/guides/PaperChecklist.

[26] OpenAI. *GPT-4o Model — OpenAI API Documentation (Pricing and Specs)*. https://developers.openai.com/api/docs/models/gpt-4o. Accessed: 2026-03-27. 2026.

[27] OpenAI. *GPT-5 Model — OpenAI API Documentation (Pricing and Specs)*. https://developers.openai.com/api/docs/models/gpt-5. Accessed: 2026-03-27. 2026.

[28] OpenAI. *Pricing — OpenAI API Documentation*. https://developers.openai.com/api/docs/pricing. Accessed: 2026-03-27. 2026.

[29] OpenAI. *Vector embeddings — OpenAI API Documentation*. https://developers.openai.com/api/docs/guides/embeddings. Accessed: 2026-03-27. 2026.

[30] Mourad Ouzzani et al. “Rayyan—a web and mobile app for systematic reviews”. In: *Systematic Reviews* 5.1 (2016).

[31] Alison O’Mara-Eves et al. “Using text mining for study identification in systematic reviews: a systematic review of current approaches”. In: *Systematic reviews* 4.1 (2015).

[32] pdf2htmlEX. *pdf2htmlEX: Convert PDF to HTML without losing text or format*. https://github.com/pdf2htmlEX/pdf2htmlEX. Accessed: 2026-03-27. 2026.

[33] Joelle Pineau et al. “Improving Reproducibility in Machine Learning Research (A Report from the NeurIPS 2019 Reproducibility Program)”. In: *Journal of Machine Learning Research* 22.164 (2021).

[34] Pydantic. *Pydantic: Data validation using Python type hints*. `https://docs.pydantic.dev/`. Accessed: 2026-03-27. 2026.

[35] SeleniumHQ. *Selenium: Browser Automation*. `https://www.selenium.dev/`. Accessed: 2026-03-27. 2026.

[36] Victoria C. Stodden. “Reproducible Research: Addressing the Need for Data and Code Sharing in Computational Science”. In: *Computing in Science & Engineering* 12.5 (2010).

[37] Rens Van De Schoot et al. “An open source machine learning framework for efficient and transparent systematic reviews”. In: *Nature Machine Intelligence* 3.2 (2021).

[38] Zhiyuan Zhu et al. “ Hierarchical Corpus-View-Category Refinement for Carotid Plaque Risk Grading in Ultrasound ”. In: *proceedings of Medical Image Computing and Computer Assisted Intervention – MICCAI 2025*. 2025.