Automated Classification of Pull Requests in Scientific Software using LLMs

by

Shiyu (Vivienne) Zeng
B.S.ENG., University of Victoria, 2023

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

We acknowledge and respect the Lək̓ʷəŋən (Songhees and Xʷsepsəm/Esquimalt) Peoples on whose territory the university stands, and the Lək̓ʷəŋən and W̱SÁNEĆ Peoples whose historical relationships with the land continue to this day.

Automated Classification of Pull Requests in Scientific Software using LLMs

by

Shiyu (Vivienne) Zeng
B.S.ENG., University of Victoria, 2023

Supervisory Committee

Dr. Neil A. Ernst, Supervisor
(Department of Computer Science)

Dr. Daniel German, Departmental Member
(Department of Computer Science)

ABSTRACT

Scientific software relies on contributions that combine domain-specific expertise with software engineering skills, but identifying which contributions require deep scientific knowledge remains a persistent challenge in project maintenance. We analyzed 1,074 pull requests from three established scientific repositories, Trilinos, Mantid, and AMReX, and developed a binary classification framework that distinguishes contributions requiring scientific knowledge from those focused on software concerns. To contextualize model performance, we compare LLM-based classification against simple baseline approaches, demonstrating that shallow statistical methods are insufficient for capturing domain-specific distinctions. Our approach achieves near-human reliability, with DeepSeek-R1 demonstrating a Krippendorff's α of 0.789 through iterative prompt refinement and human validation. The analysis reveals different review characteristics: scientific contributions require 67% longer review times, involve 64% more unique reviewers, generate twice the discussion comments, and undergo over 300% more revision cycles than software-focused changes. These patterns persist after controlling for the size of the pull request and the effects of the repository. A validation study on 75 PlasmaPy issues achieves 89.33% accuracy, indicating the framework applies to other contribution types. These findings establish that LLM-based classification can effectively support automated triage in interdisciplinary software teams. This enables more efficient allocation of scarce domain expertise while empirically confirming that scientific contributions require different review processes.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Scientific software underpins modern discovery, but code reviews stall since maintainers must vet both domain knowledge and engineering quality. Scientific research across domains now heavily relies on software to conduct experiments, simulations, and data analyses, making the correctness and efficiency of scientific software paramount [18]. Because results produced by such software can drive important scientific decisions, rigorous software engineering practices are needed to ensure reliability [18].

Scientific software (also called research software) is typically developed by interdisciplinary teams that include domain scientists (e.g., physicists, biologists) often lacking formal software engineering training, alongside professional software engineers. This blend of expertise creates fundamental tensions in development practices, particularly in code review—a proven practice for catching defects and improving code quality that is less formal and less prevalent in scientific software development [10]. Studies have found that while research software teams do review code, they often lack a formal review process and have an inadequate number of qualified reviewers [10]. This gap is partly due to the unique challenges of interdisciplinary development: scientists may have different expectations and workflows than software engineers, leading to mismatches in review standards [40]. Ensuring effective code reviews in this context is difficult when reviewers must understand complex domain-specific logic as well as software design concerns.

Modern software projects (especially open-source) commonly use pull requests (PRs) as the mechanism for proposing and reviewing code changes. The pull-based development model has been shown to offer fast turnaround and increased community engagement in general software projects [15]. In scientific software repositories, contributions fall into two broad categories: scientific-specific PRs (those that implement or modify

domain-specific algorithms, scientific data processing, or computational models) and software-specific PRs (those that refactor code, improve infrastructure, fix generic bugs, update documentation, or enhance performance without altering scientific functionality). These different types of contributions require different expertise to review. For instance, a PR adding a new data analysis method may need a scientist's scrutiny for correctness, whereas a PR upgrading build scripts or refactoring modules benefits from a software engineer's input.

Prior work suggests that such domain versus general distinctions matter; for example, one study on infrastructure-as-code versus application code found distinct review patterns, with domain-specific code changes involving more discussion and churn than others [4]. In practice, maintainers must quickly identify the nature of each PR to assign appropriate reviewers. However, doing so manually is labor-intensive, especially as project size and contributor base grow. Open-source integrators report difficulty in maintaining code quality and prioritizing contributions under a high volume of incoming PRs [16]. In popular scientific software projects, the backlog of PRs awaiting review can slow down progress and deter contributors.

The rise of large language models (LLMs) such as ChatGPT, Gemini, and DeepSeek has opened new possibilities for automating software engineering tasks. However, prior automation for pull request triage treats all changes alike and does not target the scarce expertise demanded by science-intensive contributions. This research gap motivates our work: we investigate whether LLMs can automatically classify PRs by their domain knowledge requirements and how this classification reveals fundamental differences in review processes.

1.1 Motivation

Scientific software development requires collaboration between domain scientists and software engineers, yet the expertise required for reviewing contributions is often unclear. While prior work has focused on predicting review outcomes or optimizing reviewer assignment, less attention has been given to understanding how different types of contributions demand different forms of expertise and review processes.

In practice, PRs vary significantly in their knowledge requirements. Some contributions involve domain-specific scientific reasoning (e.g., proposing new scientific models or algorithms), while others focus on software engineering concerns (e.g., refactoring code, fixing bugs, improving test coverage). This distinction is often implicit and diffi-

cult to identify systematically. As a result, researchers and practitioners lack tools to characterize PRs based on their expertise requirements and to analyze how these differences influence review dynamics.

This work is motivated by the need to support researchers studying scientific software repositories. Given a set of PRs, an initial classification based on scientific knowledge requirements is necessary to enable deeper analysis of review processes. In this work, we investigate whether LLMs can be used to perform this classification, and how the resulting categorization reveals systematic differences in review processes. Rather than focusing on pre-review prediction or reviewer assignment, this study analyzes completed PRs to better understand the role of domain knowledge in collaborative development.

1.2 Problem Statement and Research Questions

Scientific software development is inherently cross-disciplinary, often requiring collaboration between experts in domain science (e.g., physics, biology, astronomy) and experts in software engineering. These collaborations surface complex challenges related to communication, tooling, and code contributions. One crucial area where this complexity manifests is in PRs, which are the central mechanism for proposing and discussing changes in version-controlled scientific software.

Maintainers and contributors often face uncertainty when reviewing PRs, especially when those PRs require deep domain-specific knowledge or advanced software engineering practices. Manual triage of PRs to determine their nature and the expertise required for review can be time-consuming and error-prone. This issue is exacerbated in research software repositories, where contributions can range from fixing numerical algorithms and introducing new data models to adding documentation or changing internal APIs.

This thesis investigates two core challenges: (1) quantifying differences in review processes between PR types, and (2)automating PR classification in research software using LLMs. Specifically, we address the following research questions:

RQ-1: ***What are the measurable differences between science-specific and software-specific PRs in terms of review characteristics, such as review time, number of comments, and reviewer participation?***

To address this question, we perform a comparative analysis of PR review behaviors based on the LLM-assigned classifications. We focus on quantifiable review metrics,

such as the time from PR creation to merging, the number of total comments, and the number of unique reviewers participating. This analysis allows us to examine whether scientific-specific PRs, those more reliant on domain knowledge, differ from software-specific PRs in terms of review length and interaction complexity. Understanding these distinctions can reveal underlying bottlenecks or collaboration needs and inform better review practices within scientific software development.

RQ-2: ***Can science-specific and software-specific PRs be accurately classified? How effectively can LLMs distinguish between them based on PR content?***

This question is motivated by the need for improved triage and developer support in projects where PRs often contain domain-heavy or implementation-heavy content. We explore how LLMs, when prompted with structured information from PR discussions and file changes, can distinguish whether the contribution is more scientific or more software-oriented. We further compare LLM-based classification with simple baseline models to assess whether advanced semantic reasoning provides measurable benefits over low-cost statistical approaches. In doing so, we aim to identify which PR characteristics signal the required expertise, and assess the consistency of these classifications across projects.

By answering these research questions, this thesis aims to illuminate how LLMs can support PR triage and how different types of contributions influence the review process. These findings can guide the development of practical tools and processes to improve collaboration and code quality in research software.

1.3 Contributions

This work addresses the intersection of software engineering and scientific computing by bringing advanced AI techniques to bear on the PR review process. The key contributions are:

- **LLM-Based PR Classification Framework**: We design and apply an approach that uses LLMs (ChatGPT, Gemini, DeepSeek) to automatically classify PRs from scientific software projects into scientific-specific vs. software-specific categories. Our work demonstrates that a simplified binary classification framework is far more reliable than complex multi-scale approaches, achieving near-human level reliability ($\alpha = 0.789$) with DeepSeek-R1.

- **Baseline Comparison for PR Classification: We implement simple baseline**

models, including a majority class (ZeroR) classifier and a Naive Bayes classifier, to assess whether low-cost approaches can approximate LLM performance. Our results show that these baselines fall significantly short, highlighting the importance of semantic reasoning in distinguishing scientific and software-specific contributions.

- **Human-LLM Validation Study**: We curate a dataset of PRs labeled by human experts and systematically compare the LLMs' classification results against these human annotations. This evaluation sheds light on the performance, strengths, and limitations of current LLMs in understanding and categorizing software changes, revealing that LLMs cannot be treated as interchangeable and that model selection is crucial for classification tasks.
- **Empirical Analysis of Review Processes**: We conduct a comprehensive study of code review characteristics for different PR types, revealing that scientific-specific PRs experience significantly longer review times (67% longer), require more reviewers (64% more), and generate more discussion comments (100% more) than software-specific PRs. These findings provide insights into how the nature of a code change influences review effort and dynamics.
- **Generalization to Issue Tracking Systems**: We extend our classification framework to issue tracking through a case study of PlasmaPy, demonstrating 89.33% accuracy and confirming that our approach generalizes beyond PRs to primarily textual contexts.

Overall, this research demonstrates a novel application of NLP advances to software engineering for science. It not only evaluates the feasibility of automating an important aspect of project management (PR classification) with AI, but also deepens our understanding of how scientific content impacts the software peer-review process. The insights from this study can help bridge the gap between domain scientists and software engineers, ultimately contributing to more efficient and effective development of reliable scientific software.

1.4 Thesis Outline

This thesis is organized as follows:

- **Chapter 1: Introduction**
  This chapter introduces the motivation for classifying PRs in research software development using LLMs. It outlines the research questions, contributions, and the overall context of this work.

- **Chapter 2: Background & Related Work**
  This chapter presents foundational concepts in research software development, the SciCat dataset, and the use of LLMs in software engineering. It also reviews prior work on PR classification, review processes, and LLM-based approaches to mining software repositories.

- **Chapter 3: Dataset & Experimental Setup**
  This chapter describes the dataset construction, labeling framework, and annotation process. It also presents the experimental setup and statistical analyses used to examine differences in review processes between scientific-specific and software-specific PRs (RQ1).

- **Chapter 4: Classification Models**
  This chapter introduces the LLM-based classification framework, including prompt design, refinement, and evaluation methodology. It also includes comparisons with baseline models and an external validation study on PlasmaPy issues (***RQ2***).

- **Chapter 5: Discussion & Limitations**
  This chapter discusses the implications of the findings, outlines limitations, and examines the role of LLMs in research software workflows.

- **Chapter 6: Future Work & Conclusion**
  This chapter summarizes the contributions of the thesis and proposes directions for future research.

- **Appendix**
  The appendix contains supplementary materials, including additional tables, prompts, and supporting results.

Chapter 2

Background & Related Work

2.1 Background

2.1.1 Scientific Software Development

Scientific software plays a central role in modern research, influencing both research methods and collaboration practices across scientific disciplines, functioning not only as a tool for discovery but also as a key research output [19, 20, 22].

Scientific knowledge refers to the domain-specific understanding that links scientific theory with computational implementation, consistent with definitions of scientific software as "application software that includes a large component of knowledge from the scientific application domain and is used to increase scientific knowledge for the purpose of solving real-world problems" [23]. Unlike general SE knowledge, this type of expertise encompasses understanding scientific principles, implementing and validating models or algorithms, and ensuring the scientific correctness of computational results [23, 41].

Hannay et al. found that many scientists learn software development through peer interaction and self-study, and frequently feel insufficiently trained in areas such as testing [17]. Despite limited formal training, studies show that scientists are aware of certain software engineering (SE) best practices and tend to emphasize productivity-oriented practices, including code reuse and the use of third-party libraries [2].

However, balancing scientific goals such as accuracy, performance, and discovery with SE concerns like maintainability, readability, and sustainability remains a persistent challenge. Sun et al. found that while contributors in the Astropy ecosystem often possess interdisciplinary expertise bridging science and software, tensions still arose

between those prioritizing scientific outcomes and those emphasizing software robustness [42].

In practice, this interdisciplinary context means that PRs in scientific software often contain both scientific and engineering contributions. Reviewers must therefore assess domain-specific logic alongside software design and quality. Prior studies show that contributors vary widely in their expertise across these domains [42], leading to differing priorities and occasional friction in collaboration.

2.1.2 Research Software and the SciCat Dataset

Research software differs from general-purpose software in that it embeds domain-specific knowledge and prioritizes scientific validity alongside code correctness. Kelly [23] characterized research software development as a process of knowledge acquisition by scientists, not a traditional engineering process.

The SciCat dataset [29] is a curated collection of research software repositories created by analyzing README files using LLMs. SciCat provides high-confidence annotations of scientific software across domains such as physics, chemistry, and biology, offering a scalable foundation for identifying scientifically intensive projects while reducing selection bias in our study.

2.1.3 LLMs for Software Engineering and Annotation

The rapid advancement of LLMs has started to influence SE practices and research. LLMs such as OpenAI's GPT series [5] and Codex [6] have demonstrated strong capabilities in code generation, natural language understanding, and reasoning about code.

Prior work has explored a wide range of SE applications for LLMs, including code completion, bug fixing, test generation, and documentation [11, 39]. Tools powered by LLMs, such as GitHub Copilot [6], have already been adopted by developers for routine coding assistance [47].

Researchers are also actively investigating how LLMs can support more specialized SE tasks. Benchmarks like SWE-Bench [21] assess model performance on realistic coding tasks, highlighting the ability of LLMs to understand, reason about, and modify complex software. Agentic approaches extend this further by enabling LLMs to act autonomously as problem-solving agents, integrating tools, memory, and reasoning to perform tasks such as code modification, testing, and documentation generation [48].

These developments demonstrate the potential for LLMs to understand domain-specific content and make informed decisions about software changes.

Ahmed et al. [1] demonstrated that LLMs can achieve inter-rater agreement levels similar to human annotators in labeling tasks. They propose using model-model agreement to evaluate reliability and suggest combining multiple LLMs to ensure robustness. This supports our approach of using multiple LLMs to classify PRs.

2.1.4 Prompt Engineering for Mining Repositories

The Prompt Refinement and Insights for Mining Empirical Software repositories (PRIMES) framework, as proposed by De Martino et al. [9], provides a structured and four-stage methodology for using LLMs in repository mining studies. It begins with (1) defining clear study objectives and selecting an appropriate prompt strategy to align the LLM's task with research goals. This is followed by (2) an iterative prompt piloting and refinement phase on a single LLM, where outputs are statistically validated against human annotations to improve clarity and accuracy. After that, (3) it evaluates among multiple LLMs, using metrics like classification accuracy and cost, the constructed oracle, and benchmarks to select the most suitable model. Finally, (4) output validation is conducted to detect and correct issues.

This framework is particularly suitable for our study on classifying PRs in scientific software because it directly addresses the core challenges of using LLMs for classifying tasks in the SSW development. The iterative refinement and statistical validation ensure our prompts reliably distinguish between scientific and software-specific PRs. Evaluating multiple LLMs allows us to select the most accurate and suitable model for analyzing PRs in a large dataset.

2.2 Related Work

2.2.1 Pull Request Classification and Characterization

Prior works have extensively examined PRs in open-source software, focusing largely on factors influencing PR acceptance or review latency rather than on the classification of PR content. Gousios et al. conducted an early large-scale study of the pull-based model on GitHub, finding that successful PRs typically exhibit minimal discussion and fast turnaround, with 80% of PRs merged within about 3.7 days [14].

Studies have identified a range of predictors for PR metrics like acceptance and merge time, including contributor experience, test presence, patch size, and social interaction patterns. Tsay et al. [46] categorized these predictors into technical factors (relating to the code itself) and social factors (relating to collaboration dynamics).

Our work introduces an orthogonal dimension: the domain knowledge context of a PR—specifically, whether it primarily reflects scientific or general software expertise. While technical and social factors describe how contributions are made or evaluated, our classification focuses on what kind of expertise the contribution requires.

While prior works have not explicitly classified PRs in research software projects based on domain-specific content, related works have explored automated PR categorization in other contexts. Fridh et al. [12] used transformer-based models trained on release notes to classify PRs by purpose, and Colavito et al. [8] applied LLMs in a zero-shot setting to label issues. Building on this emerging line of work, our study distinguishes PRs into two categories—Scientific-Specific and Software-Specific—to capture how domain knowledge influences scientific software development.

2.2.2 Analysis of PR Review Process

Understanding these categories also helps explain differences in how PRs are reviewed. Prior works have analyzed PR review processes, identifying factors that affect review time and decision outcomes. For example, Yu et al. found that more reviewers and discussion typically prolong reviews, whereas automated tests and contributor reputation accelerate them by building trust [49].

The intuitive assumption that smaller PRs merge faster has also been questioned: Kudrjavets et al., analyzing over 845,000 PRs, found no consistent link between PR size and review speed [25], suggesting that cognitive complexity and content type play a more decisive role than change size alone.

By explicitly distinguishing between scientific and software-focused PRs, we examine how the presence of domain-specific content influences review effort, as reflected in metrics such as review time, comment volume, and reviewer participation. No prior work has explored the impact of domain expertise on PR review dynamics in scientific software, making our contribution both novel and complementary to existing research on PR classification and review processes.

2.2.3 LLMs in Collaborative Development

In the context of PRs, Liu et al. proposed methods to automatically generate PR descriptions from commit data, helping developers summarize changes efficiently [28]. Similarly, studies have begun observing how developers integrate conversational LLMs into collaborative workflows. Chouchen et al. found that ChatGPT is often used on larger, more complex PRs, which tend to take significantly longer to close, suggesting that developers turn to LLM assistance for challenging or time-consuming tasks [7].

Our work complements prior PR-focused automation by shifting from artifact generation or generic classification to expertise-sensitive categorization that directly supports reviewer assignment and project analytics in scientific codebases [28, 7]. To mitigate known challenges in LLM-assisted SE—including risks of hallucination and code correctness concerns—we validate model judgments with human reviews, following emerging methodological guidance for reliable LLM use in repository mining [9, 3, 1].

2.3 Chapter Summary

This chapter outlines the interdisciplinary nature of scientific software (SSW) development, which requires a blend of domain-specific scientific knowledge and software knowledge. It introduces the SciCat dataset for identifying research software and proposes a novel classification of PRs as either Scientific-Specific or Software-Specific within SSW projects using LLMs. This demonstrates the potential of LLMs for reliable annotation and classification, guided by methodologies like the PRIMES framework.

Chapter 3

Dataset & Experimental Setup

3.1 Introduction

This chapter details the methodology developed to answer the first research question of this thesis:

RQ1: *What are the measurable differences between science-specific and software-specific PRs in terms of review characteristics, such as review time, number of comments, and reviewer participation?*

We leverage the SciCat dataset [29] to ensure domain relevance by focusing on high-confidence scientific repositories. We develop a systematic pipeline to select and process PRs from three representative projects (AMReX, Mantid, and Trilinos), including filtering and data extraction, to construct a high-quality dataset for subsequent classification and analysis. The extraction of metadata (e.g., discussion duration, labels, review activity) primarily supports the subsequent analysis, which investigates the correlation between PR type and review characteristics.

Figure 3.1 provides an overview of our classification methodology, which integrates automated LLM evaluation with human validation to ensure reliable results.

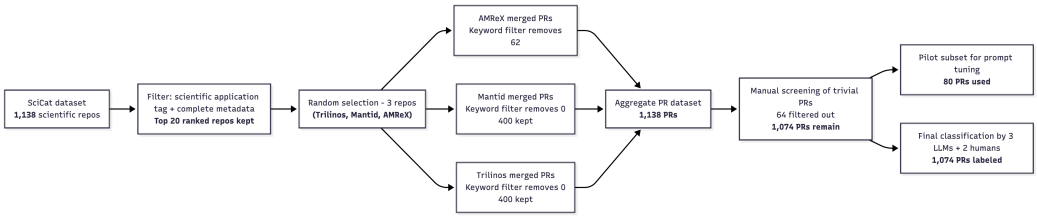

Figure 3.1: Dataset curation and classification workflow.

Our hypothesis is that scientific-specific and software-specific PRs exhibit significant differences across multiple review outcome metrics. To test this, we analyze the effect of a PR's scientific knowledge requirement on review outcomes using both group comparison tests and regression models that account for potential confounding factors.

The primary objective is to determine whether scientific-specific PRs differ from software-specific PRs in terms of process characteristics such as review duration, reviewer involvement, and extent of discussion. Understanding these patterns can inform strategies for optimizing reviewer assignment, improving triage efficiency, and supporting LLM-assisted workflows in research software development.

Section 3.2 describes the dataset construction process. Section 3.3 introduces the labeling framework, while Section 3.4 details the annotation and consensus procedures. Section 3.5 summarizes the dataset through descriptive statistics. Section 3.6 defines the evaluation metrics, followed by Section 3.7, which presents the statistical analysis. Finally, Section 3.8 reports the results of the hypothesis evaluation.

3.2 Dataset Construction

3.2.1 Project Selection

To construct a high-quality dataset for analyzing PRs in research software, we required a source of repositories that were both definitively scientific in nature and actively maintained to ensure sufficient PR data. Manual identification of such repositories at scale is infeasible and prone to selection bias. Therefore, we relied on the SciCat dataset [29], a curated collection of research software repositories introduced by Malviya-Thakur et al. We chose SciCat because it uses LLMs to analyze README files, providing high-confidence annotations that a project is research software within domains like physics, chemistry, and biology. This pre-vetting ensured high domain relevance for our study and allowed us to avoid the noise of generic software projects.

To narrow this list to a manageable set of suitable candidates, we developed a Python script to automate a multi-step filtering and ranking process. Each step was designed to select repositories that support reliable PR classification and enable subsequent analysis of review process characteristics.

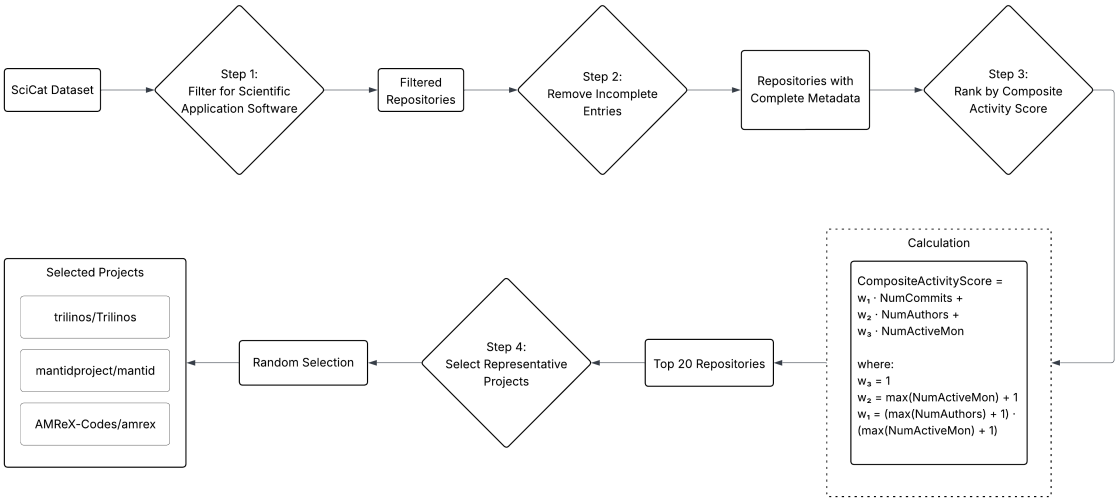

Figure 3.2: Project Selection Process

As shown in 3.2, it illustrates the four-step process used to select three representative research software projects from the SciCat dataset.

3.2.1.1 Step 1: Filter for Scientific Application Software

The SciCat dataset includes repositories classified as "scientific application software", "scientific tools", and "scientific libraries". We retained only those tagged as *scientific application software*. This focus ensures the selected projects are end-user applications or frameworks where the interplay between domain science and software engineering is most pronounced, providing the clearest examples of PRs that might blend these types of knowledge.

3.2.1.2 Step 2: Remove Incomplete Entries

Repositories missing critical metadata (e.g., commit history, author information) were excluded. This step is necessary to ensure the integrity of our subsequent activity analysis and to guarantee we can accurately extract PR data and calculate metrics like the number of contributors and active months.

3.2.1.3 Step 3: Rank by Composite Activity Score

To ensure our study is based on viable, collaborative projects with enough data for analysis, we ranked the remaining repositories. Each repository was scored using a compos-

ite index based on three primary indicators of health and activity:

1) Total number of commits: A proxy for the overall level of development and project maturity.

2) Number of distinct contributors: Indicates community engagement and a diversity of perspectives, which is crucial for generating a variety of PR types.

3) Number of months with active development: Measures sustained activity over time, reducing the risk of selecting abandoned projects.

We define the composite activity score as:

$$\textit{CompositeActivityScore} = w_1 \cdot \textit{NumCommits} + w_2 \cdot \textit{NumAuthors} + w_3 \cdot \textit{NumActiveMon},$$

where

- $w_3 = 1$,
- $w_2 = \max(\textit{NumActiveMon}) + 1$,
- $w_1 = (\max(\textit{NumAuthors}) + 1) \cdot (\max(\textit{NumActiveMon}) + 1)$.

Any increase in the number of commits contributes more to the composite score than the maximum possible change in either the number of contributors or the number of active months. Similarly, any increase in the number of contributors dominates any possible change in the number of active months. As a result, repositories are ranked primarily by development volume, secondarily by community size, and finally by sustained activity duration.

The use of this composite activity score prioritizes repositories with intensive and ongoing development, while avoiding the selection of projects that may appear active due to longevity alone but lack sustained collaborative contribution. The top 20 repositories based on this composite activity score are shown in Table 3.1.

Table 3.1: Top Repositories in SciCat Dataset

ProjectID	**URL**
pytorch_pytorch	`https://github.com/pytorch/pytorch`
tango-controls_cppTango	`https://github.com/tango-controls/cppTango`
trilinos_Trilinos	`https://github.com/trilinos/Trilinos`
apache_beam	`https://github.com/apache/beam`
root-project_root	`https://github.com/root-project/root`
commaai_openpilot	`https://github.com/commaai/openpilot`
gem_oq-engine	`https://github.com/gem/oq-engine`
mantidproject_mantid	`https://github.com/mantidproject/mantid`
project-millipede_jclouds	`https://github.com/project-millipede/jclouds`
Unidata_awips2	`https://github.com/Unidata/awips2`
eclipse_sumo	`https://github.com/eclipse/sumo`
kaldi-asr_kaldi	`https://github.com/kaldi-asr/kaldi`
lscsoft_lalsuite-archive	`https://github.com/lscsoft/lalsuite-archive`
cp2k_cp2k	`https://github.com/cp2k/cp2k`
lightningnetwork_lnd	`https://github.com/lightningnetwork/lnd`
geonetwork_core-geonetwork	`https://github.com/geonetwork/core-geonetwork`
opencollab_scilab	`https://github.com/opencollab/scilab`
GeoNode_geonode	`https://github.com/GeoNode/geonode`
ccxt_ccxt	`https://github.com/ccxt/ccxt`
AMReX-Codes_amrex	`https://github.com/AMReX-Codes/amrex`

3.2.1.4 Step 4: Select Representative Projects for Deep Analysis

From the ranked list in Table 3.1, we randomly selected three representative projects for detailed classification and analysis.

`trilinos/Trilinos` [45]: A collection of libraries for numerical solvers and multi-physics simulation on HPC systems.

`mantidproject/mantid` [44]: A software suite for processing data from neutron and muon scattering experiments.

`AMReX-Codes/amrex` [50]: A framework for solving partial differential equations on block-structured adaptive meshes for large-scale physical simulations.

Table 3.2: Characteristics of Selected Scientific Software Repositories

Metric	Trilinos	Mantid	AMReX
Stars	606	148	375
Contributors	681	236	418
Total Commits	348,208	155,183	77,800
Active Months	300	178	329
Files in Repository	163,398	69,849	36,806

As shown in Table 3.2, these projects demonstrate sustained maintenance and high activity, which ensures a large volume of PRs for statistical analysis. The selected repositories cross different scientific domains, programming language ecosystems, and project maturity levels, providing a representative sample of scientific software.

These projects collectively provide an ideal dataset for our study, covering diverse scientific domains including physics, chemistry, and numerical simulation. These characteristics make them particularly suitable for examining the impact of scientific knowledge on PR review processes, which we analyze in this chapter.

3.2.2 Data Collection and Preprocessing

Our goal was to collect a substantial dataset of merged PRs representative of recent development activity in each of the three selected repositories. We implemented a Python-based crawler using the GitHub REST API to systematically extract PR data.

The script was designed to collect up to 400 merged PRs per repository, which reflects our strategy of fetching the most recently updated PRs first until we either reached the 400 PR limit or had included all PRs that met our strict inclusion criteria, as detailed below. The target of 400 PRs per repository was set to ensure a substantial sample size for statistical analysis, while remaining feasible given API limitations and the manual annotation effort required to label each PR as scientific or non-scientific.

To ensure the scientific relevance of our dataset, we implemented a filtering process to exclude PRs that were exclusively related to generic software maintenance. During collection, our script automatically excluded a PR if its title contained keywords indicative of such non-scientific maintenance. The exclusion list was derived from an initial manual inspection of each project's PR history and included terms such as: "moving", "support", "deprecation", "debug", "typo", "fix", "remove", "update", "bug", "cleanup", "re-

lease", "refactor", "test", "version", "Windows", and "CI".

By removing this noise, we were left with a dataset enriched with PRs related to scientific features, algorithms, and performance improvements that were more likely to involve domain-specific knowledge.

For each qualifying PR, we extracted a comprehensive set of metadata to support both the review process analysis in Chapter 3 and the classification task presented in Chapter 4. The extracted variables are summarized in Table 3.3.

Table 3.3: Metadata Variables Extracted for Each PR

Variable	**Description**
`repository`, `pr_number`, `url`, `title`, `user`	Identifiers for each PR, including repository name, PR number, URL, title, and author.
`created_at`, `merged_at`	Creation and merge timestamps, used to measure the PR's duration.
`commits`	Total number of commits in the PR.
`commits_after_review`	Number of commits pushed after the first review comment.
`inline_comments`	Number of comments made directly on code lines.
`issue_comments`	Number of comments in the PR discussion thread.
`review_comments`	Number of formal review actions (e.g., "Approve", "Request Changes").
`total_discussion_comments`	Total number of discussion comments (inline, issue, and review combined).
`time_between_first_last_comment`	Time span (in seconds) between the first and last comment.
`time_to_merge`	Total time (in seconds) from PR creation to merge.
`requested_changes`	Number of "Request Changes" submissions.
`unique_reviewers`	Number of distinct reviewers who participated.
`avg_comment_resolution_sec`	Average time (in seconds) between a comment and the commit that addressed it.
`labels`	Labels applied by maintainers (e.g., "bug", "enhancement", "domain:physics").

3.2.3 Data Cleaning and Final Dataset

Through the application of strict exclusion criteria, we obtained a filtered dataset that is largely composed of scientific PRs, though some non-scientific contributions remain. The PR counts for each repository varied according to the density of recent, domain-relevant activity:

- **AMReX**: 338 PRs
- **Mantid**: 400 PRs
- **Trilinos**: 400 PRs

For each repository, we selected up to 400 PRs; however, AMReX contributed fewer PRs because fewer than 400 PRs met all selection criteria. This resulted in a total dataset of **1,138 PRs** for classification and analysis.

A final cleaning step was then applied to remove noise and ensure data quality:

- **Bot-Generated PRs**: PRs with titles containing the term "bot" were excluded, as these are automatically created and do not need contributions from software engineers or domain experts.
- **Generic Development PRs**: PRs with titles containing generic development keywords such as "chore", "docs", or "test" were excluded.

This cleaning step removed a total of **64 PRs**. The final PR counts for each repository were:

- **AMReX**: 355 PRs
- **Mantid**: 381 PRs
- **Trilinos**: 338 PRs

Resulting in a final dataset of **1,074 PRs** for classification and analysis.

In addition to the primary dataset described above, we constructed a second dataset using a complementary keyword-based filtering strategy to focus on software-oriented PRs.

Specifically, while the primary dataset excludes PRs associated with general software maintenance (e.g., "bug", "refactor") to emphasize domain-relevant contributions, the second dataset applies an inverse filtering approach by excluding PRs containing

domain-specific scientific keywords. All other data collection and filtering steps remain identical across the two datasets. In addition, both datasets exclude PRs with non-informative maintenance labels (e.g., “moving”, “typo”, “remove”, “cleanup”, “release”, “version”).

This symmetric design ensures that both datasets are constructed under consistent conditions while emphasizing different types of contributions. Due to the interdisciplinary nature of scientific software, neither dataset is perfectly separated, and both may contain a small proportion of mixed or ambiguous PRs. However, this proportion is limited and does not materially affect the overall analysis.

To ensure consistency, all PRs from both datasets are classified using the same binary labeling framework (SK=0/1), which forms the basis for downstream analysis.

3.3 Labeling Framework

This section defines the concepts and labeling criteria used to classify PRs based on their knowledge requirements. The goal of this framework is to provide a consistent and reproducible method for distinguishing between PRs that require scientific domain knowledge and those that primarily involve software engineering knowledge.

3.3.1 Definition of Scientific Knowledge and Software Knowledge

In the context of research software, contributions often require a combination of domain-specific understanding and software engineering expertise. To support systematic classification, we define these two types of knowledge as follows [23]:

- **Scientific Knowledge (SK)** refers to domain-specific understanding required to correctly interpret, implement, or validate scientific concepts within the software.
- **Software Knowledge (SW)** refers to general software engineering skills required to design, implement, and maintain code, independent of specific scientific domains.

3.3.2 Initial Multi-scale Classification Approach and Challenges

We initially designed a detailed classification system that rated both Scientific Knowledge (SK) and Software Knowledge (SW) on a 1–3 scale. This multi-scale approach

aimed to capture the nuance of scientific contributions, with a score of 3 indicating advanced innovation or design and a score of 1 indicating minimal scientific knowledge requirements.

However, applying this framework revealed significant reliability challenges. Three LLMs (ChatGPT-4o, Gemini 2.5 Pro, DeepSeek-R1) independently scored 400 PRs, resulting in low agreement levels. As shown in Table 3.4, inter-LLM agreement was slight ($\alpha = 0.143$), and agreement between LLMs and human annotators was only fair ($\alpha = 0.244$). The low reliability was caused by two primary factors: systematic scoring biases between models (e.g., DeepSeek assigned SK=3 to 54 PRs compared to ChatGPT's 28) and inherent ambiguity in distinguishing between SK-1, SK-2, and SK-3, which all indicated that knowledge was needed but at different levels of complexity.

Table 3.4: Initial 1-3 Scale Agreement Metrics (n=400 PRs)

Agreement Type	**Krippendorff's α**	**Interpretation**
Inter-LLM (All)	0.143	Slight agreement
Human-Human	0.369	Fair agreement
LLM-Human Consensus	0.244	Fair agreement

Given these reliability challenges, we concluded that the multi-scale system was too complex and ambiguous for consistent application by both humans and LLMs. We therefore transitioned to a simpler, more robust binary classification focused exclusively on the presence or absence of scientific knowledge.

3.3.3 Transition to Binary Classification Method

Given the reliability challenges of the multi-scale approach, we simplified our classification framework to improve consistency. This transition involved three key methodological changes:

1. **Trial Sample Selection**: We selected a stratified sample of 80 PRs from the first two repositories, Trilinos and Mantid, for initial testing and refinement of the new approach.

2. **Binary Classification Framework**: We shifted to a binary classification system focusing exclusively on the need for Scientific Knowledge (SK):

- **SK-0**: No scientific knowledge required for implementation or review.
- **SK-1**: Scientific knowledge required at any level of complexity.

3. **Collaborative Adjudication Process**: Researchers independently labeled all 80 PRs using the binary criteria, followed by structured consensus meetings to resolve disagreements and establish a ground truth.

3.3.4 Classification Criteria

The binary classification was guided by the following criteria:

- **Scientific Knowledge Required (SK-1)**: This classification applies to PRs involving modification or implementation of scientific logic (e.g., solvers, equations), selection of scientific models or parameters, or configuration of components requiring domain understanding.
- **No Scientific Knowledge Required (SK-0)**: This classification applies to PRs involving copying existing patterns, parameter plumbing without scientific reasoning, documentation changes, or tooling modifications.
- **Key Consideration**: PRs containing scientific terminology (e.g., "algorithm", "matrix") without actual modifications to scientific logic were classified as SK-0, as the changes required software engineering knowledge rather than domain expertise.

3.3.4.1 Binary Classification Examples

SK-0 Example: Parameter Configuration Without Scientific Innovation

```
URL: https://github.com/trilinos/Trilinos/pull/6535
Title: MueLu: Exposing aggregation algorithm via parameterlist
Body: Testing: RHEL6/OpenMP
```

Rationale: While mentioning "aggregation algorithm", this PR only exposes existing functionality through parameters without modifying scientific logic, requiring knowledge of the software's parameter system rather than scientific principles.
SK-0 Example: Pattern Replication

```
URL: https://github.com/trilinos/Trilinos/pull/13960
Title: MueLu: Add material vector information into multiphysics class
```

```
Body: This adds material property information for the individual
subblocks in the `MultiPhys` class.
```

Rationale: This PR introduces "material vectors" but simply duplicates existing coordinate-handling patterns without requiring materials science knowledge, demonstrating how scientific terminology alone does not indicate a need for scientific knowledge.

SK-1 Example: **Numerical Solver Semantics and Data Handling**

```
URL: https://github.com/trilinos/Trilinos/pull/7678
Title: Amesos2: Handle x and b copying more efficiently.
Body: The solver gets the x and b data from the adapter and it may be
directly assigned (no deep copy) if memory and types match. This PR
adds a flag bInitialize so the 'get x' call can inform the adapter
that initializing the data is not necessary. This means that when the
adapter does not match and has to allocate the data for x, it won't
deep copy its x values into the x space. That was a wasted copy before
since the solve was going to overwrite x anyways.
Also the get x or get b calls now receive a bDidAssign flag to tell
the solver if the data is pointing directly to the adapter memory
space. If x is directly assigned, there is no need to 'put x' after
the solver and this can now be skipped.
Also SuperLU and Klu2 may modify b during the solve under some
conditions. If the adapter directly assigned b, it's necessary to
deep_copy b first to avoid overwriting the adapters b data.
Previously, this deep_copy would always happen but now only when
necessary.
```

Rationale: This PR changes how the solver obtains and manages the solution vector `x` and right-hand-side vector `b` from the adapter. The modifications depend on whether the solver will overwrite `x` and on conditions under which underlying solvers (SuperLU and KLU2) may modify `b`. Therefore, solving them requires numerical algorithm knowledge.

SK-1 Example: **Implementation of a Domain-Specific Scientific Algorithm**

```
URL: https://github.com/mantidproject/mantid/pull/37753
Title: Add algorithm for calculating Wildes polarization efficiencies
```

`Body: Adds new algorithm PolarizationEfficienciesWildes that calculates the polarization efficiencies for a two-flipper instrument setup according to the Wildes 2006 paper. This implementation is based on the implementation that POLREF are currently using in their reduction code.`

Rationale: This PR implements a domain-specific scientific algorithm derived from the Wildes (2006) paper, clearly necessitating scientific domain expertise.

3.4 Annotation Process

This section describes the annotation process used to assign scientific knowledge labels to PRs. It outlines the procedures for human annotation, the consensus process for resolving disagreements, and the methods used to evaluate annotation reliability. These steps ensure the validity and consistency of the labels used in subsequent analysis and model evaluation.

3.4.1 Coding Guide Reference

Two human annotators independently classified each PR according to a defined coding guide. They were explicitly instructed to assign a binary label (SK=0 or SK=1) to each PR based on whether scientific knowledge was required to understand or validate the changes, using the provided definitions and examples as guidance. For example, PRs were labeled as follows:

- **SK = 0**: *MueLu: Add material vector information into multiphysics class*, which duplicated an existing coordinate-handling pattern without requiring domain reasoning.
- **SK = 1**: Implemented or configured a solver algorithm whose correct use depends on understanding physical model constraints.

Detailed examples are provided in Section 3.3.4.1. The annotation process evolved in two phases.

3.4.1.1 Initial Multi-Scale Phase

In the initial phase, annotators were provided with the following definitions to calibrate their understanding of scientific versus software knowledge:

Scientific Knowledge (SK)

- **1 (Low)**: No or minimal scientific concepts involved.
- **2 (Medium)**: Some scientific content, minor modeling, or algorithmic discussions.
- **3 (High)**: Heavily focused on scientific models, equations, or domain-specific algorithms.

Software Knowledge (SW)

- **1 (Low)**: No or minimal software engineering changes.
- **2 (Medium)**: Some refactoring, performance tuning, or architectural decisions.
- **3 (High)**: Major software improvements, code structure changes, or optimizations.

Under this scheme, each PR received two independent ratings: one for Scientific Knowledge (SK: 1–3) and one for Software Knowledge (SW: 1–3).

3.4.1.2 Binary Classification Guidance

Due to reliability challenges with the multi-scale approach, the framework was simplified to a binary classification focused exclusively on Scientific Knowledge. The annotators were instructed as follows:

- **0 (No Need)**: PRs that can be fully understood and implemented without any domain-specific scientific knowledge.
- **1 (Need)**: PRs that require an understanding of domain sciences (e.g., physics, materials science, computational chemistry) to implement or review correctly.

For PRs that were labeled using the original scale, the Scientific Knowledge ratings were simplified by converting all values of **1** into **0**, and **2** and **3** into **1**, where **1** indicates that scientific knowledge is required. The Software Knowledge ratings were not used in the final analysis, since the classification only focused on whether scientific knowledge was needed. For PRs labeled after this change, the binary scale was used directly.

3.4.2 Human Annotation

Two human annotators, Rater 1 and Rater 2, independently classified all 1,074 PRs using the binary classification framework (see Section 3.3.3), following the guidance in Section 3.4.1.2.

Annotators reviewed the full context of each PR, including its title, description, and linked issues, the list of modified files and associated code diffs, and the labels and metadata provided by maintainers. When needed, they also accessed the PR through the provided URL to examine the same information in more detail.

Rater 1 labeled 654 PRs (60.9%) as requiring scientific knowledge, while Rater 2 labeled 513 PRs (47.8%) as such. The two annotators agreed on 906 PRs (84.4%), yielding a Krippendorff's α of 0.686, which is indicative of acceptable reliability for tentative conclusions [24]. Disagreements were most frequent in borderline cases involving exposure or configuration of existing scientific components, where the necessity of domain expertise was open to interpretation.

3.4.3 Consensus Process

To ensure the reliability of the classification labels, a consensus process was applied to resolve disagreements between the two human annotators.

Each annotator independently assigned a binary label (SK=0 or SK=1) to all PRs following the predefined coding guide. After the initial labeling phase, the annotations were compared to identify cases of disagreement.

For PRs where the annotators assigned different labels, a joint review process was conducted. During this process, both annotators revisited the PR content, including the title, description, changed files, and code diffs, and discussed their interpretations based on the classification criteria.

The discussion focused on whether evaluating the correctness of the PR required scientific domain knowledge. Annotators referred to the coding guide and representative examples to ensure consistent reasoning. Through this process, disagreements between annotators were retained and reported as "Undecided". However, descriptive statistics and subsequent analyses requiring clear classification are computed using only PRs with agreed labels (SK=0 or SK=1).

3.4.4 Agreement Analysis Method: Krippendorff's α

To quantify the reliability of agreement between independent raters, we utilize Krippendorff's alpha (α), a robust statistical measure for inter-rater reliability [24]. This metric is particularly suitable for our study as it can handle missing data and accounts for the possibility of agreement occurring by chance. Krippendorff's α accommodates multiple raters and various data types (nominal, ordinal, interval), making it versatile for diverse annotation tasks.

An α value of 1 indicates perfect agreement, 0 indicates agreement equivalent to chance, and negative values mean the raters' observed agreement is less than the agreement expected by random guessing. Following established interpretation guidelines [24], we consider $\alpha \geq 0.823$ to represent good agreement, while $\alpha \geq 0.667$ is acceptable for tentative conclusions; values below 0.667 indicate substantial unreliability.

Our choice of Krippendorff's α aligns with recent research applying this metric to evaluate agreement between human annotators and LLMs in software engineering tasks [1]. This method effectively captures the context-dependent judgments required when evaluating software artifacts such as PR classifications.

3.4.5 Human Annotation Reliability

The transition from a multi-scale to a binary classification system improved the reliability carried through to the full dataset. The two human annotators demonstrated strong agreement in their classifications. Rater 1 labeled 654 PRs (60.9%) as requiring scientific knowledge, while Rater 2 labeled 513 PRs (47.8%) as such. Despite these different absolute counts, the annotators agreed on 906 of the 1,074 PRs (84.4%).

Krippendorff's α for human-human agreement was 0.686, which falls within the acceptable range for tentative conclusions according to established guidelines [24]. Most disagreements came from cases where PRs added or exposed configuration options for existing scientific features. In these situations, it was not always clear whether scientific expertise was needed, so annotators sometimes evaluated the knowledge requirement differently.

This relatively high level of agreement supports the reliability of the binary scientific knowledge framework after the adjudication process established during the preliminary trial phase. It also provides a reliable benchmark for evaluating LLM performance against human judgment.

3.5 Descriptive Statistics

To evaluate the proposed hypothesis, we first conduct descriptive statistical analysis using a set of basic metrics, including the number of files changed, the number of unique reviewers, and the total number of lines changed.

The descriptive analysis reveals systematic differences in PR characteristics between scientific (SK=1) and non-scientific (SK=0) PRs based on human annotations. In our labeling framework, each PR is explicitly assigned a binary label: SK=1 if scientific knowledge is required to understand or validate the contribution, and SK=0 if it is not. Annotators were required to make a direct choice between these two labels for every PR.

Agreed labels refer to PRs where both annotators independently assigned the same binary classification (SK=0 or SK=1), while disagreements are reported as "Undecided". These PRs are excluded from the computation of median statistics, which are calculated using only PRs with agreed labels.

Table 3.5 presents the distribution of PR classifications and the labeling behavior of each human annotator across repositories.

Table 3.5: Distribution of PR classifications and human annotation results across repositories

Repository	**Category**	**Trilinos**	**Mantid**	**AMReX**
Counts	Agreed SK=0	189	61	156
	Agreed SK=1	124	259	117
	Undecided	42	61	65
Annotators (SK=1)	Rater 1	162 (45.6%)	314 (82.4%)	178 (52.7%)
	Rater 2	127 (35.8%)	265 (69.6%)	121 (35.8%)

Table 3.6 summarizes the median values for key baseline metrics.

Table 3.6: Baseline Characteristics by Scientific Knowledge Requirement (Median)

Repository	Metric	SK=0	SK=1	Diff.
Trilinos	Files Changed	3.0	7.0	+4.0
	Unique Reviewers	1.0	2.0	+1.0
	Total Lines Changed	47.5	53.0	+5.5
AMReX	Files Changed	2.0	3.0	+1.0
	Unique Reviewers	1.0	2.0	+1.0
	Total Lines Changed	24.0	61.5	+37.5
Mantid	Files Changed	4.0	7.0	+3.0
	Unique Reviewers	1.0	2.0	+1.0
	Total Lines Changed	109.5	122.0	+12.5

The results indicate that scientific PRs differ measurably from software-specific PRs, exhibiting larger code changes, greater reviewer involvement, and broader modification scope.

However, these metrics capture only limited aspects of the review process. A more comprehensive analysis is therefore required to further evaluate the hypothesis.

3.6 Rationale for Metric Selection

Our goal is to understand whether PRs that require scientific knowledge differ from those that do not in terms of how they are reviewed and discussed. Scientific PRs often involve complex algorithms, specialized data structures, or domain-specific assumptions. These aspects may demand more reviewer effort and collaboration compared to routine contributions.

To explore this, we selected six process-oriented metrics that reflect different facets of review effort and coordination. Together, these metrics provide a broad view of how much discussion, iteration, and scrutiny a PR receives. These metrics were selected based on their relevance to review dynamics, collaboration, and procedural complexity:

1. **Time to Merge (days)**: Duration from PR creation to merge. Calculated by dividing `time_to_merge` (in seconds) by 86,400.

2. **Number of Unique Reviewers**: Indicates the diversity of expertise involved. Count of distinct users who participated in the review process through comments or review submissions (`unique_reviewers`).

3. **Total Discussion Comments**: Sum of inline code review comments, issue comments, and review comments. Combined total of three comment types:

   - `inline_comments`: Line-specific comments in the PR diff.
   - `issue_comments`: General comments in the PR timeline.
   - `review_comments`: Summary comments included in formal reviews.

   The final value is the sum of all three.

4. **Commits After First Review**: Number of commits pushed after the first review comment was received (`commits_after_review`).

5. **Time Between First and Last Comments (days)**: Duration of the review discussion. Elapsed time between the first and last PR comments, computed as `time_between_first_`

6. **Requested Changes**: Count of review requests for changes before approval, counted from review events labeled `CHANGES_REQUESTED`..

We also considered other commonly used measures, such as the number of files changed or lines of code modified. While these metrics quantify the size of a change, they do not reliably reflect review effort, particularly in research software, where a small modification may require careful validation of scientific assumptions, models, or numerical behavior, whereas a larger change may consist of repetitive or well-understood refactoring. Merge conflicts were not considered because they are relatively rare and often resolved outside the review process. Similarly, labels and tags were excluded because they vary between projects and cannot be compared in a consistent manner across repositories.

Other metrics could further illuminate the differences between scientific-specific PRs and software-specific PRs. For instance, the average reviewer experience level could indicate if scientific PRs require more senior oversight. However, since most of the PRs we used cannot find the reviewers' seniority within the project, this metric is not feasible for our analysis. Furthermore, the proportion of discussion containing questions or citations might measure the need for clarification or external justification. Tracking the number of distinct review cycles (periods of activity separated by inactivity) could

quantify iterative validation. These metrics were not selected for this analysis primarily due to the significant additional data extraction effort required and the challenge of defining them consistently across projects.

3.7 Statistical Analysis

3.7.1 Descriptive Analysis and Distribution Comparison

Before applying the logarithmic transformation, we first examine the distributions of each metric using histograms. In addition, we overlay maximum likelihood estimated (MLE) Gaussian mixture density curves to provide a smooth, model-based approximation of the histogram shapes. The following histograms (Figure 3.3– 3.8) provide a visual comparison of the six core metrics between scientific (SK=1) and non-scientific (SK=0) PRs, plotted on a log scale.

3.7.1.1 Commits After First Review

Figure 3.3 shows the distributions of commits after the first review across repositories. Software-specific PRs are heavily concentrated at zero or one commit, indicating limited post-review iteration. In contrast, scientific-specific PRs exhibit higher values and longer tails, reflecting more frequent revision cycles after initial feedback and suggesting a more iterative review process.

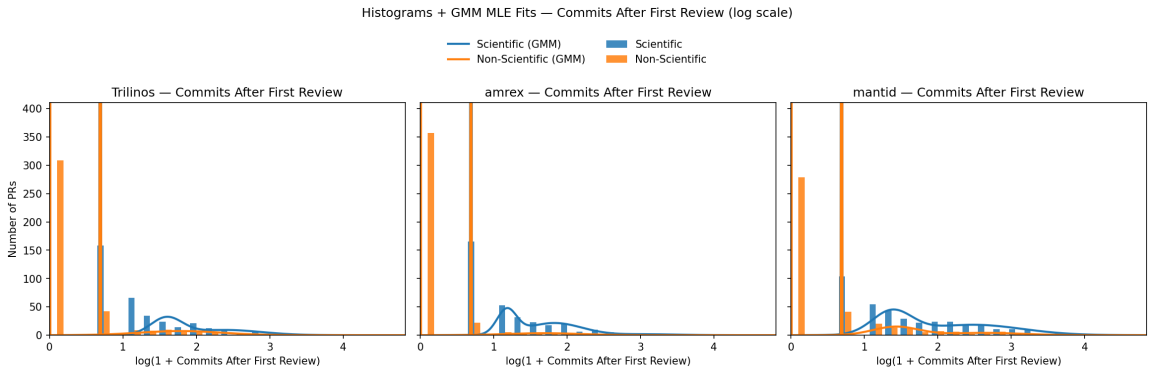

Figure 3.3: Histograms of commits after first review across repositories (log scale).

3.7.1.2 Number of Unique Reviewers

As illustrated in Figure 3.4, the distributions of the number of unique reviewers (log scale) are shifted toward higher values for scientific-specific PRs compared to software-specific PRs. While software-specific PRs are more concentrated at lower reviewer counts, scientific PRs more frequently involve larger reviewer sets, indicating broader participation. This pattern is consistent with the need to involve multiple contributors with complementary expertise when evaluating domain-specific changes.

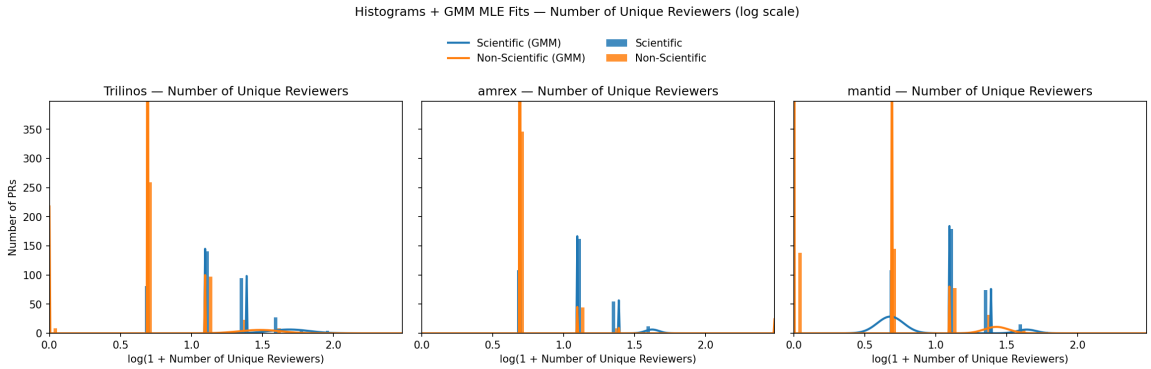

Figure 3.4: Histograms of number of unique reviewers across repositories (log scale).

3.7.1.3 Requested Changes

Figure 3.5 presents the distributions of requested changes using a log transformation. Although most PRs in both categories receive no change requests, scientific-specific PRs are more likely to appear in higher bins, particularly in Mantid. This suggests that scientific PRs are more likely to undergo multiple rounds of revision before approval.

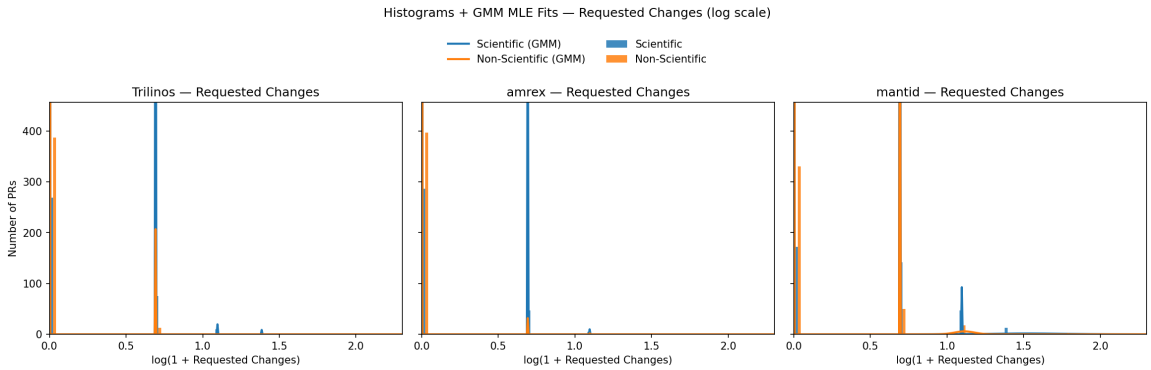

Figure 3.5: Histograms of requested changes across repositories (log scale).

3.7.1.4 Time Between First and Last Comments (days)

Figure 3.6 highlights differences in review duration on a log scale. Software-specific PRs are often concentrated at shorter discussion spans, whereas scientific-specific PRs exhibit wider distributions with longer tails. This indicates that reviews of scientific PRs are more likely to involve sustained interaction over time rather than brief exchanges.

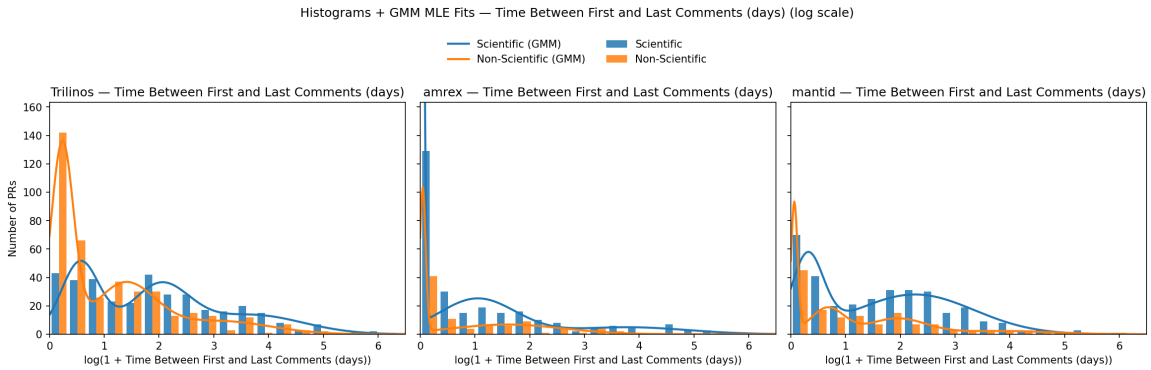

Figure 3.6: Histograms of the time between first and last comments across repositories (log scale).

3.7.1.5 Time to Merge (days)

Figure 3.7 shows the distributions of time to merge (log scale) across repositories. Scientific-specific PRs exhibit a higher concentration at short merge times, indicating that many are merged quickly, while they also display a heavier right tail than software-specific PRs. This pattern suggests greater variability in merge times for scientific PRs, with a subset undergoing substantially longer review cycles.

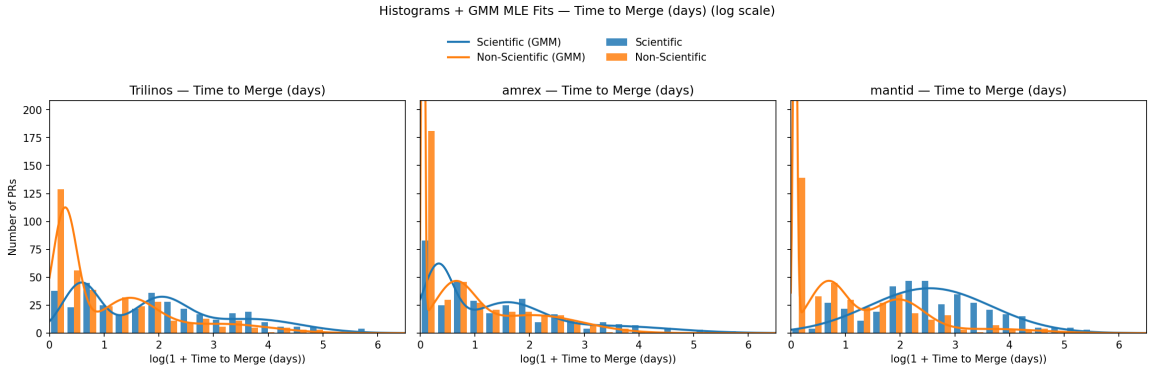

Figure 3.7: Histograms of time to merge across repositories (log scale).

3.7.1.6 Total Discussion Comments

As shown in Figure 3.8, scientific-specific PRs tend to generate more extensive discussion than software-specific PRs. The distributions for scientific PRs are shifted toward higher values and exhibit heavier right tails on the log scale, indicating a greater propensity for multifaceted review conversations.

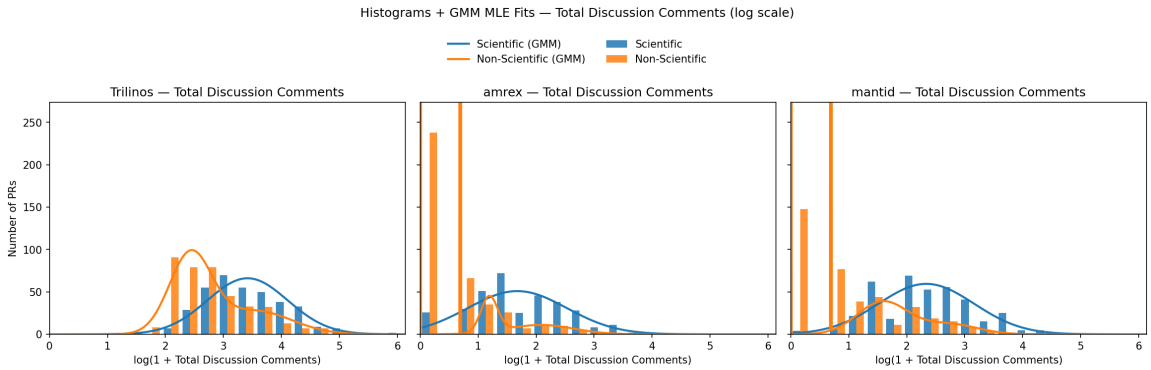

Figure 3.8: Histograms of total discussion comments across repositories (log scale).

These results provide an initial indication of systematic differences between scientific-specific and software-specific PRs.

3.7.2 Welch's T-Tests with Bonferroni Correction

We conducted Welch's two-sample t-tests with Bonferroni correction across the six process metrics to assess whether the mean values differ significantly between the two groups. All tests were performed on log-transformed values ($\log(1+x)$) to reduce skewness. Since six hypotheses were tested per repository, we applied a Bonferroni correction to control for family-wise error. This yields an adjusted significance threshold of $\alpha = 0.05/6 \approx 0.0083$.

Decision criterion. For each metric, the null hypothesis (H_0) states that the mean values of scientific (SK=1) and non-scientific (SK=0) PRs are equal. The alternative hypothesis (H_1) is that they differ. The Welch's t-test produces a p-value, which represents the probability of observing a difference as extreme as the one in the data if H_0 were true. We then multiply the raw p-value by the number of tests (six per repository) to obtain the Bonferroni-adjusted value, p_{Bonf}. If $p_{\text{Bonf}} < 0.05$ (equivalently, raw $p < 0.0083$), the metric is declared statistically significant. This correction ensures that the probability of any false positive across all six tests remains below 5%.

Table 3.7: Welch's t-tests with Bonferroni correction comparing scientific (SK=1) and non-scientific (SK=0) PRs across repositories.

Repository	Metric	Mean (SK=0)	Mean (SK=1)	p_{Bonf}	Sig.
Trilinos	Time to Merge (days)	6.57	19.99	3.64×10^{-18}	Yes
Trilinos	Number of Unique Reviewers	1.44	2.30	2.38×10^{-37}	Yes
Trilinos	Total Discussion Comments	21.74	38.90	5.33×10^{-29}	Yes
Trilinos	Commits After First Review	0.86	3.45	5.72×10^{-71}	Yes
Trilinos	Time Between First and Last Comments (days)	6.56	20.02	6.20×10^{-18}	Yes
Trilinos	Requested Changes	0.03	0.28	1.04×10^{-15}	Yes
amrex	Time to Merge (days)	4.15	13.38	2.57×10^{-9}	Yes
amrex	Number of Unique Reviewers	1.18	1.92	1.11×10^{-47}	Yes
amrex	Total Discussion Comments	1.49	6.68	5.06×10^{-67}	Yes
amrex	Commits After First Review	0.33	3.15	8.35×10^{-102}	Yes
amrex	Time Between First and Last Comments (days)	4.41	12.61	0.999	No
amrex	Requested Changes	0.01	0.17	4.71×10^{-11}	Yes
mantid	Time to Merge (days)	6.42	23.28	2.23×10^{-61}	Yes
mantid	Number of Unique Reviewers	1.06	2.00	3.15×10^{-47}	Yes
mantid	Total Discussion Comments	3.46	13.50	2.91×10^{-80}	Yes
mantid	Commits After First Review	1.50	6.90	6.69×10^{-72}	Yes
mantid	Time Between First and Last Comments (days)	10.56	12.72	0.022	No
mantid	Requested Changes	0.23	0.82	5.09×10^{-26}	Yes

Significance criterion: $p_{\text{Bonf}} < 0.05$ (equivalently, raw $p < 0.0083$ for six tests per repo).

Interpretation. Across all three repositories, scientific PRs consistently require more reviewers, more discussion, and more *commits after first review* than software-specific PRs. In `Trilinos` and `Mantid`, every metric shows a highly significant difference. In `AMReX`, five metrics are significant, while *Time Between First and Last Comments* does not show a reliable difference ($p = 0.999$ after correction). These results strengthen the claim that PRs requiring scientific knowledge involve more complex and iterative review processes.

3.7.3 Regression Test

The t-test analysis does not control for potential confounding variables across repositories. We thus created multiple regression models controlling for PR size, complexity, and repository effects. The general model specification is:

$$Y = \beta_0 + \beta_1 \cdot SK + \beta_2 \cdot Files\_Changed + \beta_3 \cdot Lines\_Changed + \beta_4 \cdot NReviewers + \beta_5 \cdot Repo + \epsilon \quad (3.1)$$

Where

- Y is the dependent variable representing a review outcome (e.g., Time to Merge, Total Discussion Comments).
- SK is the binary Scientific Knowledge Indicator (1 for scientific-specific PRs, 0 for software-specific PRs).
- $Files\_Changed$ is the number of files changed.
- $Lines\_Changed$ is the total lines changed (in hundreds).
- $NReviewers$ is the number of unique reviewers.
- $Repo$ represents repository fixed effects (its own culture, size, complexity, and review practices).
- β_1, β_2, β_3, β_4, and β_5 are regression coefficients corresponding to the independent variables in the model.
- ϵ is the error term.

Excluded variables. While the regression model controls for PR size, reviewer participation, and repository effects, several potentially relevant variables are not included due to data availability and consistency constraints. These include reviewer seniority or expertise, contributor experience within the project, geographic distribution and time zones, developer workload or availability, etc. In addition, factors, such as review queue length, project deadlines, and external coordination requirements (e.g., dependencies on experimental results or scientific validation), are not captured in the GitHub metadata used in this study. Although these factors may influence review dynamics, they are difficult to obtain and operationalize consistently across repositories.

3.7.3.1 Regression Test Result

Table 3.8 presents the regression estimates quantifying how the presence of Scientific Knowledge (SK = 1) affects various PR outcomes after controlling for repository effects and PR complexity. Across all models, the SK coefficient is positive and highly significant ($p < 0.001$), indicating that PRs requiring scientific knowledge consistently involve greater effort and coordination than non-scientific ones.

Table 3.8: Regression results for the effect of Scientific Knowledge (SK) on PR metrics

Outcome Variable (Y)	SK Coefficient (β)	% Change (exp(β)-1)
Log(Time to Merge)	0.512	+66.9%
Number of Unique Reviewers (NB)	0.492	+63.5%
Total Discussion Comments (NB)	0.696	+100.7%
Commits After First Review (NB)	1.399	+305.1%
Log(Time Between First & Last Comments)	0.234	+26.4%
Requested Changes (NB)	1.399	+305.1%

These results support the t-tests from the previous section. They confirm the notion that domain knowledge, in the form of science knowledge, requires more complex software processes.

3.8 Hypothesis Evaluation

Our hypothesis is supported by the results from both our group comparison tests and multiple regression analyses.

3.8.1 Evaluation Across Statistical Tests

The Welch's t-tests with Bonferroni correction provide direct group comparisons. Across all three repositories, five of the six metrics were statistically significant ($p_{Bonf} < 0.05$) for scientific PRs: *Time to Merge*, *Number of Unique Reviewers*, *Total Discussion Comments*, *Commits After First Review*, and *Requested Changes*. All observed differences were consistent with the hypothesized direction, indicating that scientific PRs exhibited longer durations, greater participant involvement, and more iterative discussion. The one exception was the metric *Time Between First and Last Comments*, which showed a significant difference only in `Trilinos` and not in `AMReX` or `Mantid`. This pattern suggests that while the volume of discussion is consistently higher for scientific PRs, the timespan of that discussion may be more contingent on project-specific workflows, resulting in only partial support for the hypothesis on this specific temporal characteristic.

3.8.2 Evaluation Across Regression Models

The multiple regression analysis controls for potential confounds like PR size and repository effects. In every regression model, the Scientific Knowledge (SK) indicator was

a positive and highly significant predictor ($p < 0.001$) of increased review effort and complexity. The impact of requiring scientific knowledge on the review process is large. Scientific PRs are associated with increases ranging from about 26% longer discussion duration to more than 300% more post-review revisions and requested changes. This confirms that the observed differences in these metrics are directly associated with the requirement for scientific knowledge, rather than differences in PR size, as measured by files changed and lines modified.

3.9 Discussion

This chapter examined how knowledge requirements influence PR review processes in research software by comparing PRs that require scientific knowledge (SK=1) with those that do not (SK=0).

3.9.1 Review Effort and Iteration

Our analysis shows that the observed differences between PR types are not merely a function of code change size. The effects remain statistically significant even after controlling for the number of files changed, lines changed, and repository-specific factors through multiple regression.

The overall pattern is one of increased effort, iteration, and scrutiny. Scientific PRs attract significantly more unique reviewers, suggesting that domain expertise is often distributed and requires input from multiple contributors. The near-doubling of discussion comments and the more than 300% increase in commits after the first review and requested changes indicate a highly iterative review process.

Reviewers of scientific PRs engage in more extensive discussion and require more revisions from contributors. This likely reflects the complexity of verifying scientific correctness in algorithms, models, and data transformations. As a result, scientific PRs take substantially longer to merge, with an average increase of 67%.

3.9.2 Review Intensity vs. Review Duration

The metric *Time Between First and Last Comments* provides an important nuance. While scientific PRs consistently generate higher volumes of interaction, they do not always lead to longer discussion durations.

This suggests that review intensity and review duration are influenced by different factors. While knowledge requirements drive the amount of discussion and iteration, the overall timeline appears to be shaped more strongly by project-specific workflows, reviewer availability, and community practices. This observation is consistent with prior work [26], which shows that temporal metrics often stabilize within project-specific norms.

3.9.3 The Use of a Binary Classification Framework

A key finding of this study is that transitioning from a multi-scale classification to a binary scientific knowledge (SK) framework substantially improved inter-rater reliability. The original 1–3 scale for scientific knowledge (SK) and software knowledge (SW) introduced ambiguity that annotators could not apply consistently, resulting in low agreement.

Reducing the task to a binary decision simplified the classification process. This simplification, combined with the consensus-based adjudication procedure, increased human–human agreement to $\alpha = 0.682$, a level considered acceptable for preliminary conclusions.

This simplification, however, represents a trade-off. While effective for distinguishing broad categories of PRs, it collapses a spectrum of scientific complexity into a single label. As a result, differences in the type or depth of scientific knowledge required are not captured.

A potential direction for future work is a hierarchical or multi-stage classification framework. An initial binary classification could be followed by a more fine-grained categorization of scientific knowledge types (e.g., numerical methods, physical modeling, data interpretation) or levels of complexity. This would enable a more detailed understanding of knowledge requirements while maintaining the reliability of the initial classification.

3.9.4 Implications for Research Software Collaboration

These findings highlight that not all PRs impose the same review burden. Contributions requiring scientific knowledge consistently demand more coordination, discussion, and revision effort.

This suggests that knowledge requirements fundamentally shape review dynamics in research software development. Scientific PRs involve deeper validation processes

and interdisciplinary reasoning, which naturally lead to more complex and iterative reviews. Recognizing these differences is important for understanding collaboration patterns and improving the evaluation of contribution processes in research software projects.

3.10 Limitations

Several limitations should be considered when interpreting the results of this analysis.

First, the construction of the datasets relies on keyword-based filtering strategies, which may introduce selection bias. Although the symmetric filtering approach enriches for different types of contributions, it does not produce perfectly pure datasets. Some PRs may still contain mixed characteristics, and relevant PRs may be excluded if they use non-standard or project-specific terminology.

Second, the classification of PRs into scientific (SK=1) and non-scientific (SK=0) categories depends on human annotation. While a consensus process was used to improve consistency, some cases remain inherently ambiguous due to the interdisciplinary nature of research software. As a result, the ground truth labels may still contain some uncertainty.

Third, the analysis is based on observable PR metadata and discussion activity. While metrics such as comments, reviewer counts, and commits capture aspects of review effort, they do not fully reflect the cognitive complexity or depth of reasoning involved in reviewing scientific contributions.

Fourth, time-based metrics such as *time-to-merge* have inherent limitations. They are influenced by social and organizational factors, including contributor availability, project workflows, and review policies. As a result, *time-to-merge* cannot be interpreted as a direct measure of contribution complexity.

3.10.1 Use of *Time-to-Merge* as a Metric

While *time-to-merge* is a commonly used proxy for contribution complexity or review friction, it has several limitations, as shown in Table 3.9. These limitations are not specific to scientific software and apply broadly to PR–based development. However, they are particularly substantial in scientific software projects, where review timelines are more strongly influenced by factors such as contributor availability, interdisciplinary coordination, and the need for domain-specific validation. As a result, *time-to-merge*

must be complemented by additional process metrics to accurately characterize review effort and contribution complexity in research software.

Table 3.9: Limitations of *Time-to-Merge* Metrics

Limitation	Explanation
Affected by social or organizational factors	Merge latency may reflect time zones, team load, or project policies (e.g., number of reviewers).
Does not reflect pre-PR work	Contributors may have spent considerable time preparing before the PR.
Does not reveal knowledge type	Delay cannot be attributed to SK/SW misalignment or simply busy schedules.
Ignores content of discussion	A long merge time could stem from trivial conversations, not substantial review.

To mitigate these issues, we included additional fields like `commits_after_review`, `requested_changes`, and `avg_comment_resolution_sec` to more accurately infer PR complexity and response dynamics.

Finally, the scope of this study is limited to three large, actively maintained research software repositories, primarily in physics and numerical computing. These projects may not fully represent other domains, programming languages, or project structures. Further validation across a broader range of research software systems is required to assess the generalizability of the findings.

3.11 Conclusion

This chapter investigated how knowledge requirements influence PR review processes in research software by comparing scientific-specific PRs and software-specific PRs.

The results show clear and consistent differences between the two types of contributions. PRs that require scientific knowledge involve significantly more reviewers, generate substantially more discussion, and undergo more iterative revisions after the first review. These differences remain statistically significant even after controlling for code change size and repository-specific factors, indicating that they are driven by knowledge requirements rather than implementation scale.

In particular, scientific PRs require, on average, 67% more time to merge, involve 64% more unique reviewers, and produce over 100% more discussion comments. They also

exhibit a substantially higher number of commits after the first review and requested changes, reflecting a more iterative and collaborative review process. These findings demonstrate that contributions requiring scientific knowledge impose greater review effort, coordination, and validation.

At the same time, not all aspects of the review process are equally affected. While scientific PRs consistently generate higher volumes of interaction, temporal metrics such as the duration of discussion are influenced more strongly by project-specific workflows and organizational factors. This highlights the importance of considering multiple metrics when analyzing review processes.

Overall, these findings address ***RQ1*** by demonstrating that scientific-specific and software-specific PRs exhibit systematically different review characteristics in terms of effort, interaction, and iteration.

Building on these findings, Chapter 4 investigates whether LLMs can reliably classify PRs based on their scientific knowledge requirements. This enables the automated identification of PR types used in the analysis presented in this chapter.

Chapter 4

Classification Models

4.1 Introduction

Our ***RQ1*** results showed that PRs requiring scientific knowledge differ systematically from those that do not in their review processes. In this chapter, we investigate ***RQ2:*** *How can LLMs be leveraged to automatically classify PRs in research software into scientific versus software-oriented contributions?*

Automating this classification is a critical step towards improving PR triage in research software, where contributions blend domain expertise and software engineering. Prior work has highlighted the challenges of this task due to the hybrid nature of contributions [23, 29]. Our approach combines structured metadata extraction with an LLM-based classification framework, guided by the PRIMES model [9].

4.2 PRIMES Framework Implementation

Our classification methodology follows a systematic integration of automated and human evaluation, guided by the PRIMES framework, Prompt Refinement and Insights for Mining Empirical Software repositories [9]. This framework provides a structured approach for using LLMs in repository mining studies, emphasizing iterative prompt refinement, multi-model evaluation, and rigorous validation to ensure reliable and reproducible results. We adapted this framework as follows:

1. **Multi-LLM Scoring**: Three LLMs were selected to independently classify each PR: ChatGPT-4o from OpenAI, Gemini 2.5 Pro from Google, and DeepSeek-R1 from DeepSeek AI. This selection was based on a strategy of architectural and provider

diversity to mitigate individual model biases and to assess the generalizability of our prompts across different models. Using multiple LLMs reduces the risk of model-specific bias and enables a more robust comparison of classification consistency. Each model was provided with structured prompts to generate binary scientific knowledge ratings.

2. **Human Annotation and Consensus**: Human annotation and the consensus-building procedure follow the methodology described in Section 3.4. As described in Section 3.4, two human annotators independently classified all PRs using the binary scientific knowledge framework, and disagreements were identified and retained as "Undecided" labels, with agreed labels used as the reference baseline for evaluation.

3. **LLM Validation and Prompt Refinement**: The human labels were used as a benchmark to validate the LLM outputs. We measured agreement among the three models, referred to as inter-LLM agreement. In cases of low inter-LLM agreement, we audited the performance of individual models to identify systematic errors. For models showing lower alignment with the human labels, we performed iterative prompt refinement to improve their performance.

4. **Held-Back Data**: To support iterative prompt testing and refinement without biasing the final evaluation, we reserved a stratified random sample of 120 PRs from the dataset described in Section 3.2, with 40 PRs from each repository. These PRs were used only for prompt development and were excluded from the final analysis.

4.3 LLM-Based Classification

4.3.1 Prompt Design

The prompt design aims to operationalize the binary scientific knowledge (SK) classification framework defined in Section 3.3 into a structured decision-making process that can be consistently applied by LLMs. The objective is to ensure that LLM-based classifications align with the same criteria used during human annotation.

Each prompt is constructed to mirror the information available to human annotators, including PR metadata, discussion context, and code changes, ensuring consistency between human and LLM classification.

To promote consistent and reproducible reasoning, the prompt explicitly structures the decision process into a sequence of evaluation steps. Specifically, the model is encouraged to:

1. Identify the primary purpose of the PR (e.g., bug fix, refactoring, algorithm implementation).
2. Examine whether the changes involve scientific logic, models, or domain-specific computations.
3. Distinguish between superficial use of scientific terminology and actual reliance on scientific reasoning.
4. Assign a binary label (SK=1 or SK=0) based on whether domain-specific scientific knowledge is required to understand or validate the changes.

To reduce ambiguity, the prompts are designed to provide clear instructions and encourage consistent reasoning across different cases. The classification focuses on whether a PR requires scientific reasoning, rather than relying solely on the presence of domain-specific terminology.

This design ensures that LLM classifications remain consistent with the operational definition of scientific knowledge used in human annotation, enabling direct and reproducible comparison across models.

4.3.2 Prompt Evolution

The evolution of our prompts targeted specific failure modes observed during testing (see Appendix A.1):

1. **Version 1 to Version 2**: Transitioned from a complex 1–3 scale for both Scientific Knowledge (SK) and Software Knowledge (SW) to a simple binary SK (0/1) classification. This resolved fundamental scale ambiguity and improved overall agreement.
2. **Version 2 to Version 3**: Added code diffs and changed file lists to provide necessary implementation context. This helped address false positives but introduced new challenges; inter-LLM agreement decreased by 37%, primarily due to ChatGPT's significant performance drop when processing code snippets.

3. **Version 3 to Version 4**: Included clear instructions to ignore scientific terminology when it appeared only in configuration settings or parameter adjustments. Such cases require software engineering knowledge rather than domain science expertise. This refinement significantly improved precision for Gemini and DeepSeek.

These results demonstrate that prompt effectiveness is highly model-dependent, and that improvements in contextual information or instruction specificity do not uniformly translate into better performance across LLM architectures.

4.3.2.1 Version Performance and Rationale

Version 1: Multi-scale Classification (Appendix A.1.1) The original prompt used a 1-3 scale for both SK and SW. This approach showed the poorest agreement ($\alpha = 0.143$) due to scale ambiguity and systematic biases between models. These results validated our decision to transition to a binary classification system.
Version 2: Binary Classification Foundation (Appendix A.1.2) As described in Section 3.3.3, the transition from a multi-scale classification scheme to a binary SK (0/1) formulation significantly improved agreement. The shift to binary SK labeling (0/1) significantly improved agreement ($\alpha = 0.421$). However, limitations remained as LLMs frequently over-classified PRs with titles containing scientific terminology such as "algorithm" or "solver" as SK=1, even when no actual scientific reasoning was required.
Version 3: Adding Code Context (Appendix A.1.3) The addition of code diffs and changed files aimed to provide implementation context. While Gemini showed improved human alignment ($\alpha = 0.304$), overall agreement decreased ($\alpha = 0.264$). ChatGPT's performance dropped sharply ($\alpha = 0.127$) as it struggled with code snippets. This showed architecture differences in how models process technical information.
Version 4: Model-Specific Refinements (Appendix A.1.4) Based on observed failure patterns, we developed customized versions for each LLM. For Gemini, we added explicit instructions to ignore scientific terms present only in titles or descriptions when they were not relevant to the code changes. For DeepSeek, we enhanced context handling with clearer examples distinguishing scientific innovation from routine software changes. ChatGPT's V4 prompt lacked sufficient ambiguity guidance, leading to excessive risk aversion and under-classification.

Table 4.1: Prompt Performance Evolution Across Versions (80 PR Sample). Final column reports Krippendorf α for given LLM with human label.

LLM	Ver.	# Sci PRs	Prec.	Recall	F1	α
ChatGPT-4o	V2	47	0.49	0.78	0.60	0.446
ChatGPT-4o	V3	52	0.48	0.76	0.59	0.316
ChatGPT-4o	V4	28	0.71	0.61	0.66	0.217
Gemini 2.5 Pro	V2	47	0.55	0.78	0.64	0.395
Gemini 2.5 Pro	V3	47	0.55	0.78	0.64	0.395
Gemini 2.5 Pro	V4	33	0.67	0.67	0.67	0.406
DeepSeek-R1	V2	54	0.52	0.85	0.64	0.434
DeepSeek-R1	V3	54	0.52	**0.85**	0.64	0.434
DeepSeek-R1	V4	31	**0.74**	0.71	**0.72**	**0.647**

4.3.3 Final Model Selection

Table 4.2 summarizes the final prompt selection for each LLM based on performance across the 80 PR samples. DeepSeek achieved the highest agreement with human raters, followed by Gemini and ChatGPT.

Table 4.2: Final prompt selection and performance metrics (80 PR sample)

LLM	Prompt Version	Scientific PRs	Agreement with Humans	Krippendorff's α
ChatGPT-4o	V2	27	66.3%	0.464
Gemini 2.5 Pro	V4	35	65.0%	0.406
DeepSeek-R1	V4	29	86.3%	0.647

Figure 4.1 provides a comparison of the final prompt performance for all three LLMs, which shows Krippendorff's α values computed separately against each human rater. DeepSeek shows the strongest and most stable agreement, achieving substantial alignment with Rater 1 ($\alpha = 0.741$) and moderate alignment with Rater 2 ($\alpha = 0.516$). ChatGPT demonstrates moderate agreement with both raters, but remains less consistent than DeepSeek. Gemini achieves fair agreement across both raters, with scores that are similar between Rater 1 and Rater 2 but lower than DeepSeek's. Overall, the figure highlights that DeepSeek provides the most reliable performance.

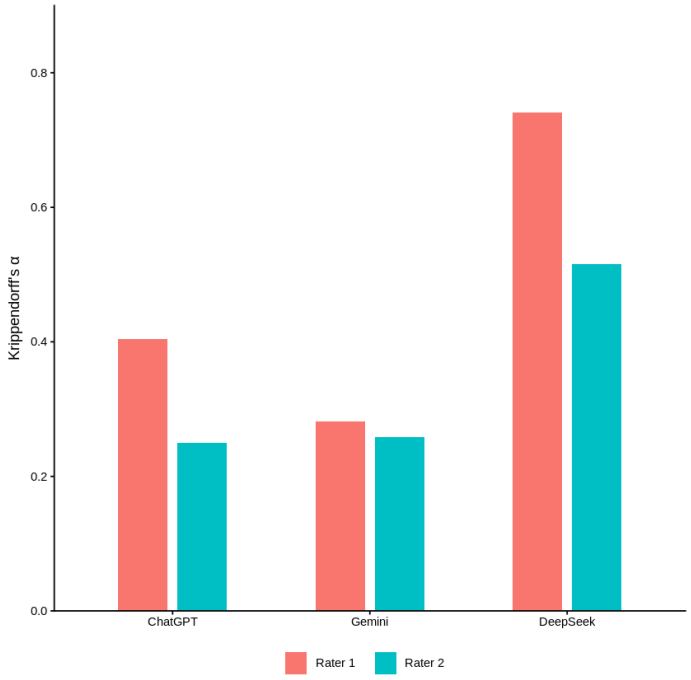

Figure 4.1: Krippendorff's α for each LLM with each human rater, using the final selected prompt version (ChatGPT-4o V2, Gemini 2.5 Pro V4, DeepSeek-R1 V4). Higher values indicate stronger agreement with the corresponding human rater.

The observed performance differences illustrate that the three LLMs responded differently to the same prompt-engineering strategies. In our experiments, DeepSeek exhibited the largest performance gains as prompt context and task constraints were incrementally refined, whereas ChatGPT achieved its best performance with a more concise and streamlined prompt structure. Gemini benefited most from explicit instructions to avoid over-classification based on scientific terminology alone.

4.3.3.1 Final Prompt Specifications

ChatGPT-4o (V2): We selected Version 2 for ChatGPT as it avoided the risk-averse defaults observed in later versions while maintaining balanced performance. This version provided consistent agreement rates of 71.3% with Rater 1 and 68.8% with Rater 2.
Gemini 2.5 Pro (V4): Version 4 addressed Gemini's tendency to over-classify PRs containing scientific terminology by adding explicit instructions to ignore scientific terms present only in configuration changes or parameter adjustments.
DeepSeek-R1 (V4): In Version 4, DeepSeek benefited from improved context handling through clearer examples and sharper distinctions between scientific innovation and

routine software changes. This led to the highest overall human alignment, with 87.5% agreement with Rater 1 and 77.5% agreement with Rater 2.

Complete prompts for each LLM are provided in Appendix A.2.

4.3.4 LLM Results

Using the optimized prompts described in Section 4.3.2, we applied three LLMs, ChatGPT-4o, Gemini 2.5 Pro, and DeepSeek-R1, to classify all 1,074 filtered PRs as either requiring scientific knowledge (label = 1) or not (label = 0). Each model was run independently using its final selected prompt version (see Table 4.2), ensuring consistent input structure across the dataset.

The classification process for each PR involved:

1. Parsing the PR metadata, including title, description, labels, and discussion activity.
2. Analyzing associated code changes, including files modified and relevant code diffs, to provide implementation context.
3. Applying the binary scientific knowledge criteria defined in Section 3.3 to assign a label.

Table 4.3 summarizes the number and proportion of PRs each LLM labeled as requiring scientific knowledge.

Table 4.3: Counts of PRs labeled as requiring scientific knowledge by each LLM

LLM	**Count of PRs**	**% of Dataset**
ChatGPT-4o	297	27.6%
Gemini 2.5 Pro	676	63.0%
DeepSeek-R1	654	60.9%

To further examine how these differences vary across repositories and to compare them with human judgments, Table 4.4 provides a per-repository breakdown of classifications for the human agreed labels and LLMs.

Table 4.4: Number and percentage of PRs classified as Scientific (SK=1) by human agreed labels and LLMs across repositories

Model	Trilinos	Mantid	AMReX
Agreed Labels (Human)	124 (34.9%)	259 (68.0%)	117 (34.6%)
ChatGPT-4o	58 (16.3%)	193 (50.7%)	46 (13.6%)
Gemini 2.5 Pro	124 (34.9%)	362 (95.0%)	191 (56.5%)
DeepSeek-R1	162 (45.6%)	314 (82.4%)	178 (52.7%)

As shown in Table 4.4, the three LLMs exhibit distinct and systematic classification biases relative to the human agreed labels. ChatGPT-4o consistently under-classifies scientific PRs across all repositories, producing substantially lower proportions than the human baseline. In contrast, Gemini 2.5 Pro demonstrates a clear tendency to over-classify PRs as scientific, particularly in Mantid, where nearly all PRs are labeled as SK=1 (95.0%). DeepSeek-R1 shows the closest alignment with the human agreed labels, producing proportions that are broadly consistent across all repositories.

Agreement within the LLM set was moderate. All three models produced the same label for 602 PRs (56.0% of the dataset), corresponding to a Krippendorff's α of 0.415. Pairwise comparisons further indicate that DeepSeek aligns more closely with the human agreed labels than the other LLMs. These results suggest that, while prompt optimization improves per-model performance, substantial inter-model variability remains in interpreting the binary scientific knowledge definition.

4.4 Baseline Model

To better understand the performance of LLM-based classification, we establish simple baseline models that require minimal computational cost and no prompt engineering. The purpose of these baselines is to evaluate whether simple, low-cost methods can achieve comparable performance to LLM-based approaches.

4.4.1 Majority Class Baseline (ZeroR)

To establish a lower performance bound, we implement a ZeroR classifier, which assigns all PRs to the majority class observed in the dataset. This model ignores all input features and relies solely on the class distribution of the labels.

To obtain a high-confidence ground truth, we evaluate only on the subset of PRs where both human annotators agreed on the classification. This agreement-based subset comprises 906 PRs and reduces label noise, providing a reliable basis for baseline comparison.

4.4.2 Naive Bayes Classifier

To complement the ZeroR baseline and provide a simple feature-based comparison, we implement a Multinomial Naive Bayes classifier as a lightweight machine learning baseline. Unlike ZeroR, which relies solely on label distribution and ignores all input features, Naive Bayes incorporates textual information from PRs to learn patterns associated with scientific and non-scientific contributions.

To provide a stronger yet still lightweight baseline, we use the PR title and body as textual input features. These fields are concatenated and transformed into TF-IDF representations, allowing the model to learn shallow lexical patterns associated with the binary classification labels (SK = 0 or SK = 1). The model is trained to estimate the probability of each class based on the presence of words in the input text.

This design allows us to evaluate whether simple text-based statistical learning is sufficient for this task, or whether more advanced semantic understanding, as provided by LLMs, is required.

4.4.3 Result

The Naive Bayes classifier is evaluated on the agreement subset, where both human annotators assign identical labels. The dataset is split into training (80%) and testing (20%) sets. Although the model benefits from both the PR title and description, its performance remains substantially below that of the best-performing LLMs, indicating that shallow textual learning is insufficient for accurately capturing domain-specific scientific knowledge requirements.

Table 4.5: Baseline model performance

Model	Evaluation Set	Train Size	Test Size	Predicted SK=1	Predicted SK=0	Accuracy (%)
ZeroR	Agreement	–	906	906	0	55.19
Naive Bayes	Agreement	724	182	127	55	64.84

Table 4.5 summarizes the results of both baseline models on the agreement subset. ZeroR establishes a minimal reference point by always predicting the majority class, achieving an accuracy of 55.19%, which reflects the class distribution rather than any learned distinction.

The Naive Bayes classifier improves upon this baseline, achieving an accuracy of 64.84% by leveraging TF-IDF representations of PR titles and descriptions. However, the improvement remains limited, indicating that shallow textual features are insufficient for capturing the domain-specific scientific knowledge required for accurate classification.

4.5 Model Comparison

Table 4.6 presents the classification performance of the LLMs on the agreement subset. All models substantially outperform the baseline approaches, demonstrating their ability to capture meaningful patterns beyond simple statistical or feature-based methods.

Table 4.6: LLM classification performance on agreement subset

Model	Total PRs	Predicted SK=1	Predicted SK=0	Accuracy
ChatGPT-4o	906	280	626	72.85%
Gemini 2.5 Pro	906	516	390	83.66%
DeepSeek-R1	906	500	406	100.00%

The ZeroR baseline achieves an accuracy of 55.19% by predicting the majority class (SK = 1) for all PRs, providing a minimal performance threshold. The Naive Bayes classifier improves upon this baseline, achieving 64.84% accuracy by leveraging textual features from PR titles and descriptions. However, the improvement remains limited, indicating that shallow lexical patterns are insufficient for reliably distinguishing between scientific and non-scientific contributions.

In contrast, all LLMs achieve significantly higher accuracy. ChatGPT reaches 72.85%, while Gemini achieves 83.66%, demonstrating strong performance gains over traditional baselines. Most notably, DeepSeek achieves an apparent accuracy of 100.00% on the agreement subset. However, this result should be interpreted with caution. The evaluation is restricted to PRs where both human annotators agree, which reduces ambiguity and simplifies the classification task.

In addition, DeepSeek's predicted class distribution exactly matches the ground truth distribution (500 SK=1 and 406 SK=0), suggesting that the perfect accuracy may reflect sensitivity to dataset characteristics rather than true flawless generalization.

This behavior may indicate that the model is strongly aligned with dominant patterns in the dataset, and the agreement-based evaluation setting may inadvertently favor such alignment. Therefore, while the result demonstrates high consistency with human labels, it does not necessarily imply perfect classification performance in more ambiguous or real-world settings.

Instead, Table 4.3 presents the distribution of predictions produced by each model. These results highlight systematic differences in model behavior, with ChatGPT-4o tending to under-classify scientific PRs, while Gemini and DeepSeek show a stronger tendency to assign SK=1 labels.

These results highlight a clear performance gap between classical approaches and LLM-based methods. While baseline models rely on label distribution or surface-level textual features, LLMs are able to interpret context, capture domain-specific concepts, and reason about the intent of PRs. This capability is particularly important in research software, where distinguishing scientific contributions often requires understanding specialized terminology and implicit domain knowledge.

4.6 External Validation (PlasmaPy)

To evaluate the generalizability of our approach beyond PRs, we conducted a complementary experiment using issues from the PlasmaPy/PlasmaPy repository. This project provided a distinctive test case due to its explicit use of domain-specific labeling conventions that indicate scientific complexity levels, a feature not present in our primary study repositories (Trilinos, Mantid, AMReX). Unlike PRs, which typically include code changes, issues primarily contain textual descriptions, problem statements, and design discussions, allowing us to assess LLM performance in a more metadata-driven context.

4.6.1 Methodology

We collected 75 issues from PlasmaPy/PlasmaPy using the GitHub API, focusing on those tagged with the project's domain-specific labels shown in Table 4.7. These labels represent a formalized expertise hierarchy, ranging from beginner to expert levels and covering specific plasma physics subdomains.

Table 4.7: Domain-related labels used for issue collection in `PlasmaPy/PlasmaPy` [36]

Label	Description
needs subject matter expert	Requires expertise from a plasma science domain expert
Plasma Lv1 | Beginner	Issues appropriate for someone who has some knowledge of physics
Plasma Lv2 | Intermediate	Issues that require an intermediate knowledge of plasma physics
Plasma Lv3 | Proficient	Issues that require proficiency in plasma physics
Plasma Lv4 | Expert	Issues that require very detailed knowledge of plasma physics
plasmapy.analysis	Related to the plasmapy.analysis subpackage; focused on plasma data analysis [31]
plasmapy.diagnostics	Related to the plasmapy.diagnostics subpackage; focused on plasma diagnostic methods and data processing [32]
plasmapy.formulary	Related to the plasmapy.formulary subpackage; focused on plasma physics formulas, physical parameters, and core plasma models [34]
plasmapy.formulary.quantum	Related to the plasmapy.formulary.quantum module; focused on quantum plasma physics [35]
plasmapy.particles	Related to the plasmapy.particles subpackage; focused on atomic, ionic, and particle-level plasma representations and ionization states [37]
plasmapy.plasma	Related to the plasmapy.plasma subpackage; focused on plasma object models and physical plasma system representations [38]
plasmapy.dispersion	Related to the plasmapy.dispersion subpackage; focused on plasma wave dispersion relations and wave propagation modeling [33]

Our initial assumption that maintainer-assigned domain labels could serve as reliable ground truth for scientific knowledge requirements was found to be incorrect. PlasmaPy's labeling scheme represents a graded expertise hierarchy intended to guide

task assignment rather than a binary indicator of whether scientific domain knowledge is required to resolve an issue. In particular, issues labeled at lower domain levels (e.g., Plasma Lv1 | Beginner) typically require only basic scientific knowledge rather than expert-level domain knowledge. Instead, they may primarily demand general programming knowledge or higher-level software engineering expertise. When we assigned SK = 1 to all labeled issues under this naive assumption, comparison with DeepSeek's predictions showed poor reliability ($\alpha = -0.2845$), with the negative score indicating systematic disagreement rather than random variation. This disagreement arises from conflating task-assignment semantics with our operational definition of scientific knowledge requirements. Re-labeling the issues using our established binary SK definition resolved this mismatch and restored substantial agreement.

We therefore manually relabeled all 75 issues using our established binary criteria, creating validated ground truth through expert assessment. This process yielded 49 issues requiring scientific knowledge (SK=1) and 26 requiring only general software knowledge (SK=0).

For classification, we adapted our PR prompt to the issue context, replacing code-specific elements with issue metadata including titles, descriptions, labels, and discussion threads. The core binary classification framework remained consistent with our main study to ensure comparability.

4.6.1.1 Results

DeepSeek demonstrated strong performance on the issue classification task, achieving substantial agreement with human labels ($\alpha = 0.7793$) and an overall accuracy of 89.33%. As shown in Table 4.8, the model presented high precision (0.911) for identifying scientific issues, with a slightly lower recall (0.837), reflecting its cautious labeling style that prioritized avoiding false positives instead of assigning all true scientific issues.

Table 4.8: DeepSeek classification performance on PlasmaPy issues (ground truth: manual labels)

Metric	Value
Accuracy	0.893
Precision (SK=1)	0.911
Recall (SK=1)	0.837
F1-score (SK=1)	0.872
Matthews Correlation Coefficient (MCC)	0.800

This evaluation is not circular, even though the issues were manually labeled and the prompt includes definitions and examples. The manual labels establish what is meant by "scientific knowledge required", while the prompt instructs DeepSeek to apply this definition in a consistent manner. In this setting, strong performance is expected and appropriate. The goal is not to test whether the model can discover the labels, but whether an LLM can consistently apply a human-defined labeling rule across a large set of issues. DeepSeek's role is therefore to replicate human judgment at scale, reducing the need for repeated manual labeling rather than replacing the human-defined classification itself.

The confusion matrix (Table 4.9) reveals that DeepSeek never misclassified non-scientific issues as scientific (no false positives), but missed 8 scientific issues (false negatives). This conservative approach ensures domain experts won't be overloaded with irrelevant issues but may occasionally miss appropriate assignments.

Table 4.9: Confusion matrix for PlasmaPy issue classification

	$\mathbf{Pred}_{SK=0}$	$\mathbf{Pred}_{SK=1}$
$\mathbf{Actual}_{SK=0}$	26	0
$\mathbf{Actual}_{SK=1}$	8	41

Notably, our manual validation confirmed the effectiveness of PlasmaPy's labeling system. As shown in Table 4.10, we observed a clear monotonic relationship between the designated expertise levels and actual scientific knowledge requirements. Beginner-level issues rarely required domain expertise (16.7%), while expert-level issues consistently demanded advanced scientific knowledge (100%). This progression validates the

project's labeling conventions and suggests such systems could be valuable for other scientific software projects.

Table 4.10: SK $= 1$ distribution by Plasma Level and Domain-Related Tags

Category	**Total Issues (N)**	**Count$_{\text{SK}=1}$**	**Rate$_{\text{SK}=1}$**
Plasma Lv1 (Beginner)	24	4	16.7%
Plasma Lv2 (Intermediate)	20	10	50.0%
Plasma Lv3 (Proficient)	12	10	83.3%
Plasma Lv4 (Expert)	8	8	100.0%
needs subject matter expert	12	10	83.3%
subpackage tags (`plasmapy.*`)	46	33	71.7%

4.7 Discussion

The results presented in Section 4.3.4 and the external validation in Section 4.6 provide strong evidence that LLMs can effectively automate the classification of PRs into scientific versus software contributions, which addressed ***RQ2***. Our findings yield several key insights into model selection, the value of prompt engineering, and the nature of scientific software contributions.

First, the consistent performance gap between baseline models and LLM-based approaches highlights the importance of semantic understanding for this task. The ZeroR and Naive Bayes baselines rely on label distribution and shallow lexical features, respectively, and fail to capture the contextual and domain-specific reasoning required to distinguish scientific from software-oriented contributions. In contrast, LLMs demonstrate a clear ability to interpret intent, reason about code changes, and incorporate both textual and structural information from PRs.

Second, the observed differences across LLMs reveal that model behavior is highly sensitive to both architecture and prompt design. ChatGPT tends to adopt a conservative classification strategy, under-labeling scientific PRs, while Gemini exhibits the opposite tendency, frequently over-classifying based on the presence of scientific terminology. DeepSeek achieves the strongest alignment with human annotators, suggesting that it is more effective at integrating contextual signals and distinguishing between superficial terminology and genuine scientific reasoning. These findings indicate that LLM-based classification is not model-agnostic, and careful model selection is required.

Third, prompt design plays a critical role in shaping model performance. The transition from multi-scale labeling to binary classification significantly improved agreement, confirming that reducing task ambiguity is essential. Additionally, explicitly instructing models to ignore irrelevant scientific terminology proved necessary to mitigate systematic biases. The differing responses of the three LLMs to prompt refinements further emphasize that prompt engineering must be tailored to individual model characteristics rather than applied uniformly.

Fourth, the agreement-based evaluation strategy provides a reliable framework for benchmarking model performance. By restricting evaluation to the subset of PRs where human annotators agree, we reduce label noise and establish a high-confidence ground truth. The near-perfect alignment achieved by DeepSeek on this subset demonstrates that LLMs can replicate human judgment with high fidelity when clear labeling criteria are provided.

Finally, the external validation on PlasmaPy issues demonstrates that the proposed classification framework generalizes beyond PRs to other repository artifacts. Despite the absence of code diffs, DeepSeek maintains strong performance, indicating that LLMs can effectively operate in metadata-driven contexts. This suggests that LLM-based classification can support a broader range of repository mining tasks, including issue triage and task assignment.

Overall, these findings support the feasibility of using LLMs as scalable proxies for human judgment in classifying scientific knowledge requirements, while also highlighting the importance of prompt design, model selection, and evaluation methodology.

4.8 Limitations

Several limitations should be considered when interpreting the results of this study.

First, the binary classification framework simplifies the spectrum of scientific knowledge requirements. While the binary SK=0/1 formulation improves agreement and reproducibility, it does not capture finer-grained distinctions in the level or type of domain expertise required. Some PRs may involve moderate or partial scientific reasoning that is not fully represented by a binary label.

Second, the evaluation relies on agreement between two human annotators, with consensus used to resolve disagreements. Although this approach improves label reliability, it remains subjective and dependent on the annotators' expertise and interpretation of the labeling criteria. The absence of a larger and more diverse annotation pool

may limit the generalizability of the ground truth.

Third, the study focuses on three repositories (Trilinos, Mantid, and AMReX), which, while representative of research software, do not cover the full diversity of scientific software ecosystems. Different domains may exhibit different patterns of contribution, which could affect classification performance.

Fourth, the held-back dataset used for prompt refinement is relatively small (120 PRs), and the prompt optimization process may still be influenced by overfitting to this subset. Although care was taken to exclude these PRs from final evaluation, the limited size constrains the exploration of more diverse prompt variations.

Fifth, the comparison between LLMs and baseline models is limited to relatively simple baselines. While ZeroR and Naive Bayes provide useful reference points, more advanced traditional models, such as transformer-based classifiers fine-tuned on repository data, could offer stronger baselines for comparison.

Finally, the external validation on PlasmaPy issues involves a relatively small dataset (75 issues) and relies on manual relabeling. While the results demonstrate promising generalization, further validation on larger and more diverse issue datasets is required to confirm robustness.

These limitations suggest that while the results are encouraging, they should be interpreted as an initial step toward scalable classification rather than a complete solution.

4.9 Conclusion

This study investigates the use of LLMs for automatically classifying PRs in research software based on their scientific knowledge requirements. Using the PRIMES framework, we developed a structured methodology that integrates prompt engineering, multi-model evaluation, and human validation.

Our results show that LLMs substantially outperform traditional baseline models, demonstrating the ability to capture contextual and domain-specific signals that are not accessible through shallow feature-based approaches. Among the evaluated models, DeepSeek achieves the highest agreement with human annotators, indicating that LLMs can replicate human judgment with high reliability when guided by clear and well-defined prompts.

The findings also highlight the importance of prompt design and model-specific optimization. Reducing task ambiguity, incorporating relevant context, and explicitly addressing known failure modes significantly improve classification performance. At the

same time, differences in model behavior indicate that LLM-based approaches require careful calibration rather than a one-size-fits-all strategy.

Beyond PR classification, the successful application of the framework to PlasmaPy issues demonstrates its potential for broader use in repository mining tasks. LLMs can serve as scalable tools for supporting triage, task assignment, and analysis in research software projects.

Overall, this work provides evidence that LLMs can effectively bridge the gap between domain knowledge and software engineering in the context of research software. By enabling automated classification of scientific contributions, this approach supports more efficient repository management and opens new directions for applying LLMs in empirical software engineering.

Chapter 5

Discussion & Limitations

5.1 Discussion

5.1.1 Classification and LLM as Judge

The performance of DeepSeek-R1 ($\alpha = 0.789$ with human annotators on PRs) and its consistent results on issues ($\alpha = 0.779$ on PlasmaPy) confirm that automated identification is not only feasible but can also achieve a reliability that approaches "good agreement" thresholds established in the literature. The inclusion of baseline models further highlights that the observed performance gains are not due to simple statistical patterns, but instead require the semantic and contextual reasoning capabilities of LLMs. The significant performance differences observed among the three LLMs, DeepSeek-R1, Gemini 2.5 Pro, and ChatGPT-4o, can be attributed to a combination of architectural differences, training data composition, and built-in model biases.

These differences demonstrate that LLMs cannot be treated as interchangeable, which makes model selection a crucial decision that depends on the task. This would typically involve a pilot phase in which candidate models are benchmarked on a small, representative sample of the target data using task-relevant reliability metrics. Subsequently, the selected model requires iterative prompt calibration to align its reasoning with the task-specific definition, as illustrated by the model-specific refinements in our study. In deployed settings, this process may need to be periodically refined to accommodate model updates or shifts in project requirements.

Overall, the evidence indicates that LLMs, particularly DeepSeek-R1 with optimized prompting, have strong potential to support automated reviewer assignment systems in scientific software development. When integrated into a human-guided workflow

where the LLM provides initial classification and humans review uncertain cases, this approach could significantly reduce classification costs and time, balance workload distribution, and ensure that contributor expertise is applied efficiently. This aligns with recent results in LLMs-as-judges in software engineering tasks [1].

5.1.2 Context Engineering

We followed the PRIMES framework [9] in iteratively creating prompts to guide the LLMs. However, this is only one aspect of leveraging LLMs for these purposes. Adding additional agents that could retrieve relevant scientific knowledge, applying reasoning models, or implementing more robust context engineering practices would likely improve performance further. However, correctly tuning context to avoid context rot and forgetting is not trivial [27]. Domain knowledge is increasingly important in the context engineering era. LLMs have excellent general-purpose insights but require careful tuning for specific domains.

5.1.3 Implications for Research Software Engineers

An important constituency is the practitioners who support these scientific repositories. Our results suggest that the consistent resource intensity of scientific PRs indicates they should be prioritized and staffed differently. PRs requiring scientific knowledge (SK=1) undergo a fundamentally different and more intensive review process compared to their non-scientific counterparts. This aligns with insights [43] that research software often crosses disciplinary boundaries, imposing a higher degree of cognitive friction.

Allocating more time for review of scientific PRs, proactively assigning multiple reviewers with complementary expertise (e.g., a domain scientist and a software architect), and setting expectations for a longer, more iterative process could improve both efficiency and code quality.

5.1.4 Methodological and Interpretive Limitations

While this thesis demonstrates that LLM-based classification of PRs in research software is feasible, several methodological and interpretive limitations should be acknowledged.

First, the classification framework relies on a human-defined operationalization of "scientific knowledge," which necessarily reflects the perspectives and judgments of the annotators involved. Although consensus-based adjudication improved reliability, the

boundary between scientific-specific and software-specific contributions remains sensitive. Contributors who work closely on a given project may have different opinions on classifications, especially in borderline cases where scientific ideas appear through configuration files or parameter settings rather than explicit changes to algorithms. For this reason, the labels should be understood as a practical approximation of expertise requirements rather than a definitive accepted ground truth.

Second, the study focuses on a limited number of large, mature research software projects. While this selection enables robust statistical analysis and reduces noise from inactive repositories, it may bias findings toward projects with established review cultures and clear role differentiation. Smaller or less formal research codebases may exhibit different dynamics, and the observed patterns may not fully generalize to those settings.

Third, the use of LLMs as classification agents introduces dependencies on model behavior, prompt design, and provider-specific updates. Although Chapter 3 demonstrates that careful prompt refinement and human validation can yield reliable results, long-term deployment would require ongoing monitoring to ensure stability as models evolve. This also raises questions about how contributors would perceive and trust automated classifications in practice.

Fourth, our comparison is limited to relatively simple baseline models. More advanced non-LLM approaches, such as fine-tuned transformer-based classifiers, may provide stronger baselines and should be explored in future work.

Finally, the analysis in Chapter 4 relies on observable process metrics derived from the two datasets we constructed from PRs in the selected repositories. While combining multiple metrics mitigates the limitations of any single measure, these metrics still only approximate review effort and coordination. They cannot capture informal communication, offline discussions, or the perceived difficulty of a change for reviewers, all of which may affect the review process but are not visible in repository data.

5.2 Implications

5.2.1 Implications for Maintainers and Developers

The introduction of LLM-driven PR classification in research software projects can significantly influence developer workflows. For maintainers, an immediate benefit is more efficient triage and reviewer assignment [15]. With each incoming pull request labeled

as requiring scientific knowledge (SK=1) or not (SK=0), maintainers can quickly route scientific PRs to domain experts and software PRs to core engineers.

This targeted assignment can reduce review iteration cycles, as the right expertise is applied from the start. In practice, this could be implemented via integration with platforms like GitHub: an automated comment or tag could appear on new PRs indicating their category and potentially pinging relevant expert teams.

Another implication is risk management: knowing the nature of a PR helps focus review efforts appropriately. Maintainers might treat scientific PRs with extra scrutiny regarding correctness of results, while software PRs might be examined more for code quality and maintainability. Over time, these practices could improve overall codebase quality—scientific code receives thorough domain review reducing scientific errors, while software changes get attention needed to maintain software health.

For everyday developers (especially new contributors), these labels also serve as feedback and learning opportunities. A contributor might realize their PR touched areas beyond their expertise if classified as scientific, prompting them to seek input from domain specialists [30]. Clearly labeled PRs can lead to more focused discussion threads.

5.2.2 Implications for Research and Tool Builders

Our work sits at the intersection of software engineering and AI, opening several research directions. It demonstrates a novel application of NLP and LLM techniques to repository data by categorizing changes by domain knowledge requirements, a dimension not previously explored in depth.

This suggests researchers should further explore semantic understanding of code changes: beyond our categories, what other high-level meanings of code edits can AI capture? The success of LLMs in interpreting PR descriptions implies future studies could mine repositories for other latent dimensions (e.g., “educational value” of a PR, or whether a PR represents quick fix vs. architectural change).

It also highlights the importance of datasets that include domain context, encouraging researchers to incorporate domain-specific analysis in empirical studies. For the research community working on automated code review, our approach demonstrates that context-aware review tools are feasible, moving beyond checking code style or tests to understanding what kind of knowledge is needed for review.

We expect to see more intelligent review assistants that can, for example, read a PR and suggest relevant domain documentation to reviewers, or flag when a PR might need

outside expert input.

5.2.3 Integration into Real Workflows

Tool builders and platform providers (such as GitHub or GitLab) might consider integrating this classification as a feature. The implication is that developer platforms could become more "intelligent" by providing insights, not just raw data. For instance, GitHub could use a model to auto-tag PRs (similar to how issues can be auto-labeled by bots [13]). This would especially benefit large interdisciplinary projects where triage is difficult.

Our findings that scientific-heavy PRs take longer to merge, require more reviewers, and generate more discussion could encourage project managers to allocate resources differently. If a project knows that domain-centric changes stagnate due to the lack of expertise, they might formally assign a domain scientist as a co-maintainer or ensure documentation is improved for that code area.

In essence, one implication is cultural or process change: projects might adopt dual review policies for scientific PRs (requiring both science and software reviewers), or create roles like "science software liaison" to handle cross-cutting changes.

Our classification and project metrics could also feed into analytics dashboards. Maintainers could monitor the proportion of Scientific vs. Software PRs over time as an indicator of project evolution (e.g., are we spending more time on science or software this release?).

5.2.4 Summary

In conclusion, the implications of LLM-driven PR classification are both practical and aspirational. Practically, it promises more efficient project maintenance and clearer allocation of expert attention. From a broader perspective, it points toward a future where AI assists in bridging gaps between different expertise domains, ultimately fostering software that is both scientifically correct and engineering-wise robust. By integrating these intelligent tools, both researchers and practitioners can move toward more seamless synergy in research software development.

Chapter 6

Future Work & Conclusion

6.1 Conclusion

This thesis presents a novel and effective methodology for automating the classification of pull requests in scientific software using Large Language Models. Our investigation shows that the core challenge of triaging contributions in interdisciplinary projects—distinguishing those requiring domain expertise from those focused on software engineering—can be addressed with strong reliability.

To address **RQ1**, we conducted a systematic empirical analysis of review processes using PRs categorized by their scientific knowledge requirements. The results reveal consistent and substantial differences in review characteristics between scientific-specific and software-specific pull requests. Even after controlling for change size and repository-specific effects, scientific PRs require significantly more intensive review processes: they take 67% longer to merge, involve 64% more unique reviewers, generate over 100% more discussion comments, and undergo more than 300% more post-review revisions. These findings indicate that scientific contributions impose a distinct and heavier review burden, characterized by increased validation effort and coordination among contributors.

To address **RQ2**, we developed a binary classification framework based on the necessity of scientific knowledge. This simplified approach proved significantly more reliable than more complex multi-scale methods. While the initial multi-scale system yielded low agreement, the final binary framework enabled DeepSeek-R1 to achieve near-human reliability with a Krippendorff's α of 0.789. These results demonstrate that LLMs can effectively distinguish between scientific-specific and software-specific con-

tributions based on PR content. The generalizability of this approach is further supported by the PlasmaPy case study, where the same framework achieved 89.33% accuracy in issue classification.

Beyond these measurable findings, the proposed framework has broader implications for interdisciplinary collaboration. By making expertise requirements explicit, it can support more informed reviewer assignment, improve triage efficiency, and facilitate knowledge exchange between scientists and software engineers. Over time, this visibility may encourage cross-disciplinary learning and contribute to the development of more well-rounded contributors.

In summary, this work establishes a foundation for integrating LLM-based automation into scientific software workflows. By enabling reliable identification of the expertise required for reviewing contributions, it addresses a key challenge in collaborative research environments and supports more efficient allocation of domain knowledge, ultimately contributing to more robust and maintainable scientific software.

6.2 Future Work

Although our LLM-driven PR classification approach showed promising results, several avenues for future improvement and exploration remain.

6.2.1 Real-Time Integration

One immediate next step is integrating the classification system into real development workflows. For example, a real-time GitHub bot could use our model to label incoming PRs automatically. This would require engineering the solution for live use, possibly using faster or more cost-efficient models, or leveraging GitHub Actions to trigger classification on new PRs.

Studying such a bot in practice would allow observation of how maintainers use the labels—do they assign different reviewers automatically based on the tag? Does it reduce review latency? A live deployment would provide usability feedback and expose unforeseen challenges, such as the need for manual override in misclassification cases.

6.2.2 Broader Validation Studies

We focused on a handful of repositories; evaluating the approach across a wider range of research software projects would strengthen generalizability. Future work should

apply classification to diverse domains (e.g., bioinformatics, climate science, astronomy) and analyze whether observed patterns persist.

It would also be useful to investigate outlier projects that show no significant difference between Scientific and Software PRs, to understand mitigating factors like team size or structure. Developer surveys and interviews could validate whether the categories are meaningful and identify opportunities to refine or expand them.

6.2.3 Multi-Dimensional Classification

Our current approach classifies PRs by scientific vs. software content. Future work could explore multi-label classification, such as tagging both purpose (bug fix, feature, refactoring) and domain orientation.

For instance, a PR might be labeled "Feature + Scientific" or "Bug Fix + Software." This could improve triage by assigning both domain reviewers and testing specialists. Achieving this requires prompt or model design supporting multi-faceted outputs, potentially combining prior commit classification work with our domain-based categorization.

6.2.4 Impact on Collaboration

Another direction is assessing how classification affects team collaboration and knowledge sharing. Longitudinal studies could monitor whether labeled PRs lead to more cross-domain participation. For example, if engineers review more Scientific PRs or domain experts engage with Software PRs.

Interviews with contributors could reveal whether classification helps identify silos or promotes mentorship between scientists and developers. Metrics like co-review frequency or response time could indicate whether labels impact workflow efficiency and inclusivity.

A subtle but important implication is potential improvement in knowledge sharing. By making PR types explicit, teams acknowledge that different expertise areas exist and are valued.

Over time, this can encourage cross-pollination: software specialists might pay more attention to scientific PRs to learn about the domain, and scientists might learn software best practices by engaging with software PRs.

In academic software teams, students or researchers with primarily scientific training might become more aware of software engineering considerations when they see

PRs labeled as requiring software expertise, prompting them to learn or consult with engineers on performance or refactoring issues. This could lead to more well-rounded contributors.

From an education perspective, automated classification could be used in mentoring: newcomers can be shown feeds of past PRs labeled by type to understand what contribution types are typical and how they were handled.

6.2.5 Domain-Specific Model Variants

As LLMs evolve, domain-specific variants may offer greater classification accuracy. Future models could incorporate scientific knowledge (e.g., from papers or formulas) or software engineering context (e.g., design patterns) to more precisely identify change nature.

Hybrid or ensemble systems could combine multiple classifiers, such as one for scientific logic and one for software structure, merging their outputs into unified labels with explanations like: "Classified as Scientific because it modifies a physics equation-solving function."

6.2.6 Summary

Future work spans both technical improvements (accuracy, speed, deployment) and sociotechnical dimensions (collaboration, adoption, organizational impact). By addressing these areas, we can move closer to making intelligent PR classification a standard tool in research software engineering, ultimately improving how interdisciplinary teams collaborate on scientific software development.

Bibliography

[1] Toufique Ahmed, Premkumar Devanbu, Christoph Treude, and Michael Pradel. Can llms replace manual annotation of software engineering artifacts? In *Proceedings of the 21st International Conference on Mining Software Repositories (MSR 2024)*. ACM, 2024.

[2] Elvira-Maria Arvanitou, Apostolos Ampatzoglou, Alexander Chatzigeorgiou, and Jeffrey C. Carver. Software engineering practices for scientific software development: A systematic mapping study. *Journal of Systems and Software*, 172:110848, February 2021. doi:10.1016/j.jss.2020.110848.

[3] Sebastian Baltes, Florian Angermeir, Chetan Arora, Marvin Muñoz Barón, Chunyang Chen, Lukas Böhme, Fabio Calefato, Neil Ernst, Davide Falessi, Brian Fitzgerald, Davide Fucci, Marcos Kalinowski, Stefano Lambiase, Daniel Russo, Mircea Lungu, Lutz Prechelt, Paul Ralph, Rijnard van Tonder, Christoph Treude, and Stefan Wagner. Guidelines for empirical studies in software engineering involving large language models, 2025. URL: https://arxiv.org/abs/2508.15503, arXiv:2508.15503.

[4] Narjes Bessghaier, Ali Ouni, Mohammed Sayagh, Moataz Chouchen, and Mohamed Wiem Mkaouer. Towards understanding code review practices for infrastructure-as-code: An empirical study on openstack projects. *Empirical Software Engineering*, 30(3):106, 2025.

[5] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever,

and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020. URL: https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[6] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL: https://arxiv.org/abs/2107.03374, arXiv:2107.03374.

[7] Moataz Chouchen, Narjes Bessghaier, Mahi Begoug, Ali Ouni, Eman Alomar, and Mohamed Wiem Mkaouer. How do software developers use chatgpt? an exploratory study on github pull requests. In *Proceedings of the 21st International Conference on Mining Software Repositories*, MSR '24, page 212–216, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3643991.3645084.

[8] Giuseppe Colavito. Foundation models for automatic issue labeling. In *2025 IEEE/ACM 47th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 127–131, 2025. doi:10.1109/ICSE-Companion66252.2025.00038.

[9] Vincenzo De Martino, Joel Castaño, Fabio Palomba, Xavier Franch, and Silverio Martínez-Fernández. Experiences from using llms for repository mining studies in empirical software engineering. In *Proceedings of the 2nd International Workshop on*

Methodological Issues with Empirical Studies in SE (WSESE 2025, ICSE Workshops). IEEE, 2025.

[10] Nasir U. Eisty and Jeffrey C. Carver. Developers perception of peer code review in research software development. *Empirical Software Engineering*, 27(1), October 2021. URL: http://dx.doi.org/10.1007/s10664-021-10053-x, doi:10.1007/s10664-021-10053-x.

[11] Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M. Zhang. Large language models for software engineering: Survey and open problems. In *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)*, pages 31–53, 2023. doi:10.1109/ICSE-FoSE59343.2023.00008.

[12] Niklas Fridh and Lukasz Stypa. Classification of pull requests using transformers. In *Proceedings of the 20th International Conference on Mining Software Repositories (MSR)*. ACM, 2022.

[13] GitHub Marketplace. Issue-Label Bot. https://github.com/marketplace/issue-label-bot. Accessed: 2026-01-01.

[14] Georgios Gousios, Martin Pinzger, and Arie Van Deursen. A dataset for pull-based development research. In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR)*, pages 368–371. ACM, 2014.

[15] Georgios Gousios, Martin Pinzger, and Arie van Deursen. An exploratory study of the pull-based software development model. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE '14. ACM, May 2014. URL: http://dx.doi.org/10.1145/2568225.2568260, doi:10.1145/2568225.2568260.

[16] Georgios Gousios, Andy Zaidman, Margaret-Anne Storey, and Arie van Deursen. Work practices and challenges in pull-based development: The integrator's perspective. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. IEEE, May 2015. URL: http://dx.doi.org/10.1109/ICSE.2015.55, doi:10.1109/icse.2015.55.

[17] Jo Erskine Hannay, Carolyn MacLeod, Janice Singer, Hans Petter Langtangen, Dietmar Pfahl, and Greg Wilson. How do scientists develop and use scientific software? In *2009 ICSE Workshop on Software Engineering for Computational*

Science and Engineering, pages 1–8, Vancouver, BC, Canada, May 2009. IEEE. doi:10.1109/SECSE.2009.5069155.

[18] Dustin Heaton and Jeffrey C. Carver. Claims about the use of software engineering practices in science: A systematic literature review. *Information and Software Technology*, 67:207–219, November 2015. URL: http://dx.doi.org/10.1016/j.infsof.2015.07.011, doi:10.1016/j.infsof.2015.07.011.

[19] James Howison and James D. Herbsleb. Scientific software production: Incentives and collaboration. In *Proceedings of the ACM 2011 Conference on Computer Supported Cooperative Work - CSCW '11*, page 513, Hangzhou, China, 2011. ACM Press. doi:10.1145/1958824.1958904.

[20] Xing Huang, Xianghua Ding, Charlotte P. Lee, Tun Lu, and Ning Gu. Meanings and boundaries of scientific software sharing. In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*, pages 423–434, San Antonio Texas USA, February 2013. ACM. doi:10.1145/2441776.2441825.

[21] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL: https://openreview.net/forum?id=VTF8yNQM66.

[22] Marina Jirotka, Charlotte P. Lee, and Gary M. Olson. Supporting Scientific Collaboration: Methods, Tools and Concepts. *Computer Supported Cooperative Work (CSCW)*, 22(4-6):667–715, August 2013. doi:10.1007/s10606-012-9184-0.

[23] Diane Kelly. Scientific software development viewed as knowledge acquisition: Towards understanding the development of risk-averse scientific software. *Journal of Systems and Software*, 109:50–61, 2015.

[24] Klaus Krippendorff. *Content analysis: An introduction to its methodology*. Sage publications, 2018.

[25] Gunnar Kudrjavets, Nachiappan Nagappan, and Ayushi Rastogi. Do small code changes merge faster? a multi-language empirical investigation. In *Proceedings of the 19th International Conference on Mining Software Repositories*, MSR '22, page 537–548, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3524842.3528448.

[26] Gunnar Kudrjavets, Nachiappan Nagappan, and Ayushi Rastogi. Are we speeding up or slowing down? on temporal aspects of code velocity. In *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*, page 267–271. IEEE, May 2023. URL: http://dx.doi.org/10.1109/MSR59073.2023.00046, doi:10.1109/msr59073.2023.00046.

[27] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024. URL: https://aclanthology.org/2024.tacl-1.9/, doi:10.1162/tacl_a_00638.

[28] Zhongxin Liu, Xin Xia, Christoph Treude, David Lo, and Shanping Li. Automatic generation of pull request descriptions. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 176–188, 2019. doi:10.1109/ASE.2019.00026.

[29] Addi Malviya-Thakur, Reed Milewicz, Lavínia Paganini, Ahmed Samir Imam Mahmoud, and Audris Mockus. Scicat: A curated dataset of scientific software repositories. In *Proceedings of the 21st International Conference on Mining Software Repositories (MSR 2024)*. ACM, 2024.

[30] Gustavo Pinto, Igor Wiese, and Luiz Felipe Dias. How do scientists develop scientific software? an external replication. In *2018 IEEE 25th international conference on software analysis, evolution and reengineering (SANER)*, pages 582–591. IEEE, 2018.

[31] PlasmaPy Developers. Plasmapy analysis documentation. https://docs.plasmapy.org/en/latest/ad/analysis/index.html.

[32] PlasmaPy Developers. Plasmapy diagnostics documentation. https://docs.plasmapy.org/en/latest/ad/diagnostics/index.html.

[33] PlasmaPy Developers. Plasmapy dispersion documentation. https://docs.plasmapy.org/en/latest/dispersion/index.html.

[34] PlasmaPy Developers. Plasmapy formulary documentation. https://docs.plasmapy.org/en/stable/formulary/index.html.

[35] PlasmaPy Developers. Plasmapy formulary quantum documentation. https://docs.plasmapy.org/en/latest/formulary/quantum.html.

[36] PlasmaPy Developers. Plasmapy github repository labels. https://github.com/PlasmaPy/PlasmaPy/labels.

[37] PlasmaPy Developers. Plasmapy particles documentation. https://docs.plasmapy.org/en/stable/particles/index.html.

[38] PlasmaPy Developers. Plasmapy plasma documentation. https://docs.plasmapy.org/en/stable/plasma/index.html.

[39] Rohith Pudari, Shiyuan Zhou, Iftekhar Ahmed, Zhuyun Dai, and Shurui Zhou. Aligning documentation and q and a forum through constrained decoding with weak supervision. In *2023 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 346–351, October 2023. doi:10.1109/ICSME58846.2023.00043.

[40] Judith Segal. Some challenges facing software engineers developing software for scientists. In *2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*, pages 9–14, 2009. doi:10.1109/SECSE.2009.5069156.

[41] Vanessa Sochat, Nicholas May, Ian Cosden, Carlos Martinez-Ortiz, and Sadie Bartholomew. The Research Software Encyclopedia: A Community Framework to Define Research Software. *Journal of Open Research Software*, 2022. doi:10.5334/jors.359.

[42] Jiayi Sun, Aarya Patil, Youhai Li, Jin L.C. Guo, and Shurui Zhou. Collaboration challenges and opportunities in developing scientific open-source software ecosystem: A case study on astropy. *Proc. ACM Hum.-Comput. Interact.*, 9(7), October 2025. URL: https://doi-org.myaccess.library.utoronto.ca/10.1145/3757462, doi:10.1145/3757462.

[43] Jiayi Sun, Aarya Patil, Youhai Li, Jin LC Guo, and Shurui Zhou. Collaboration challenges and opportunities in developing scientific open-source software ecosystem: A case study on astropy. *Proceedings of the ACM on Human-Computer Interaction*, 9(7):1–33, 2025.

[44] The Mantid Project team. Manipulation and Analysis Toolkit for Instrument Data, July 2025. URL: https://github.com/mantidproject/mantid, doi:10.5286/Software/Mantid6.13.1.

[45] The Trilinos Project Team. *The Trilinos Project Website*, 2020. URL: https://trilinos.github.io.

[46] Jason Tsay, Laura Dabbish, and James Herbsleb. Influence of social and technical factors for evaluating contribution in github. In *Proceedings of the 36th International Conference on Software Engineering (ICSE)*, pages 356–366. ACM, 2014.

[47] Priyan Vaithilingam, Tianyi Zhang, and Elena L. Glassman. Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, New York, NY, USA, April 2022. Association for Computing Machinery. doi:10.1145/3491101.3519665.

[48] Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. Openhands: An open platform for AI software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*, 2025. URL: https://openreview.net/forum?id=OJd3ayDDoF.

[49] Yue Yu, Huaimin Wang, Vladimir Filkov, Premkumar Devanbu, and Cong Yu. Wait for it: Determinants of pull request evaluation latency on github. In *Mining Software Repositories (MSR), 2015 IEEE/ACM 12th Working Conference on*, pages 367–371. IEEE, 2015.

[50] Weiqun Zhang, Ann Almgren, Vince Beckner, John Bell, Johannes Blaschke, Cy Chan, Marcus Day, Brian Friesen, Kevin Gott, Daniel Graves, Max Katz, Andrew Myers, Tan Nguyen, Andrew Nonaka, Michele Rosso, Samuel Williams, and Michael Zingale. AMReX: A Framework for Block-Structured Adaptive Mesh Refinement. *Journal of Open Source Software*, 4(37):1370, 2019. URL: https://github.com/AMReX-Codes/amrex, doi:10.21105/joss.01370.

Appendix A

Appendix

A.1 Classification Prompts

A.1.1 Prompt V1

```
You are an expert research assistant for scientific software
development.
I've uploaded a CSV file containing metadata for pull requests (PRs)
from three scientific software repositories.
Each row contains:
repo: the repository name (e.g., trilinos, mantid, amrex)
url: the GitHub PR link
title: the PR title
user: GitHub username of the PR author
created_at: when the PR was opened
merged_at: when the PR was merged
commits: total number of commits in the PR
commits_after_review: number of commits submitted after receiving
reviews
inline_comments: number of comments made on specific lines of code
issue_comments: general comments related to the PR
review_comments: formal code review comments
total_discussion_comments: total comments across all types
time_between_first_last_comment: duration (in seconds) between first
and last comment
```

time_to_merge: time (in seconds) from PR creation to merge
requested_changes: number of times reviewers requested changes
unique_reviewers: count of unique reviewers involved
avg_comment_resolution_sec: average time (in seconds) to resolve reviewer comments
labels: GitHub labels assigned to the PR

Your task is to evaluate each PR and rate it on the following two dimensions:
1. Scientific Knowledge (SK)
1 (Low): No or minimal scientific concepts involved.
2 (Medium): Required applying well-established domain concepts and understanding how they are typically implemented.
3 (High): Heavily focused on scientific models, equations, or domain-specific algorithms, which involve complex or innovative solutions or integrate recent research.

2. Software Knowledge (SW)
1 (Low): Only required basic coding skills
2 (Medium): Understood advanced programming techniques, design patterns, and efficient implementation.
3 (High): Required deep knowledge of software architecture, performance optimization, and designing complex systems.

Please return your results in this CSV format:
repo, url, title, Scientific_Knowledge, Software_Knowledge
cp2k, https://github.com/cp2k/cp2k/pull/1234, ..., 2, 3
trilinos, https://github.com/trilinos/Trilinos/pull/5678, ..., 3, 2
...
...

Examples:
PR: https://github.com/trilinos/Trilinos/pull/7678
Title: Amesos2: Handle x and b copying more efficiently

Body: This PR optimizes how the Amesos2 solver handles copying of x and b vectors during solve. It introduces flags to skip unnecessary deep copies and only performs memory duplication when required. This improves performance for solvers like SuperLU and KLU2.
→ Scientific_Knowledge = 2, Software_Knowledge = 3
PR: https://github.com/mantidproject/mantid/pull/32598
Title: Move Crystal Field Python interface documentation to correct folder
Body: This PR moves documentation for the Crystal Field Python interface from the indirect interface folder to the direct interface folder. No code changes were made.
→ Scientific_Knowledge = 1, Software_Knowledge = 1
PR: https://github.com/mantidproject/mantid/pull/37753
Title: From Mantid: Add algorithm for calculating Wildes polarization efficiencies
Body: This PR introduces a new algorithm, PolarizationEfficienciesWildes, to calculate polarization efficiencies for a two-flipper instrument setup based on the Wildes 2006 paper, with error handling using default Mantid errors.
→ Scientific_Knowledge = 3, Software_Knowledge = 2

A.1.2 Prompt V2

Introduction
You are a research assistant with expertise in scientific software development and domain-specific reasoning. I have uploaded a CSV file that contains metadata for pull requests (PRs) from three scientific software repositories: Trilinos, Mantid, and AMReX.

Each row in the dataset represents a single PR and includes the following metadata:
repo: name of the repository (e.g., trilinos, mantid, amrex)
url: GitHub link to the PR
title: title of the PR
user: GitHub username of the author

created_at / merged_at: timestamps for when the PR was opened and merged
commits / commits_after_review: total commits and number of commits submitted after review
inline_comments / issue_comments / review_comments: breakdown of comment types
total_discussion_comments: total number of all comments
time_between_first_last_comment / time_to_merge / avg_comment_resolution_sec: time-related metrics (in seconds)
requested_changes: number of times reviewers requested changes
unique_reviewers: number of unique reviewers who participated
labels: GitHub labels assigned to the PR
files_changed: names of the files modified in the pull request
Your Task
For each PR, evaluate whether understanding or contributing to the PR would require scientific domain knowledge, and assign a Scientific Knowledge (SK) label based on the following scale:
Action Instructions
You can now begin assigning the Scientific Knowledge (SK) label (0 = No Need, 1 = Need) to each PR based on the full set of metadata columns provided in the CSV file.

Please return a modified version of the input CSV file with an additional column titled Scientific_Knowledge, containing the SK label for each PR. The output must be in CSV format. Do not return a preview or partial table.
Scientific Knowledge Label Definitions
0 (No Need)
The PR may involve domain-related variables or concepts in name only, but understanding or modifying the code does not require any actual domain knowledge.
Common signs:
- The change is a straightforward extension of an existing pattern
- The code is copied and pasted from similar logic (e.g., copying "coordinates" logic to create "materials" logic)

- The update involves wiring, plumbing, or parameter propagation without reasoning about scientific models

Example:
PR: https://github.com/trilinos/Trilinos/pull/13960
Title: MueLu: Add material vector information into MueLu multiphysics class
Although the PR introduces a variable named arrayOfMaterials, the code simply duplicates logic used for arrayOfCoords. No knowledge of how material vectors are derived, used, or modeled is needed.
→ Scientific_Knowledge = 0

1 (Need)
The PR does require understanding scientific concepts to implement or review correctly. This includes:
- Algorithms grounded in scientific papers
- Simulation model design
- Physical solver behavior
- Data transformations based on physical rules
- Choosing between scientific models or parameters

Note: "Scientific knowledge" refers specifically to knowledge in domain sciences (e.g., physics, materials science, computational chemistry), not general programming or software engineering skills.
Output Format
Please return your results in a CSV format with the following columns: repo, url, title, Scientific_Knowledge
Examples for Calibration
Example 1
PR: https://github.com/trilinos/Trilinos/pull/7678
Title: Amesos2: Handle x and b copying more efficiently
Body: Optimizes copying behavior in the Amesos2 solver for better performance when interfacing with SuperLU and KLU2 solvers.
→ Scientific_Knowledge = 1

Example 2
PR: https://github.com/mantidproject/mantid/pull/32598
Title: Move Crystal Field Python interface documentation to correct folder
Body: Relocates documentation; no changes to scientific computation or logic.
→ Scientific_Knowledge = 0

Example 3
PR: https://github.com/mantidproject/mantid/pull/37753
Title: Add algorithm for calculating Wildes polarization efficiencies
Body: Implements a scientific algorithm based on a peer-reviewed paper to calculate polarization efficiency in neutron instruments.
→ Scientific_Knowledge = 1

A.1.3 Prompt V3

Introduction
You are a research assistant with expertise in scientific software development and domain-specific reasoning. I have uploaded a CSV file that contains metadata for pull requests (PRs) from three scientific software repositories: Trilinos, Mantid, and AMReX.

Each row in the dataset represents a single PR and includes the following metadata:
repo: name of the repository (e.g., trilinos, mantid, amrex)
url: GitHub link to the PR
title: title of the PR
user: GitHub username of the author
created_at / merged_at: timestamps for when the PR was opened and merged
commits / commits_after_review: total commits and number of commits submitted after review
inline_comments / issue_comments / review_comments: breakdown of comment types
total_discussion_comments: total number of all comments

time_between_first_last_comment / time_to_merge /
avg_comment_resolution_sec: time-related metrics (in seconds)
requested_changes: number of times reviewers requested changes
unique_reviewers: number of unique reviewers who participated
labels: GitHub labels assigned to the PR
files_changed: names of the files modified in the pull request
code_diff: a summary or snippet of the code difference introduced by the pull request
Your Task
For each PR, evaluate whether understanding or contributing to the PR would require scientific domain knowledge, and assign a Scientific Knowledge (SK) label based on the following scale:
Action Instructions
You can now begin assigning the Scientific Knowledge (SK) label (0 = No Need, 1 = Need) to each PR based on the full set of metadata columns provided in the CSV file.

Please return a modified version of the input CSV file with an additional column titled Scientific_Knowledge, containing the SK label for each PR. The output must be in CSV format. Do not return a preview or partial table.
Scientific Knowledge Label Definitions
0 (No Need)
The PR may involve domain-related variables or concepts in name only, but understanding or modifying the code does not require any actual domain knowledge.
Common signs:
- The change is a straightforward extension of an existing pattern
- The code is copied and pasted from similar logic (e.g., copying "coordinates" logic to create "materials" logic)
- The update involves wiring, plumbing, or parameter propagation without reasoning about scientific models

Example:
PR: https://github.com/trilinos/Trilinos/pull/13960

Title: MueLu: Add material vector information into MueLu multiphysics class
Although the PR introduces a variable named arrayOfMaterials, the code simply duplicates logic used for arrayOfCoords. No knowledge of how material vectors are derived, used, or modeled is needed.
→ Scientific_Knowledge = 0

1 (Need)
The PR does require understanding scientific concepts to implement or review correctly. This includes:
- Algorithms grounded in scientific papers
- Simulation model design
- Physical solver behavior
- Data transformations based on physical rules
- Choosing between scientific models or parameters

Note: "Scientific knowledge" refers specifically to knowledge in domain sciences (e.g., physics, materials science, computational chemistry), not general programming or software engineering skills.
Important Clarification
Do not assign Scientific_Knowledge = 1 just because a PR mentions scientific-sounding terms like "algorithm," "matrix," or "aggregation."
You must evaluate whether the logic being modified or added involves scientific computation, modeling, or reasoning.

For example, the following changes should still be labeled 0:
- Adding or registering parameter strings (e.g., "aggregation: coloring algorithm")
- Adding print/logging statements
- Changing default settings or mode selectors
- Extending a list of options that invoke pre-existing algorithms, without modifying the algorithm itself

Scientific_Knowledge = 1 should only apply when the PR changes or implements scientific logic, such as:
- New solver routines
- Numerical method changes
- Physics-based model adjustments
- Changes grounded in equations or papers
Internal Reasoning Checklist
For each PR, internally verify:
- What logic is being changed?
- Does resolving this PR require understanding of a scientific model or simulation concept?
- Is the author modifying or adding a scientific algorithm, or just exposing/configuring one?
Then, based on this reasoning, assign a final Scientific_Knowledge value.
Output Format
Please return your results in a CSV format with the following columns: repo, url, title, Scientific_Knowledge
Examples for Calibration
Example 1
PR: https://github.com/trilinos/Trilinos/pull/7678
Title: Amesos2: Handle x and b copying more efficiently
Body: Optimizes copying behavior in the Amesos2 solver for better performance when interfacing with SuperLU and KLU2 solvers.
→ Scientific_Knowledge = 1

Example 2
PR: https://github.com/mantidproject/mantid/pull/32598
Title: Move Crystal Field Python interface documentation to correct folder
Body: Relocates documentation; no changes to scientific computation or logic.
→ Scientific_Knowledge = 0

Example 3

PR: https://github.com/mantidproject/mantid/pull/37753
Title: Add algorithm for calculating Wildes polarization efficiencies
Body: Implements a scientific algorithm based on a peer-reviewed paper to calculate polarization efficiency in neutron instruments.
→ Scientific_Knowledge = 1

A.1.4 Prompt V4

Introduction
You are a research assistant with expertise in scientific software development and domain-specific reasoning. I have uploaded a CSV file that contains metadata for pull requests (PRs) from three scientific software repositories: Trilinos, Mantid, and AMReX.

Each row in the dataset represents a single PR and includes the following metadata:
repo: name of the repository (e.g., trilinos, mantid, amrex)
url: GitHub link to the PR
title: title of the PR
user: GitHub username of the author
created_at / merged_at: timestamps for when the PR was opened and merged
commits / commits_after_review: total commits and number of commits submitted after review
inline_comments / issue_comments / review_comments: breakdown of comment types
total_discussion_comments: total number of all comments
time_between_first_last_comment / time_to_merge / avg_comment_resolution_sec: time-related metrics (in seconds)
requested_changes: number of times reviewers requested changes
unique_reviewers: number of unique reviewers who participated
labels: GitHub labels assigned to the PR
files_changed: names of the files modified in the pull request
code_diff: a detailed code diff for all files changed in the PR, including additions and deletions

Your Task
For each PR, evaluate whether understanding or contributing to the PR would require scientific domain knowledge, and assign a Scientific Knowledge (SK) label based on the following scale:

Action Instructions
You can now begin assigning the Scientific Knowledge (SK) label (0 = No Need, 1 = Need) to each PR based on the full set of metadata columns provided in the CSV file.

Please return a modified version of the input CSV file with an additional column titled Scientific_Knowledge, containing the SK label for each PR. The output must be in CSV format. Do not return a preview or partial table.

Scientific Knowledge Label Definitions
0 (No Need)
The PR may reference domain-related variables or concepts in name only, but understanding or modifying the code does not require any actual domain knowledge.
Common signs:
- The change is a straightforward extension of existing logic without requiring reasoning about the science
- The code reuses patterns from unrelated logic without altering the underlying scientific model
- The update involves software infrastructure, build scripts, or documentation without affecting scientific computation

Example:
PR: https://github.com/trilinos/Trilinos/pull/13960
Title: MueLu: Add material vector information into MueLu multiphysics class
Although the PR introduces a variable named arrayOfMaterials, the change is structurally identical to adding arrayOfCoords. No understanding of materials science is required.

→ Scientific_Knowledge = 0

1 (Need)
The PR requires comprehension of scientific concepts to implement or review correctly. This includes:
- Algorithms or methods grounded in scientific research
- Modifications to numerical solvers or simulation logic
- Changes based on physics, chemistry, or other domain-specific models
- Implementation of equations or algorithms from peer-reviewed sources

Important Clarification
Do not assign Scientific_Knowledge = 1 solely because scientific-sounding terms appear in variable names, comments, or labels.
Assign SK=1 only when the change requires applying, interpreting, or implementing scientific logic or methodology.

For example, SK=0 if:
- The change simply adds parameters to existing functions
- The PR adjusts configuration files or test cases without altering the scientific model
- The PR only wraps or exposes pre-existing algorithms without modifying their core scientific logic

Internal Reasoning Checklist
For each PR, internally verify:
- What logic is being changed?
- Does this logic require understanding of a scientific model or simulation process?
- Is the PR introducing or altering a scientific algorithm, or is it simply refactoring existing code?
Then, based on this reasoning, assign a final Scientific_Knowledge value.

Output Format

Please return your results in a CSV format with the following columns: repo, url, title, Scientific_Knowledge

Examples for Calibration
Example 1
PR: https://github.com/trilinos/Trilinos/pull/7678
Title: Amesos2: Handle x and b copying more efficiently
Body: Optimizes Amesos2 solver behavior for specific scientific solvers, requiring understanding of solver data structures and numerical methods.
→ Scientific_Knowledge = 1

Example 2
PR: https://github.com/mantidproject/mantid/pull/32598
Title: Move Crystal Field Python interface documentation to correct folder
Body: Relocates documentation without changing computation or scientific content.
→ Scientific_Knowledge = 0

Example 3
PR: https://github.com/mantidproject/mantid/pull/37753
Title: Add algorithm for calculating Wildes polarization efficiencies
Body: Implements a scientific algorithm based on a peer-reviewed publication, involving physics-based computations for neutron polarization.
→ Scientific_Knowledge = 1

A.2 PlasmaPy Issue Classification Prompt

A total of 75 issues containing at least one of the predefined PlasmaPy labels (as listed in Section 4.6) were extracted. All were **open** at the time of extraction and had no linked PRs.

```
Introduction
You are a research assistant with expertise in scientific software
development and domain-specific reasoning. I have uploaded a CSV file
that contains metadata for issues from the PlasmaPy scientific
software repository.
Each row in the dataset represents a single issue and includes the
following metadata:
URL: GitHub link to the issue
Title: Title of the issue
State: Whether the issue is open or closed
Type: Should always be "Issue"
Labels: GitHub labels assigned to the issue
Description: The main issue body content written by the author and any
details or proposals
Linked PRs: Pull requests linked to the issue (if any)

Your Task
For each issue, evaluate whether understanding or resolving it would
require scientific domain knowledge, and assign a Scientific Knowledge
(SK) label based on the following scale:

Action Instructions
You can now begin assigning the Scientific Knowledge (SK) label (0 =
No Need, 1 = Need) to each issue based on the full set of metadata
columns provided in the CSV file.

Please return a modified version of the input CSV file with an
additional column titled Scientific_Knowledge, containing the SK label
for each issue. The output must be in CSV format. Do not return a
preview or partial table.
```

Scientific Knowledge Label Definitions
0 (No Need)
The issue relates to purely general software engineering tasks or community logistics. The discussion and resolution do not require domain-specific knowledge in plasma physics or scientific modeling.
Common signs:
- Typos, documentation, or formatting issues
- GitHub configuration, CI, or test infra work
- UI/UX bugs or general API usability
- Refactoring or parameter renaming without touching scientific logic
- Meta discussions (e.g., about roadmaps or labels)

1 (Need)
The issue involves understanding or modifying scientific logic, equations, plasma physics concepts, or numerical methods. These require background knowledge to interpret or resolve.
Common signs:
- Discussion of physical models, algorithms, or parameters
- Designing/adjusting numerical solvers or simulations
- Referencing scientific papers or plasma concepts
- Adding/modifying scientific data processing or simulation logic

Important Clarification
Do not assign Scientific_Knowledge = 1 just because the issue mentions scientific-sounding terms. Label 1 should only apply when resolving the issue requires actual scientific understanding.

Internal Reasoning Checklist
For each issue, internally verify:
- What logic is being changed?
- Does resolving this issue require understanding of a scientific model or simulation concept?
- Is the author modifying or adding a scientific algorithm, or just exposing/configuring one?

Then, based on this reasoning, assign a final Scientific_Knowledge value.

Output Format
Please return your results in a CSV format with the following columns: repo, url, title, Scientific_Knowledge
Where Scientific_Knowledge is either 0 or 1 based on your reasoning.

Examples for Calibration
Example 1
URL: "https://github.com/PlasmaPy/PlasmaPy/issues/766"
Title: "Create a class to represent a dust particle in a plasma"
Description: "Dust is common in many plasmas such as the interstellar medium (ISM). (Yay space dust!) We should have a means of representing dust particles somewhere in `plasmapy.particles`, perhaps called `DustParticle`. Dust particles have some basic attributes like mass and charge. There will be more complicated attributes like composition, which I'm not sure how to describe. There will likely need to be subclasses such as `SphericalDustParticle` and `EllipticalDustParticle`. I don't know if this should be a subclass of `CustomParticle` (#447, #755). We should talk with dusty plasma physicists to get their thoughts on what should be included in `DustParticle` and how we describe it. "
Scientific_Knowledge = 1

Example 2
URL: "https://github.com/PlasmaPy/PlasmaPy/issues/2923"
Title: "Create a class to represent a dust particle in PlasmaPy"

```
Description: "I recently learned about
[JuliaPlasma](https://github.com/JuliaPlasma) which is a project to
implement awesome plasma functionality in Julia. JuliaPlasma is the
home of
[`PlasmaFormulary.jl`](https://github.com/JuliaPlasma/PlasmaFormulary.jl),
which is a Julia package being developed to contain a plasma
formulary, akin to `plasmapy.formulary`.     We should:     - [ ] Link
to JuliaPlasma in a prominent location on PlasmaPy's documentation   -
[ ] Link to `PlasmaFormulary.jl` from the main page for
`plasmapy.formulary`    It'd also help to add this to our website in
addition to our documentation, but that's a separate repo."
Scientific_Knowledge = 0
```

A.3 PlasmaPy Issues with Divergent Labels

Table A.1 lists the 8 PlasmaPy issues where DeepSeek's scientific knowledge classification differed from both human raters. Each entry includes the GitHub issue number, title, human SK label, DeepSeek SK label, and a brief note on possible causes for the disagreement.

Issue #	Title	Rater_{SK}	$\text{DeepSeek}_{\text{SK}}$	Possible Cause
2639	Add helper functionality to plot quantities against temperature with axes labeled in both K and eV	1	0	DeepSeek treated this as a generic plotting task, missing that labeling axes in electronvolts requires understanding plasma physics unit conventions.
2141	Enable '@particle_input' to accept iterables for 'mass_numb' and 'Z'	1	0	DeepSeek focused on the decorator enhancement but overlooked that handling particle isotopes and atomic numbers demands domain knowledge of particle physics notation.
1286	Create a Sphinx role to represent particles	1	0	DeepSeek saw only the Sphinx documentation tooling aspect, failing to recognize that formatting particle representations correctly requires plasma particle expertise.
2464	Link to PySPEDAS null point solver from our documentation & notebook	1	0	DeepSeek interpreted this as a simple documentation link update, missing that "null point solver" refers to a specialized plasma physics algorithm.
1575	Adjust signature of 'magnetic_field' method for magnetostatic classes	1	0	DeepSeek focused on method signature refactoring while overlooking that magnetic field coordinate design requires magnetostatics domain knowledge.
1412	Create a table of standard variable names and mathematical symbols	1	0	DeepSeek treated this as documentation organization, failing to see that defining standard notation for plasma quantities requires plasma physics expertise.
1393	Perform user experience testing for plasma calculator	1	0	DeepSeek viewed this as usability testing, missing that evaluating a plasma calculator's correctness requires understanding of plasma parameter calculations.
732	Plasma parameters need to be tested to a precision expected by their uncertainty	1	0	DeepSeek considered this generic testing methodology, overlooking that determining appropriate precision for plasma parameters requires knowledge of their physical uncertainties.

Table A.1: PlasmaPy issues labeled $SK = 1$ by both human raters but $SK = 0$ by DeepSeek.