prismAId User Manual

Riccardo Boero

ribo@nilu.no

May 5, 2025

Version: v0.0.3

DOI: 10.5281/zenodo.15025694

Contents

Foreword

The `prismAId` user manual is designed to help researchers, academics, and professionals leverage the power of `prismAId` toolkit for conducting systematic reviews efficiently. This document provides a structured approach to installing, configuring, and using the various components of `prismAId`, ensuring that users can quickly get started while also exploring advanced features when needed.

Reminder: No coding skills are required to effectively conduct a systematic review with `prismAId`.

This manual is divided into five distinct parts, each catering to different user needs:

- **Part 1: Introduction** – If you are new to `prismAId`, we recommend starting with the chapter **Introduction to prismAId**. This section explains what `prismAId` is, the modular tools it provides, who can benefit from it, and why it is a valuable toolkit for systematic reviews.

Tip: The many open science advantages of `prismAId` are introduced and discussed in the **Introduction**.

- **Part 2: Getting Started** – If you need to install and configure `prismAId`, go directly to **Installation & Setup**. This section provides step-by-step installation instructions for Windows, Mac, and Linux. Once installed, **Configuring prismAId** explains how to customize settings and prepare your environment for use of each tool in the toolkit.

Note: `prismAId` is available on all platforms and operating systems. It can also be integrated programmatically with the most widely used scientific software.

- **Part 3: Conducting a Systematic Review** – If your primary goal is to **learn how to conduct a systematic review with `prismAId`**, you can skip the installation and configuration sections and go directly to the chapter **Step-by-Step Guide to Conducting a Systematic Review**. This section provides a hands-on walkthrough of:
  - Setting up a project
  - Acquiring and preparing literature using the Download and Convert tools
  - Configuring and running systematic reviews using the Review tool

Tip: For a quick but complete walkthrough on using `prismAId` in a systematic review, see the **fifth** chapter.

- Interpreting results
- Exporting findings

  Before diving into this walkthrough, you may find it helpful to read **Understanding Systematic Reviews**, which provides background information on the methodology and best practices.

- **Part 4: Advanced Features** – Users who want to explore advanced capabilities can refer to the chapters **Customization and Automation** and **Extending prismAId**. These sections cover advanced options for each tool, ensemble reviews, rate limiting, cost minimization, and integration with external tools and workflows.

- **Part 5: Troubleshooting & FAQs** – If you encounter issues, visit **Troubleshooting Common Issues**, which provides solutions to common errors for each of the toolkit's components, and **Frequently Asked Questions**, which addresses common concerns about usage and implementation.

How Should You Use This Manual?

- If you are **new to** `prismAId`, start with **Introduction**, then proceed to **Installation**.

- If you **want to conduct a complete systematic review**, go directly to **Part 3**, particularly **Step-by-Step Guide to Conducting a Systematic Review**, for a detailed walkthrough of the entire process.

- If you only need **specific tools** from the toolkit (Download, Convert, or Review), the respective chapters in **Part 3** detail how to use each component independently.

- If you need **advanced features or troubleshooting**, refer to **Parts 4 and 5** as needed.

Throughout this manual, useful commands are presented in blue boxes, as shown below:

Command: Example of a Binary Command

To run a systematic review with a TOML configuration file:

```
# For Windows
./prismaid.exe -project your_project.toml

# For downloading papers from Zotero
./prismaid.exe -download-zotero zotero_config.toml

# For converting PDF files to text
./prismaid.exe -convert-pdf ./papers
```

Tool configuration details are highlighted in red boxes, as shown below:

Configuration: Example of a TOML Setting

To configure an LLM for your review:

```
# Sample TOML configuration
[project.llm.1]
provider = "OpenAI"
api_key = ""
model = ""
temperature = 0.2
tpm_limit = 0
rpm_limit = 0
```

Final Notes

We hope this manual serves as a comprehensive guide to making the most of the `prismAId` toolkit. Whether you're a first-time user or an advanced researcher, this document is structured to help you get the information you need quickly and efficiently. The modular design of `prismAId` allows you to use only the components you need, making it adaptable to various research workflows and requirements.

For any additional questions, please refer to **the FAQ section** or contact our support team through the GitHub repository or Matrix Support Room. Happy reviewing!

Warning: The online documentation of `prismAId` is available at open-and-sustainable.github.io/prismaid/.

Part I

Introduction

Systematic reviews are at the core of evidence-based research. They help synthesize vast amounts of literature, ensuring that decisions in science, medicine, policy, and other fields are grounded in the best available evidence. However, conducting systematic reviews is a time-intensive and often overwhelming process. Screening literature, managing citations, extracting data, and ensuring methodological rigor all demand meticulous attention to detail.

`prismAId` was created to address these challenges. It is an open-source toolkit designed to assist researchers in conducting systematic reviews more efficiently using generative AI technology. By streamlining key aspects of the review process, `prismAId` helps users maintain high standards of rigor and reproducibility while reducing the manual workload.

Tip: Systematic reviews require structured workflows. `prismAId` provides modular tools to help enforce best practices at each stage of the process.

The toolkit is designed for a diverse range of users, from researchers conducting large-scale systematic reviews to students working on literature-based projects, requiring no coding skills to operate effectively. It provides structured workflows that align with best practices in systematic reviewing, ensuring that every step—from acquiring literature to extracting and analyzing data—is traceable and transparent.

The prismAId Toolkit

A key enhancement to `prismAId` is its modular design, which now offers three specialized tools to support different stages of the systematic review process:

- **Download Tool**: Acquires papers directly from Zotero collections or from URL lists, streamlining the literature acquisition phase.
- **Convert Tool**: Transforms documents from various formats (PDF, DOCX, HTML) into plain text that can be processed by large language models.
- **Review Tool**: Processes systematic literature reviews based on configurable protocols, extracting structured information from scientific papers.

This modular approach allows researchers to integrate `prismAId` into existing workflows more flexibly, using only the components needed for their specific research context.

Note: Each tool can be used independently or as part of an integrated workflow. This modular design supports diverse research needs and practices.

Open Science Foundation

One of the key motivations behind `prismAId` is its commitment to Open Science. The toolkit is fully open source, meaning its development is transparent, and researchers can inspect, modify, and contribute to its functionality. This openness ensures that systematic reviews conducted with `prismAId` can be fully reproducible and that researchers can collaborate effectively without relying on proprietary or closed software ecosystems.

Moreover, the open-source nature of `prismAId` allows for continuous improvement through community-driven development. Users can adapt the tools to fit their needs, extend their capabilities, and integrate them with other research software. This flexibility is particularly beneficial in a rapidly evolving scientific landscape where reproducibility and adaptability are crucial.

Warning: As an open-source toolkit, `prismAId` does not include proprietary support. Users are encouraged to engage with the community through GitHub issues or the Matrix Support Room for troubleshooting and development.

Multi-Platform Accessibility

`prismAId` is accessible through multiple platforms and programming languages, offering flexibility based on user preference and technical requirements:

- **Standalone Binaries**: For Windows, macOS, and Linux, requiring no coding skills
- **Go Package**: For integration in Go-based projects
- **Python Package**: For integration in Python scripts and Jupyter notebooks
- **R Package**: For use within R and RStudio environments
- **Julia Package**: For integration in Julia workflows

This multi-platform approach ensures that researchers can incorporate `prismAId` into their preferred computational environments without disrupting existing workflows.

Guiding Principles

Beyond technical efficiency, `prismAId` embodies key guiding principles that impact its users:

- **Transparency**: The logic behind how studies are managed and processed is open for review. There are no hidden decision-making mechanisms.

- **Reproducibility**: Every action taken within `prismAId` can be logged and traced, allowing others to replicate results with comprehensive documentation of methodologies.
- **Flexibility**: Researchers can configure `prismAId` to meet the specific requirements of their systematic review protocols and adapt it to various research domains.
- **Accessibility**: Easy-to-use interfaces ensure that anyone can leverage advanced AI tools for literature reviews without specialized technical knowledge.
- **Efficiency**: The toolkit is optimized for handling large datasets with minimal setup, reducing the time from research to results.
- **No Vendor Lock-in**: Users are not dependent on a single commercial provider, ensuring that research workflows remain independent.
- **Community and Collaboration**: As an open-source toolkit, `prismAId` fosters collaboration among researchers, developers, and systematic review specialists.

Reminder: Collaboration is a core feature. Consider contributing to `prismAId` if you have ideas for improvement!

By embracing these principles, `prismAId` is more than just a toolkit—it is part of a broader effort to improve the accessibility, efficiency, and reliability of systematic reviews. Whether used by a lone researcher or a large team, it provides the structure and support needed to navigate complex literature and synthesize high-quality evidence using the latest advancements in artificial intelligence.

As we move through this manual, you will learn how to install, configure, and effectively use each component of the `prismAId` toolkit to conduct systematic reviews. From getting started with basic setup to leveraging advanced features for more complex reviews, this guide will provide all the necessary information to integrate `prismAId` into your research workflow.

Note: This manual follows a structured approach. Refer to the **How Should You Use This Manual?** section in the Foreword for guidance on navigating the content.

Part II

Getting Started

Installation & Setup

This chapter explains how to install `prismAId`, covering system requirements, installation methods, and first-time setup for each of the toolkit's components.

System Requirements

Before installing `prismAId`, ensure your system meets the following requirements:

Warning: Ensure that you have administrative privileges on your system before installing `prismAId`, especially on Windows and macOS.

- **Operating System**: Windows 10 or later, macOS 11 or later, or a Linux distribution (Ubuntu 20.04+, Fedora, Arch, etc.).
- **Processor**: 64-bit CPU (Intel, AMD, or ARM64).
- **Memory**: At least 4GB RAM (8GB recommended).
- **Storage**: Minimum 500MB of free disk space.
- **Internet Connection**: Required for downloading packages and accessing LLM APIs.
- **API Keys**: To use the Review tool, you'll need an API key from at least one of the supported LLM providers (OpenAI, GoogleAI, Cohere, Anthropic, or DeepSeek).

Installation Methods

`prismAId` can be installed in multiple ways, depending on your preferences and workflow requirements. Choose the method that best suits your needs.

Method 1: Standalone Binaries

The simplest installation method is to download the pre-compiled binaries, which require no additional dependencies.

Note: Binaries are standalone executables that work on Linux, macOS, and Windows, supporting both AMD64 and Arm64 architectures.

1. Navigate to the GitHub Releases page.

2. Download the appropriate binary for your operating system.

3. Extract the file to a directory of your choice.

Windows

Command: Running `prismAId` on Windows

After downloading, navigate to the folder and run:

```
# Run the Review tool with a project configuration
./prismaid.exe -project your_project.toml

# Download papers from Zotero
./prismaid.exe -download-zotero zotero_config.toml

# Download papers from URL list
./prismaid.exe -download-URL paper_urls.txt

# Convert PDF files to text
./prismaid.exe -convert-pdf ./papers
```

Reminder: On Windows, you may need to allow execution if prompted by security settings.

macOS

Command: Running `prismAId` on macOS

After downloading, give execution permissions and run:

```
chmod +x prismaid
./prismaid -project your_project.toml

# Download papers from Zotero
./prismaid -download-zotero zotero_config.toml

# Convert various file formats to text
./prismaid -convert-pdf ./papers
./prismaid -convert-docx ./papers
./prismaid -convert-html ./papers
```

Warning: You may need to approve the application in **System Preferences** under **Security & Privacy** before running it.

Linux

Command: Running `prismAId` on Linux

```
chmod +x prismaid
./prismaid -project your_project.toml

# Initialize a new project configuration interactively
./prismaid -init

# Download papers from a URL list
./prismaid -download-URL paper_urls.txt
```

Tip: If you receive a "Permission denied" error, try running `chmod +x` again or executing with `sudo`.

Method 2: Go Package

If you are a Go developer, you can use `prismAId` as a Go package.

Tip: The Go package offers the most comprehensive functionality, as it is the native implementation.

Command: Installing the Go Package

```
go get "github.com/open-and-sustainable/prismaid"
```

Code: Using prismAId in Go

```
import "github.com/open-and-sustainable/prismaid"

// Run a systematic review
err := prismaid.Review(tomlConfigString)

// Download papers from Zotero
err := prismaid.DownloadZoteroPDFs(username, apiKey,
    collectionName, parentDir)

// Download from URL list
err := prismaid.DownloadURLList("path/to/urls.txt")

// Convert files to text
err := prismaid.Convert(inputDir, "pdf,docx,html")
```

Method 3: Python Package

For Python users, `prismAId` is available as a package on PyPI.

Note: The Python package works on Linux and Windows AMD64, and macOS Arm64.

Command: Installing the Python Package

```
pip install prismaid
```

Code: Using prismAId in Python

```python
import prismaid

# Run a systematic review
with open("project.toml", "r") as file:
    toml_config = file.read()
prismaid.review(toml_config)

# Download papers from Zotero
prismaid.download_zotero_pdfs("username", "api_key", "
    collection_name", "./papers")

# Download from URL list
prismaid.download_url_list("urls.txt")

# Convert files to text
prismaid.convert("./papers", "pdf,docx,html")
```

Method 4: R Package

For R users, `prismAId` is available as a package on R-universe.

Note: The R package works on Linux AMD64 and macOS Arm64.

Command: Installing the R Package

```r
install.packages("prismaid", repos = c("https://open-and-
    sustainable.r-universe.dev", "https://cloud.r-project.org
    "))
```

Code: Using prismAId in R

```r
library(prismaid)

# Run a systematic review
toml_content <- paste(readLines("project.toml"), collapse = "
    \n")
RunReview(toml_content)  # Note the capitalization

# Download papers from Zotero
DownloadZoteroPDFs("username", "api_key", "collection_name",
    "./papers")

# Download from URL list
DownloadURLList("urls.txt")

# Convert files to text
Convert("./papers", "pdf,docx,html")
```

Method 5: Julia Package

For Julia users, `prismAId` is available as a package.

Note: The Julia package works on Linux and Windows AMD64, and macOS Arm64.

Command: Installing the Julia Package

```
using Pkg
Pkg.add("PrismAId")
```

Code: Using prismAId in Julia

```
using PrismAId

# Run a systematic review
toml_config = read("project.toml", String)
PrismAId.run_review(toml_config)

# Download papers from Zotero
PrismAId.download_zotero_pdfs("username", "api_key", "
    collection_name", "./papers")

# Download from URL list
PrismAId.download_url_list("urls.txt")

# Convert files to text
PrismAId.convert("./papers", "pdf,docx,html")
```

Setting Up API Keys

To use the Review tool of `prismAId`, you need to set up API keys for at least one of the supported LLM providers.

Warning: Keep your API keys secure. Never share them in public repositories or unencrypted communications.

Obtaining API Keys

- **OpenAI**: Register at openai.com and obtain an API key from your account dashboard.
- **GoogleAI**: Create an account at aistudio.google.com and generate an API key.
- **Cohere**: Sign up at cohere.com and retrieve your API key from the dashboard.
- **Anthropic**: Register at anthropic.com and generate an API key.
- **DeepSeek**: Create an account at platform.deepseek.com and obtain an API key.

Configuring API Keys

There are two ways to configure your API keys:

Environment Variables

Set environment variables for your API keys:

Command: Setting API Key Environment Variables

```
# For OpenAI
export OPENAI_API_KEY="your-openai-api-key"

# For GoogleAI
export GOOGLEAI_API_KEY="your-googleai-api-key"

# For Cohere
export COHERE_API_KEY="your-cohere-api-key"

# For Anthropic
export ANTHROPIC_API_KEY="your-anthropic-api-key"

# For DeepSeek
export DEEPSEEK_API_KEY="your-deepseek-api-key"
```

Configuration File

Add your API keys directly in the TOML configuration file:

Configuration: API Keys in TOML

```
[project.llm.1]
provider = "OpenAI"
api_key = "your-openai-api-key"
model = "gpt-4o-mini"
temperature = 0.01
tpm_limit = 0
rpm_limit = 0
```

Tip: If both environment variables and configuration file entries are present, the configuration file values take priority.

First-Time Setup

After installation, create a new project configuration. This can be done interactively or by using the web-based configurator.

Interactive Terminal Setup

Command: Creating a New Project

Run the following command:

```
./prismaid -init
```

This prompts you with multiple questions and generates a TOML configuration file.

Reminder: Always keep a backup of your configuration file before making major modifications.

Web-Based Setup

Alternatively, use the web-based configurator available at open-and-sustainable.github.io/prismaid/review-configurator.html to create your configuration file interactively in a browser.

Configuration: Example of Initial Sections of a Configuration

To configure `prismAId`, edit the generated `your_project.toml` file:

```
[project]
name = "Use of LLM for systematic review"
author = "John Doe"
version = "1.0"

[project.configuration]
input_directory = "/path/to/txt/files"
results_file_name = "/path/to/save/results"
output_format = "json"
log_level = "low"
duplication = "no"
cot_justification = "no"
summary = "no"
```

Verifying the Installation

Check if `prismAId` is correctly installed:

Note: If you encounter issues, refer to the troubleshooting section or seek help from the community through GitHub issues or the Matrix Support Room.

Command: Verifying Installation

```
# For binaries
./prismaid --version

# For Python
python -c "import prismaid; print(prismaid.__version__)"

# For R
R -e "library(prismaid); cat(prismaid_version())"

# For Julia
julia -e "using PrismAId; println(PrismAId.version())"
```

Use in Jupyter Notebooks

When using prismAId in Jupyter notebooks with Python (versions ≤ 0.6.6), special handling may be required for interactive prompts:

Code: Using prismAId in Jupyter Notebooks

```
import pty
import os
import time
import select

def run_review_with_auto_input(input_str):
    master, slave = pty.openpty()  # Create a pseudo-terminal

    pid = os.fork()
    if pid == 0:  # Child process
        os.dup2(slave, 0)  # Redirect stdin
        os.dup2(slave, 1)  # Redirect stdout
        os.dup2(slave, 2)  # Redirect stderr
        os.close(master)
        import prismaid
        prismaid.RunReviewPython(input_str.encode("utf-8"))
        os._exit(0)

    else:  # Parent process
        os.close(slave)
        try:
            while True:
                rlist, _, _ = select.select([master], [], [],
                     5)
                if master in rlist:
                    output = os.read(master, 1024).decode("
                        utf-8", errors="ignore")
                    if not output:
                        break  # Process finished

                    print(output, end="")

                    if "Do you want to continue?" in output:
                        print("\n[SENDING INPUT: y]")
                        os.write(master, b"y\n")
                        time.sleep(1)
        finally:
            os.close(master)
            os.waitpid(pid, 0)  # Ensure the child process is
                 cleaned up

# Load your review (TOML) configuration
with open("config.toml", "r") as file:
    input_str = file.read()

# Run the review function
run_review_with_auto_input(input_str)
```

Conclusion

You have now installed `prismAId` and completed the first-time setup. The toolkit offers five different installation methods, making it accessible across various platforms and programming environments. Each component (Download, Convert, and Review) can be used independently or as part of an integrated workflow.

In the next chapter, we'll explore how to configure your project in detail to leverage all of `prismAId`'s capabilities.

Part III

Conducting a Systematic Review

Understanding Systematic Reviews

This chapter provides a foundational understanding of systematic reviews, their methodological framework, and best practices that `prismAId` is designed to support.

What is a Systematic Review?

A systematic review is a rigorous, transparent approach to synthesizing existing research literature on a specific research question. Unlike narrative reviews, which offer subjective summaries of selected studies, systematic reviews aim to identify, evaluate, and integrate findings from all relevant studies using predefined protocols.

Systematic reviews serve several critical purposes in scientific research:

- **Consolidating Knowledge**: They integrate findings across multiple studies, providing a comprehensive understanding of available evidence.
- **Identifying Gaps**: By mapping existing literature, they reveal knowledge gaps and research opportunities.
- **Minimizing Bias**: Their structured approach reduces subjective biases in literature selection and interpretation.
- **Supporting Evidence-Based Decisions**: They provide synthesized evidence to inform policy, practice, and further research.

Systematic reviews can also include meta-analyses, which statistically combine results from multiple studies to estimate overall effects.

Tip: The term "systematic" emphasizes that these reviews follow an explicit, reproducible methodology.

Note: Systematic reviews differ from other literature reviews in their methodological rigor, comprehensive search strategies, explicit inclusion criteria, and transparent reporting of methods.

Warning: Meta-analyses require statistical expertise and should only be conducted when studies are sufficiently similar in design and outcome measures.

Key Concepts & Methodology

The systematic review process follows a well-established methodology that ensures transparency and reproducibility. The widely adopted PRISMA framework (Preferred Reporting Items for Systematic Reviews and Meta-Analyses) outlines the following essential steps:

PRISMA 2020 Framework Overview

```
PRISMA 2020 Key Components:
1. Title & Abstract
2. Introduction (Rationale, Objectives)
3. Methods (Eligibility criteria, Information sources, Search
     strategy,
   Selection process, Data extraction, Study quality
       assessment)
4. Results (Study selection, Study characteristics, Risk of
    bias,
   Results of syntheses, Reporting biases)
5. Discussion (Summary, Limitations, Conclusions)
6. Other information (Registration, Protocol, Support)

Full checklist available at: prisma-statement.org
```

Pre-Review Planning

Before beginning the review, researchers must:

- **Formulate Research Questions**: Define specific, answerable questions using frameworks like PICO (Population, Intervention, Comparison, Outcome) or PEO (Population, Exposure, Outcome).

- **Develop Review Protocol**: Create a detailed plan specifying search strategies, inclusion/exclusion criteria, and analysis methods.

- **Register Protocol**: Pre-register the protocol in repositories like PROSPERO to enhance transparency and prevent duplication.

Tip: Well-formulated research questions are specific, clear, and focused. Avoid questions that are too broad ("What is known about climate change?") or too narrow ("What is the effect of a 1.5°C temperature increase on the reproduction of a specific butterfly species in northern Sweden?").

PICO Framework Example

```
Research Question: "What is the effect of cognitive
    behavioral therapy compared to medication on depression
    symptoms in adults?"

P (Population): Adults with depression
I (Intervention): Cognitive behavioral therapy
C (Comparison): Medication treatment
O (Outcome): Depression symptom reduction
```

Reminder: Always register your systematic review protocol before beginning the review process. This prevents bias and demonstrates methodological transparency.

Literature Search & Selection

The search and selection phase involves:

- **Comprehensive Search**: Implement systematic search strategies across multiple databases using carefully constructed search terms.
- **Screening**: Apply inclusion and exclusion criteria in a two-stage process: title/abstract screening followed by full-text assessment.
- **PRISMA Flow Diagram**: Document the selection process, including numbers of studies identified, screened, assessed for eligibility, and included.

PubMed Search Strategy Example

```
("climate change"[MeSH Terms] OR "global warming"[Title/
    Abstract])
AND
("agriculture"[MeSH Terms] OR "crop yield"[Title/Abstract] OR
     "food production"[Title/Abstract])
AND
("adaptation"[Title/Abstract] OR "mitigation"[Title/Abstract
    ])
AND
("2010"[Date - Publication] : "2023"[Date - Publication])
```

Warning: Relying on a single database significantly increases the risk of missing relevant studies. Research shows that even comprehensive databases like PubMed or Web of Science individually capture only 50-75% of eligible studies in many fields.

Data Extraction & Quality Assessment

Once relevant studies are identified, researchers:

- **Extract Data**: Systematically collect relevant information from each study using standardized forms.

- **Assess Quality**: Evaluate methodological rigor using tools like the Cochrane Risk of Bias Tool or GRADE (Grading of Recommendations Assessment, Development, and Evaluation).

- **Document Uncertainties**: Record ambiguities or missing information, often contacting original authors for clarification.

Data Extraction Template Example

```
Study ID: [Reference ID]
Citation: [Full citation in standard format]
Study Design: [RCT, cohort, case-control, etc.]
Population Characteristics:
  - Sample size: [n=]
  - Demographics: [age, gender, location, etc.]
  - Inclusion criteria: [as reported]
Intervention/Exposure:
  - Type: [specific details]
  - Duration: [time period]
  - Frequency: [how often applied]
Comparison/Control: [details of control group]
Outcomes:
  - Primary: [specific measure, unit]
  - Secondary: [additional measures]
Results:
  - Main findings: [effect sizes, p-values]
  - Subgroup analyses: [if applicable]
Study Quality Assessment:
  - Tool used: [e.g., Cochrane RoB, GRADE]
  - Overall rating: [Low/Moderate/High risk of bias]
  - Key limitations: [specific methodological concerns]
Notes: [Additional relevant information]
```

Tip: Create your data extraction form in digital format (e.g., spreadsheet) rather than paper. This facilitates easier data manipulation, sharing among team members, and integration with analysis software.

Synthesis & Analysis

The final analytical phase includes:

- **Narrative Synthesis**: Organize and summarize findings qualitatively, identifying patterns and relationships.

- **Quantitative Synthesis**: When appropriate, conduct meta-analysis to statistically combine results.

- **Heterogeneity Assessment**: Evaluate variations in study designs, populations, and outcomes.

- **Subgroup Analysis**: Explore how findings vary across different study characteristics or populations.

Note: Not all systematic reviews are suitable for meta-analysis. When studies use different outcome measures or have high methodological heterogeneity, narrative synthesis may be more appropriate.

R Code Example for Forest Plot Creation

```r
# Basic R code for meta-analysis and forest plot
library(meta)

# Create meta-analysis object
meta_analysis <- metagen(TE = effect_size,
                         seTE = standard_error,
                         studlab = study_name,
                         data = extracted_data,
                         sm = "SMD")

# Generate forest plot
forest(meta_analysis,
       sortvar = year,
       prediction = TRUE,
       print.tau2 = TRUE,
       print.I2 = TRUE)
```

Best Practices

Conducting high-quality systematic reviews involves adhering to several best practices:

Transparency & Reproducibility

- **Detailed Documentation**: Record all decisions, search strategies, and methods.
- **PRISMA Compliance**: Follow PRISMA guidelines for reporting.
- **Open Data**: When possible, share data extraction forms and analysis files.

Methodological Rigor

- **Comprehensive Searching**: Use multiple databases and supplementary methods like citation tracking.
- **Dual Screening**: Have at least two independent reviewers screen studies, with a third resolving disagreements.
- **Pilot Testing**: Test screening and data extraction procedures on a subset of studies.

Warning: Systematic reviews require significant time and resources. A comprehensive review typically takes 6-18 months to complete when conducted manually. Plan your timeline accordingly and consider the efficiency benefits tools like `prismAId` can provide.

Reminder: Document all exclusion decisions during screening. For transparency, maintain a list of studies excluded at the full-text review stage with specific reasons for exclusion.

Tip: Before full screening, conduct a calibration exercise where all reviewers independently screen the same 50-100 studies, then compare results to ensure consistent application of criteria. This identifies misunderstandings early and improves inter-rater reliability.

Screening Criteria Documentation

```
Inclusion Criteria:
- Population: Adults (18+) with Type 2 Diabetes
- Intervention: Digital health interventions
- Comparison: Standard care or other interventions
- Outcomes: HbA1c levels, quality of life measures
- Study Design: Randomized controlled trials
- Publication: Peer-reviewed, English language, 2010-2023

Exclusion Criteria:
- Studies focusing exclusively on Type 1 Diabetes
- Non-digital interventions
- Conference abstracts or proceedings
- Studies without control groups
```

Managing Bias

- **Publication Bias**: Search for unpublished studies and conduct funnel plot analysis when applicable.
- **Selection Bias**: Use predefined, objective inclusion criteria.
- **Confirmation Bias**: Have team members with diverse perspectives review the evidence.

Warning: Be cautious of language bias. Limiting searches to English-language publications can miss important evidence, particularly in fields with significant international research or regional focus.

Code for Publication Bias Assessment

```r
# Funnel plot and Egger's test in R
library(meta)

# Create funnel plot
funnel(meta_analysis,
       studlab = TRUE,
       contour = TRUE)

# Perform Egger's test for publication bias
metabias(meta_analysis, method = "linreg")
```

Team Collaboration

- **Multidisciplinary Teams**: Include subject matter experts, methodologists, and librarians.
- **Regular Calibration**: Conduct ongoing discussions to ensure consistent application of criteria.
- **Clear Roles**: Define responsibilities for each team member.

Note: Consulting with a research librarian can significantly improve search strategy quality. Their expertise in database syntax and controlled vocabulary (e.g., MeSH terms) helps ensure comprehensive literature identification.

Reminder: The most robust systematic reviews often involve teams rather than individual researchers. If working alone, consider consulting

The Role of Technology in Systematic Reviews

Traditional systematic review methods are resource-intensive and time-consuming. Recent technological advances have created opportunities to streamline the process while maintaining methodological rigor:

- **Literature Screening**: Machine learning algorithms can help prioritize relevant studies.
- **Data Extraction**: Natural language processing can identify key information from texts.
- **Evidence Synthesis**: Automated tools can assist in organizing and analyzing extracted data.
- **Protocol Management**: Specialized software can guide teams through the systematic review workflow.

`prismAId` represents a significant evolution in this technological landscape, leveraging advanced large language models to assist with multiple phases of the systematic review process.

Tip: Even with technological assistance, allocate time for pilot testing and validation. Test `prismAId` with a small subset of your literature before processing your entire dataset.

Benefits of Technology-Assisted Reviews

Technology-assisted systematic reviews offer several advantages:

- **Increased Efficiency**: Reducing time required for screening and data extraction.
- **Enhanced Consistency**: Applying criteria uniformly across all studies.
- **Improved Scalability**: Managing larger volumes of literature.
- **Better Reproducibility**: Creating structured, traceable processes.

Typical Efficiency Gains with Technology

Task	Traditional Method	Technology-Assisted	Potential Time Saved
Title/Abstract Screening	~1-2 min per reference	~30-60 sec per reference	50-75%
Data Extraction	~20-40 min per study	~5-15 min per study	60-80%
Evidence Synthesis	Days to weeks	Hours to days	70-90%

Conclusion

Systematic reviews are cornerstone methods for evidence synthesis across scientific disciplines. By following rigorous methodologies and best practices, researchers can produce high-quality reviews that reliably inform decision-making and future research directions.

`prismAId` is designed to support this methodology by automating labor-intensive aspects of the review process while maintaining the methodological rigor that makes systematic reviews valuable. In the following chapters, we'll explore how to harness `prismAId`'s capabilities to conduct efficient, protocol-driven systematic reviews.

Warning: While AI tools like `prismAId` significantly enhance efficiency, human oversight remains essential. AI systems may miss nuanced information or misinterpret specialized terminology. Always validate AI-extracted data against source documents.

Tip: Start small and scale up. If you're new to systematic reviews, consider conducting a scoping review or rapid review on a narrow topic before undertaking a comprehensive systematic review.

Reminder: Even the most sophisticated tools can't replace critical thinking. Use `prismAId` to handle repetitive tasks, but apply your subject expertise to interpret findings in context and derive meaningful conclusions.

Step-by-Step Guide to Conducting a Systematic Review

This chapter provides a comprehensive, practical walkthrough of conducting a systematic review using `prismAId`, from initial setup to final export of findings. Follow these steps sequentially to complete your review efficiently.

Setting Up a Project

Before diving into the technical aspects, proper project planning is essential for a successful systematic review.

Define Your Research Question

1. Formulate a clear, focused research question using a framework like PICO.

2. Define the scope of your review (time period, study types, etc.).

3. Determine what specific information you need to extract from each paper.

Tip: Well-defined research questions lead to more precise information extraction. Be specific about what you want to learn from the literature.

Create a Project Directory Structure

Command: Creating a Project Directory Structure

```
# Create main project directory
mkdir my_systematic_review

# Create subdirectories for different stages
mkdir my_systematic_review/papers_pdf
mkdir my_systematic_review/papers_txt
mkdir my_systematic_review/config
mkdir my_systematic_review/results
```

Reminder: A well-organized directory structure makes it easier to track your workflow and prevents confusion between original documents and processed files.

Configure Your Project

1. Generate a basic configuration file using the `prismAId` initializer.

2. Customize the configuration file to match your research questions.

Command: Initializing a Project Configuration

```
# Navigate to your project directory
cd my_systematic_review

# Initialize a new configuration file
./prismaid -init
```

Note: Alternatively, you can use the web-based configurator at open-and-sustainable.github.io/prismaid/review-configurator.html to create your configuration file with a user-friendly interface.

Configuration: Essential Project Settings

```
[project]
name = "Effects of Climate Change on Agricultural Yields"
author = "Jane Researcher"
version = "1.0"

[project.configuration]
input_directory = "/home/user/my_systematic_review/papers_txt
    "
results_file_name = "/home/user/my_systematic_review/results/
    findings"
output_format = "csv"
log_level = "medium"
duplication = "no"
cot_justification = "yes"
summary = "yes"
```

Warning: Always use absolute paths in your configuration file to prevent path-related errors. Relative paths can cause issues when running `prismAId` from different directories.

Configure LLM Settings

1. Obtain API key(s) from your preferred LLM provider(s).

2. Set up environment variables or add keys to your configuration file.

3. Define model settings based on your needs and budget.

Configuration: LLM Provider Setup

```
[project.llm.1]
provider = "OpenAI"
api_key = "" # Leave empty to use environment variable
model = "gpt-4o-mini"
temperature = 0.01
tpm_limit = 0
rpm_limit = 0
```

Tip: Using a lower temperature setting (0.01-0.1) produces more consistent results, which is generally preferable for systematic reviews where reproducibility is important.

Importing and Managing Data

Once your project is set up, you'll need to gather and prepare your literature for analysis.

Literature Acquisition using the Download Tool

Option 1: Downloading from Zotero

1. Prepare your Zotero collection with all papers for your systematic review.

2. Obtain your Zotero user ID and API key.

3. Create a Zotero configuration file or use command line options.

Configuration: Zotero Download Setup

```
# zotero_config.toml
user = "123456789"  # Your Zotero user ID
api_key = "AbCdEfGhIjKlMnOpQrStUv"  # Your Zotero API key
group = "Climate Change Research/Agriculture Papers"  # Your
    collection path
```

Command: Downloading Papers from Zotero

```
# Navigate to your project directory
cd my_systematic_review

# Download papers to the papers_pdf directory
./prismaid -download-zotero config/zotero_config.toml -o
    papers_pdf
```

Note: The collection path in Zotero follows a filesystem-like format. For example, "Parent Collection/Sub Collection" or "Group Name/Collection Name" for group libraries.

Option 2: Downloading from URL Lists

1. Create a text file with one URL per line for papers you want to download.

2. Run the URL download command to fetch all papers.

Example URL List File: paper_urls.txt

```
https://arxiv.org/pdf/2303.08774.pdf
https://www.science.org/doi/pdf/10.1126/science.1236498
https://www.nature.com/articles/s41586-021-03819-2.pdf
```

Command: Downloading Papers from URL List

```
# Navigate to your project directory
cd my_systematic_review

# Download papers to the papers_pdf directory
./prismaid -download-URL config/paper_urls.txt -o papers_pdf
```

Warning: Some publishers may restrict automatic downloads. Ensure you have appropriate access rights to all papers before attempting to download them.

Document Conversion using the Convert Tool

After downloading your papers, you need to convert them to plain text format for analysis.

Command: Converting PDF Files to Text

```
# Navigate to your project directory
cd my_systematic_review

# Convert all PDFs in papers_pdf directory to text files in
    papers_txt
./prismaid -convert-pdf papers_pdf -o papers_txt
```

Command: Converting Multiple File Types

```
# For DOCX files
./prismaid -convert-docx papers_docx -o papers_txt

# For HTML files
./prismaid -convert-html papers_html -o papers_txt
```

Reminder: Always manually check a sample of converted files to ensure the conversion quality is acceptable. PDFs with complex layouts or scanned images may not convert perfectly.

Organizing Your Literature Collection

1. Review the converted text files for quality and completeness.
2. Remove or fix any problematic conversions.
3. Optionally, rename files for better organization and tracking.

Tip: Consider removing unnecessary sections from converted texts, such as references or acknowledgments, to reduce token usage and improve extraction accuracy.

Running Analyses

With your literature prepared, you're ready to configure and execute the systematic review.

Finalizing Review Configuration

1. Define your prompt structure to instruct the LLM.
2. Specify the review items (information to extract) in your configuration.

Configuration: Prompt Structure

```
[prompt]
persona = "You are an expert agricultural scientist
    conducting a systematic review on climate change impacts.
    "
task = "Extract specific information about how climate change
     affects crop yields from the scientific paper text
    provided."
expected_result = "You should output a JSON object with key
    findings on crop types, climate factors, and measured
    impacts."
definitions = "'Crop yield' refers to the quantity of
    agricultural output harvested per unit of land area."
example = "For example, if the paper states 'wheat yields
    decreased by 5.2% per degree Celsius increase', report '
    wheat' as crop_type, 'temperature increase' as
    climate_factor, and '-5.2% per °C' as yield_impact."
failsafe = "If information on a specific field is not
    provided in the document, respond with an empty string
    value."
```

Configuration: Review Items Structure

```
[review.1]
key = "crop_type"
values = ["wheat", "rice", "maize", "soybean", "barley", "
    other", "multiple", ""]

[review.2]
key = "climate_factor"
values = ["temperature increase", "precipitation change", "
    extreme weather", "CO2 levels", "multiple factors", "
    other", ""]

[review.3]
key = "yield_impact"
values = [""]

[review.4]
key = "adaptation_strategies_discussed"
values = ["yes", "no"]

[review.5]
key = "study_timeframe"
values = ["historical", "current", "future projection", "
    mixed", ""]
```

Tip: When extracting numerical data (like percentages or measurements), use empty value arrays ‘[""]‘ to allow the model to extract the exact values rather than categorizing them.

Executing the Review

Command: Running the Systematic Review

```
# Navigate to your project directory
cd my_systematic_review

# Run the review with your configuration file
./prismaid -project config/climate_yield_review.toml
```

Note: For large reviews, consider running a test on a small subset of papers first to validate your configuration before processing all documents.

Monitoring Progress

1. Monitor the console output for progress updates.
2. Check log files if you’ve enabled detailed logging.
3. Address any errors or warnings that appear during processing.

Command: Running with Detailed Logging

```
# First modify your config file to set log_level to "high"
# Then run the review
./prismaid -project config/climate_yield_review.toml
```

Warning: Long-running reviews may encounter API rate limits. Use the ‘tpm_limit‘ and ‘rpm_limit‘ settings in your configuration to manage API usage and avoid interruptions.

Interpreting Results

Once your review is complete, you'll need to analyze and understand the extracted information.

Understanding Output Files

1. Locate your results file (CSV or JSON format).

2. Open supplementary files like justifications or summaries if enabled.

3. Understand the structure of the output data.

Example CSV Result Structure

```
filename,crop_type,climate_factor,yield_impact,
    adaptation_strategies_discussed,study_timeframe
paper1.txt,wheat,temperature increase,-6.4% per °C,yes,future
     projection
paper2.txt,rice,precipitation change,-3.2% per 10% rainfall
    decrease,yes,historical
paper3.txt,multiple,multiple factors,varied by region and
    crop,yes,mixed
```

Reminder: If you enabled 'cot_justification', each paper will have a corresponding justification file that explains the reasoning behind the extracted information. These can be invaluable for validation and deeper understanding.

Validating Extraction Quality

1. Randomly select a subset of papers for manual validation.

2. Compare the extracted information with the original papers.

3. Note any discrepancies or patterns in extraction errors.

Tip: Aim to manually validate at least 10-20% of your papers to ensure extraction quality. Focus particularly on papers with unusual or unexpected results.

Analyzing Patterns and Trends

1. Import your results into analysis software (R, Python, Excel, etc.).

2. Perform descriptive statistics on your extracted data.

3. Identify patterns, relationships, and outliers.

Example R Code for Basic Analysis

```r
# Load libraries
library(tidyverse)

# Read results
results <- read_csv("results/findings.csv")

# Basic summary
summary(results)

# Count papers by crop type
results %>%
  count(crop_type) %>%
  arrange(desc(n))

# Analyze yield impacts by climate factor
results %>%
  filter(climate_factor %in% c("temperature increase", "
      precipitation change")) %>%
  group_by(climate_factor, crop_type) %>%
  summarize(n_studies = n()) %>%
  arrange(desc(n_studies))
```

Note: Extracting numerical data from free-text fields like 'yield_impact' may require additional processing before quantitative analysis.

Exporting Findings

The final step is to prepare your findings for presentation, publication, or further analysis.

Generating Summary Tables

1. Create summary tables of key findings.

2. Format the data appropriately for your target audience.

3. Include relevant metadata about your review process.

Example Python Code for Table Generation

```python
import pandas as pd
import matplotlib.pyplot as plt

# Load results
results = pd.read_csv("results/findings.csv")

# Create pivot table
pivot = pd.pivot_table(
    results,
    index="crop_type",
    columns="climate_factor",
    values="filename",
    aggfunc="count",
    fill_value=0
)

# Save summary table
pivot.to_csv("results/summary_table.csv")
pivot.to_excel("results/summary_table.xlsx")

# Create visualization
pivot.plot(kind="bar", figsize=(12, 8))
plt.title("Number of Studies by Crop Type and Climate Factor"
    )
plt.tight_layout()
plt.savefig("results/crop_climate_summary.png", dpi=300)
```

Tip: Include both raw data and processed summary tables in your publications to enhance transparency and reproducibility.

Documenting Your Methodology

1. Document your complete review methodology.
2. Include details about:
    - Search strategy and sources
    - Inclusion/exclusion criteria
    - `prismAId` configuration and settings
    - Validation procedures
    - Analysis methods
3. Save your configuration files alongside your results.

Reminder: For publication, include your full `prismAId` configuration file (with API keys removed) in supplementary materials to enhance reproducibility.

Visualizing Results

1. Create appropriate visualizations based on your data type.
2. Consider common visualization types:
    - Bar charts for categorical comparisons

- Forest plots for effect sizes
- Heat maps for multidimensional relationships
- Network diagrams for concept relationships

3. Ensure visualizations accurately represent your findings.

Warning: Be cautious about drawing causal conclusions from your systematic review findings, especially when AI-assisted tools are used for extraction. Clearly acknowledge limitations in your reporting.

Preparing for Publication

1. Format your findings according to journal or conference requirements.
2. Follow PRISMA or other appropriate reporting guidelines.
3. Create a PRISMA flow diagram documenting the review process.
4. Properly cite `prismAId` in your methodology section.

Example Citation for prismAId

```
For the systematic information extraction, we used prismAId (
    Boero, 2024),
an open-source AI-assisted systematic review toolkit. The
    review was
conducted using [model name] with a temperature setting of [
    value] and
validation was performed on [percentage]% of the included
    studies.

Reference:
Boero, R. (2024). prismAId - Open Science AI Tools for
    Systematic,
Protocol-Based Literature Reviews. Zenodo.
https://doi.org/10.5281/zenodo.11210796
```

Note: Some journals may have specific requirements or guidelines for reporting AI-assisted analyses. Check with your target journal for any specific policies.

Conclusion

You have now completed a systematic review using `prismAId`. By following this step-by-step process, you've:

- Set up a structured, reproducible review project
- Gathered and prepared literature systematically
- Configured and executed an AI-assisted information extraction
- Analyzed and validated the extracted information
- Prepared your findings for dissemination

The combination of human expertise and AI assistance enables more comprehensive and efficient reviews while maintaining methodological rigor. The next chapter will explore advanced features of `prismAId` that can further enhance your systematic review capabilities.

Reminder: Systematic reviews are iterative processes. Based on your findings, you may decide to refine your research question, adjust your extraction parameters, or expand your literature search. `prismAId` makes these iterations more efficient than traditional methods.

Part IV

Advanced Features

Advanced Features: Ensemble Reviews

This chapter explores ensemble reviews, one of the most powerful advanced features of `prismAId`, enabling users to leverage multiple AI models simultaneously for enhanced reliability and insight.

Understanding Ensemble Reviews

Ensemble reviews represent a sophisticated approach to systematic literature analysis where multiple large language models (LLMs) are used to extract information from the same document set. This methodology parallels the traditional practice of having multiple human reviewers assess the same literature.

Concept and Benefits

- **Definition**: An ensemble review utilizes two or more AI models, either from the same provider or across different providers, to independently extract information from each document.

- **Theoretical Foundation**: The approach is grounded in ensemble learning theory, where multiple algorithms collectively produce more accurate and robust results than individual algorithms.

Tip: Ensemble reviews are particularly valuable when conducting high-stakes systematic reviews where accuracy and reliability are critical, such as in clinical guideline development or policy formation.

Configuration: Simple Ensemble with Two Models

```
[project.llm.1]
provider = "OpenAI"
api_key = ""
model = "gpt-4o-mini"
temperature = 0.01
tpm_limit = 0
rpm_limit = 0

[project.llm.2]
provider = "OpenAI"
api_key = ""
model = "gpt-4o"
temperature = 0.01
tpm_limit = 0
rpm_limit = 0
```

Key Advantages

Ensemble reviews offer several significant benefits:

- **Uncertainty Quantification**: Variations in model outputs highlight where information may be ambiguous or open to interpretation.
- **Increased Confidence**: Strong agreement across models suggests higher reliability of the extracted information.
- **Bias Reduction**: Different models may have different biases; using multiple models helps identify and mitigate these biases.
- **Robustness to Model Limitations**: Different models excel at different tasks; using multiple models compensates for individual weaknesses.

Note: Research in AI-assisted systematic reviews suggests that ensemble approaches can reduce error rates by 20-30% compared to single-model extractions, particularly for complex or ambiguous information.

Configuring Multi-Model Ensembles

`prismAId` offers exceptional flexibility in configuring ensemble reviews, allowing you to combine models from the same provider, different providers, or a mix of both.

Using Multiple Models from a Single Provider

This approach allows you to compare different models with varying capabilities from the same provider:

Configuration: Ensemble with Multiple OpenAI Models

```
[project.llm.1]
provider = "OpenAI"
api_key = ""
model = "gpt-3.5-turbo"
temperature = 0.01
tpm_limit = 0
rpm_limit = 0

[project.llm.2]
provider = "OpenAI"
api_key = ""
model = "gpt-4o-mini"
temperature = 0.01
tpm_limit = 0
rpm_limit = 0

[project.llm.3]
provider = "OpenAI"
api_key = ""
model = "gpt-4o"
temperature = 0.01
tpm_limit = 0
rpm_limit = 0
```

Tip: When using multiple models from the same provider, consider including a range of model sizes to balance cost, speed, and accuracy. For example, include both gpt-3.5-turbo and gpt-4o to compare a faster, more economical model with a more sophisticated one.

Cross-Provider Ensemble Configuration

Using models from different providers can be particularly valuable, as these models may have been trained on different datasets and using different architectures:

Configuration: Cross-Provider Ensemble Example

```
[project.llm.1]
provider = "OpenAI"
api_key = ""
model = "gpt-4o-mini"
temperature = 0.01
tpm_limit = 0
rpm_limit = 0

[project.llm.2]
provider = "GoogleAI"
api_key = ""
model = "gemini-1.5-flash"
temperature = 0.01
tpm_limit = 0
rpm_limit = 0

[project.llm.3]
provider = "Anthropic"
api_key = ""
model = "claude-3-haiku"
temperature = 0.01
tpm_limit = 0
rpm_limit = 0
```

Warning: When using models from multiple providers, ensure you have valid API keys for each provider configured either in your TOML file or as environment variables. Missing or invalid keys will cause the review to fail for that provider.

Comprehensive Five-Provider Ensemble

For maximum diversity and robustness, you can configure an ensemble using models from all supported providers:

Configuration: Full Five-Provider Ensemble

```
[project.llm.1]
provider = "OpenAI"
api_key = ""
model = "gpt-4o-mini"
temperature = 0.01
tpm_limit = 0
rpm_limit = 0

[project.llm.2]
provider = "GoogleAI"
api_key = ""
model = "gemini-1.5-flash"
temperature = 0.01
tpm_limit = 0
rpm_limit = 0

[project.llm.3]
provider = "Cohere"
api_key = ""
model = "command-r"
temperature = 0.01
tpm_limit = 0
rpm_limit = 0

[project.llm.4]
provider = "Anthropic"
api_key = ""
model = "claude-3-haiku"
temperature = 0.01
tpm_limit = 0
rpm_limit = 0

[project.llm.5]
provider = "DeepSeek"
api_key = ""
model = "deepseek-chat"
temperature = 0.01
tpm_limit = 0
rpm_limit = 0
```

Reminder: Pay close attention to the temperature settings in ensemble reviews. Lower temperatures (0.01-0.1) produce more deterministic outputs, making it easier to compare responses across models.

Analyzing Ensemble Results

After running an ensemble review, you'll need specialized approaches to analyze and interpret the results from multiple models.

Output Format and Structure

When running an ensemble review, `prismAId` creates separate result files for each model used:

Example Output Files from Ensemble Review

```
results_OpenAI_gpt-4o-mini.csv
results_GoogleAI_gemini-1.5-flash.csv
results_Anthropic_claude-3-haiku.csv
results_ensemble_summary.csv
```

The individual model files contain the standard extraction results, while the ensemble summary file contains consensus metrics and comparison data.

Quantifying Model Agreement

To analyze the level of agreement between models:

R Code for Analyzing Model Agreement

```
library(tidyverse)

# Load results from different models
model1 <- read_csv("results_OpenAI_gpt-4o-mini.csv")
model2 <- read_csv("results_GoogleAI_gemini-1.5-flash.csv")
model3 <- read_csv("results_Anthropic_claude-3-haiku.csv")

# Join datasets
comparison <- model1 %>%
  select(filename, crop_type, climate_factor) %>%
  rename(crop_type_model1 = crop_type,
         climate_factor_model1 = climate_factor) %>%
  left_join(
    model2 %>%
      select(filename, crop_type, climate_factor) %>%
      rename(crop_type_model2 = crop_type,
             climate_factor_model2 = climate_factor),
    by = "filename"
  ) %>%
  left_join(
    model3 %>%
      select(filename, crop_type, climate_factor) %>%
      rename(crop_type_model3 = crop_type,
             climate_factor_model3 = climate_factor),
    by = "filename"
  )

# Calculate agreement for categorical variables
comparison <- comparison %>%
  mutate(
    crop_type_agreement = case_when(
      crop_type_model1 == crop_type_model2 & crop_type_model2
           == crop_type_model3 ~ "full_agreement",
      crop_type_model1 == crop_type_model2 | crop_type_model1
           == crop_type_model3 | crop_type_model2 ==
          crop_type_model3 ~ "partial_agreement",
      TRUE ~ "no_agreement"
    ),
    climate_factor_agreement = case_when(
      climate_factor_model1 == climate_factor_model2 &
          climate_factor_model2 == climate_factor_model3 ~ "
          full_agreement",
      climate_factor_model1 == climate_factor_model2 |
          climate_factor_model1 == climate_factor_model3 |
          climate_factor_model2 == climate_factor_model3 ~ "
          partial_agreement",
      TRUE ~ "no_agreement"
    )
  )

# Summarize agreement levels
agreement_summary <- comparison %>%
  summarize(
    crop_type_full_agreement = mean(crop_type_agreement == "
        full_agreement"),
    crop_type_partial_agreement = mean(crop_type_agreement ==
         "partial_agreement"),
    crop_type_no_agreement = mean(crop_type_agreement == "
        no_agreement"),
    climate_factor_full_agreement = mean(
        climate_factor_agreement == "full_agreement"),
    climate_factor_partial_agreement = mean(
        climate_factor_agreement == "partial_agreement"),
    climate_factor_no_agreement = mean(
        climate_factor_agreement == "no_agreement")
  )

print(agreement_summary)
```

Tip: For categorical variables, Cohen's kappa or Fleiss' kappa (for more than two models) can provide a more sophisticated measure of inter-model agreement than simple percentage agreement.

Visualizing Ensemble Results

Visualizations can help identify patterns of agreement and disagreement:

Python Code for Visualizing Ensemble Agreement

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load results from different models
model1 = pd.read_csv("results_OpenAI_gpt-4o-mini.csv")
model2 = pd.read_csv("results_GoogleAI_gemini-1.5-flash.csv")
model3 = pd.read_csv("results_Anthropic_claude-3-haiku.csv")

# Create comparison dataframe for a specific variable
comparison = pd.DataFrame({
    'filename': model1['filename'],
    'OpenAI': model1['crop_type'],
    'GoogleAI': model2['crop_type'],
    'Anthropic': model3['crop_type']
})

# Count agreements for each paper
comparison['agreement_count'] = comparison.apply(
    lambda row: len(set([row['OpenAI'], row['GoogleAI'], row[
        'Anthropic']])),
    axis=1
)

# Create a heatmap of disagreements
# Convert to wide format with models as columns and papers as
     rows
heatmap_data = comparison.set_index('filename')[['OpenAI', '
    GoogleAI', 'Anthropic']]

# Create categorical mapping for visualization
unique_values = sorted(list(set(
    heatmap_data['OpenAI'].tolist() +
    heatmap_data['GoogleAI'].tolist() +
    heatmap_data['Anthropic'].tolist()
)))
value_map = {value: i for i, value in enumerate(unique_values
    )}

# Apply mapping
for col in heatmap_data.columns:
    heatmap_data[col] = heatmap_data[col].map(value_map)

# Create heatmap
plt.figure(figsize=(12, len(heatmap_data) * 0.3))
sns.heatmap(heatmap_data, cmap='viridis', yticklabels=True)
plt.title('Model Agreement Visualization (Crop Type)')
plt.tight_layout()
plt.savefig('ensemble_agreement_heatmap.png', dpi=300)

# Create agreement distribution chart
agreement_counts = comparison['agreement_count'].value_counts
    ().sort_index()
plt.figure(figsize=(10, 6))
agreement_counts.plot(kind='bar')
plt.title('Distribution of Model Agreement')
plt.xlabel('Number of Unique Answers (1 = Full Agreement)')
plt.ylabel('Number of Papers')
plt.tight_layout()
plt.savefig('agreement_distribution.png', dpi=300)
```

Note: For numerical data extraction (like percentages or measurements), consider using scatterplots to visualize the correlation between values extracted by different models, or boxplots to show the distribution of values across models.

Decision Strategies for Conflicting Results

When models disagree on extracted information, several strategies can be employed:

- **Majority Voting**: Use the value that the majority of models agree on.
- **Weighted Voting**: Assign higher weight to more capable models (e.g., GPT-4 over GPT-3.5).
- **Conservative Approach**: For papers with significant disagreement, flag for manual review.
- **Model-Specific Trust**: For certain types of information, one model may consistently outperform others.

Python Code for Majority Voting

```
import pandas as pd
from collections import Counter

# Load results from different models
model1 = pd.read_csv("results_OpenAI_gpt-4o-mini.csv")
model2 = pd.read_csv("results_GoogleAI_gemini-1.5-flash.csv")
model3 = pd.read_csv("results_Anthropic_claude-3-haiku.csv")

# Prepare a consolidated results dataframe
consolidated = pd.DataFrame({'filename': model1['filename']})

# Apply majority voting for each extracted field
for field in ['crop_type', 'climate_factor', '
    adaptation_strategies_discussed']:
    consolidated[field] = [
        Counter([m1, m2, m3]).most_common(1)[0][0]
        for m1, m2, m3 in zip(
            model1[field].fillna(''),
            model2[field].fillna(''),
            model3[field].fillna('')
        )
    ]

    # For fields where all models disagree, mark for review
    consolidated[f'{field}_review_needed'] = [
        len(set([m1, m2, m3])) == 3
        for m1, m2, m3 in zip(
            model1[field].fillna(''),
            model2[field].fillna(''),
            model3[field].fillna('')
        )
    ]

# For numerical fields, take the median
for field in ['yield_impact_percentage']:
    try:
        # Convert to numeric, errors='coerce' will convert
            non-numeric to NaN
        vals1 = pd.to_numeric(model1[field], errors='coerce')
        vals2 = pd.to_numeric(model2[field], errors='coerce')
        vals3 = pd.to_numeric(model3[field], errors='coerce')

        # Calculate median
        consolidated[field] = [
            pd.Series([v1, v2, v3]).median()
            for v1, v2, v3 in zip(vals1, vals2, vals3)
        ]

        # Calculate standard deviation to identify high
            variance
        consolidated[f'{field}_std'] = [
            pd.Series([v1, v2, v3]).std()
            for v1, v2, v3 in zip(vals1, vals2, vals3)
        ]

        # Flag for review if standard deviation is high
        consolidated[f'{field}_review_needed'] = consolidated
            [f'{field}_std'] > 2.0
    except:
        # Handle cases where the field might not exist or can
            't be processed
        pass

# Save consolidated results
consolidated.to_csv("results_consolidated.csv", index=False)
```

Warning: Be cautious when using majority voting for numerical values, as this may lead to misleading results. For numerical data, consider using the median or mean, and examine the standard deviation to identify high-variance cases.

Case Studies in Ensemble Reviews

To illustrate the practical application of ensemble reviews, let's examine some case scenarios and best practices.

Case 1: Basic Verification Ensemble

Configuration: Verification Ensemble

```
# A simple two-model ensemble using different providers
# for basic verification of extraction results

[project.llm.1]
provider = "OpenAI"
api_key = ""
model = "gpt-3.5-turbo"  # More economical option
temperature = 0.01
tpm_limit = 0
rpm_limit = 0

[project.llm.2]
provider = "Anthropic"
api_key = ""
model = "claude-3-haiku"  # Similar capability level
temperature = 0.01
tpm_limit = 0
rpm_limit = 0
```

Use Case: This configuration is suitable for routine systematic reviews where you want a basic cross-check between providers without significantly increasing costs. It uses comparably efficient models from different providers.

Expected Outcome: When these models agree, you can have higher confidence in the extraction. When they disagree, you might flag those specific data points for manual review.

Case 2: Tiered Capability Ensemble

Configuration: Tiered Capability Ensemble

```
# A three-model ensemble using progressively more capable
    models
# from the same provider for comparative analysis

[project.llm.1]
provider = "OpenAI"
api_key = ""
model = "gpt-3.5-turbo"  # Base level
temperature = 0.01
tpm_limit = 0
rpm_limit = 0

[project.llm.2]
provider = "OpenAI"
api_key = ""
model = "gpt-4o-mini"  # Mid level
temperature = 0.01
tpm_limit = 0
rpm_limit = 0

[project.llm.3]
provider = "OpenAI"
api_key = ""
model = "gpt-4o"  # Top level
temperature = 0.01
tpm_limit = 0
rpm_limit = 0
```

Use Case: This approach is useful for understanding how model capability affects extraction quality. It can help determine if investing in more advanced models provides meaningful improvements for your specific extraction tasks.

Expected Outcome: By comparing results across models of increasing capability, you can determine the minimum model level needed for reliable extraction, potentially optimizing future costs.

Case 3: Comprehensive Cross-Provider Ensemble

Configuration: Comprehensive Cross-Provider Ensemble

```
# A five-provider ensemble using top-tier models from each
    provider
# for maximum reliability in high-stakes reviews

[project.llm.1]
provider = "OpenAI"
api_key = ""
model = "gpt-4o"
temperature = 0.01
tpm_limit = 0
rpm_limit = 0

[project.llm.2]
provider = "GoogleAI"
api_key = ""
model = "gemini-1.5-pro"
temperature = 0.01
tpm_limit = 0
rpm_limit = 0

[project.llm.3]
provider = "Cohere"
api_key = ""
model = "command-r-plus"
temperature = 0.01
tpm_limit = 0
rpm_limit = 0

[project.llm.4]
provider = "Anthropic"
api_key = ""
model = "claude-3-opus"
temperature = 0.01
tpm_limit = 0
rpm_limit = 0

[project.llm.5]
provider = "DeepSeek"
api_key = ""
model = "deepseek-chat"
temperature = 0.01
tpm_limit = 0
rpm_limit = 0
```

Use Case: This configuration is ideal for high-stakes systematic reviews such as those informing clinical guidelines, policy decisions, or major meta-analyses where maximum reliability is essential.

Expected Outcome: This approach provides the highest confidence in results through diverse model architectures and training data. Points of unanimous agreement across all five providers suggest very high reliability.

Tip: Start with a small test dataset when using comprehensive ensembles like this to estimate costs and review time before committing to your full dataset.

Best Practices for Ensemble Reviews

Based on case studies and practical experience, here are recommended best practices:

- **Start Small**: Test your ensemble configuration on a small subset (5-10 papers) to identify any issues before running the full review.
- **Match Complexity to Need**: Use simpler ensembles for routine reviews and more comprehensive ensembles for high-stakes reviews.
- **Control Variables**: Keep temperature settings consistent across models to ensure fair comparisons.
- **Budget Appropriately**: Ensemble reviews will multiply your API costs by the number of models used; plan accordingly.
- **Document Model Versions**: Note the specific model versions used, as providers regularly update their models.
- **Analyze Disagreements**: Patterns in model disagreements often reveal ambiguities in your prompt or in the literature itself.

Reminder: When reporting results from ensemble reviews in publications, clearly state which models were used, how disagreements were resolved, and what level of agreement was observed for key findings.

Advanced Features: Debugging, Cost Management & Integration

Building on our exploration of ensemble reviews, this chapter covers additional advanced features of `prismAId` that can enhance your systematic review process: debugging techniques, cost management strategies, advanced prompt engineering, and workflow integration.

Debugging and Validation Techniques

`prismAId` offers several advanced debugging features that can help identify and resolve issues in your systematic review process.

Log Levels and Debugging Output

The `log_level` parameter in your configuration file controls the verbosity of debugging information:

Configuration: Log Level Settings

```
[project.configuration]
log_level = "high"  # Options: "low", "medium", "high"
```

Each level provides different information:

- **Low**: Minimal information, only essential status updates
- **Medium**: Detailed process information printed to the console

- **High**: Comprehensive logs saved to a file, including all API interactions

Tip: When troubleshooting issues, start by setting the log level to "medium" to get more detailed console output. If the issue persists, switch to "high" for complete logging to file.

Command: Running with High Log Level

```
# First set log_level to "high" in your config file
./prismaid -project your_project.toml

# Check the log file created in your project directory
cat prismaid_log_YYYY-MM-DD.log
```

Duplication for Consistency Testing

The duplication feature runs the same review twice on identical inputs, allowing you to assess the consistency of model responses:

Configuration: Enabling Duplication

```
[project.configuration]
duplication = "yes"  # Options: "yes", "no"
```

Warning: Enabling duplication will double your API usage and costs. Use this feature selectively for testing or when consistency validation is critical.

The duplication process: 1. Creates temporary copies of your input files 2. Processes them as a separate batch 3. Compares results for consistency 4. Cleans up temporary files

Python Code for Analyzing Duplication Results

```python
import pandas as pd

# Load original and duplicate results
original = pd.read_csv("results_original.csv")
duplicate = pd.read_csv("results_duplicate.csv")

# Compare results
comparison = pd.DataFrame({
    'filename': original['filename'],
})

# For each extracted field, calculate consistency
for field in ['crop_type', 'climate_factor', 'yield_impact']:
    comparison[f'{field}_consistent'] = original[field] ==
        duplicate[field]

# Calculate overall consistency
field_columns = [col for col in comparison.columns if col.
    endswith('_consistent')]
comparison['overall_consistency'] = comparison[field_columns
    ].mean(axis=1)

# Summary statistics
consistency_summary = {
    'overall_consistency': comparison['overall_consistency'].
        mean(),
}

for field in field_columns:
    consistency_summary[field] = comparison[field].mean()

print("Consistency Summary:")
for key, value in consistency_summary.items():
    print(f"{key}: {value:.2%}")

# Identify inconsistent papers
inconsistent_papers = comparison[comparison['
    overall_consistency'] < 1.0]
print(f"\nPapers with inconsistencies: {len(
    inconsistent_papers)}")
print(inconsistent_papers[['filename', 'overall_consistency'
    ]])
```

Chain-of-Thought Justification

The Chain-of-Thought (CoT) justification feature provides visibility into the model's reasoning process:

Configuration: Enabling CoT Justification

```toml
[project.configuration]
cot_justification = "yes"  # Options: "yes", "no"
```

When enabled, this creates a separate .txt file for each processed document containing:

- The model's reasoning for each extracted data point
- Key passages from the document that influenced the extraction
- Any uncertainty or alternative interpretations considered

Example CoT Justification Output

```
Paper: climate_impact_2022.txt

Extraction Justification:
--------------------------

- crop_type: "wheat" - The paper specifically focuses on
    wheat production.
  This is stated in the abstract: "This study examines the
      impact of
  temperature rises on wheat yields across 17 major producing
       regions"
  and is the primary crop discussed throughout.

- climate_factor: "temperature increase" - The primary
    climate factor
  analyzed is rising temperatures. The methodology section
      states:
  "We analyzed the response of wheat yields to temperature
      increases
  ranging from +1°C to +4°C." While precipitation is
      mentioned, it's
  not the main focus of analysis.

- yield_impact: "-5.7% per °C" - This value appears in the
    results
  section: "Our models predict an average yield reduction of
      5.7%
  for each degree Celsius increase in global mean temperature
      ."
  This appears to be the central finding of the paper.

- adaptation_strategies_discussed: "yes" - The paper includes
     a
  full section on adaptation (Section 4.2), discussing
      options
  including "changing planting dates, introducing heat-
      resistant
  cultivars, and implementing irrigation strategies."

- study_timeframe: "future projection" - The paper uses
    historical
  data to calibrate models but primarily presents projections
       for
  2030-2050 as stated: "We project impacts for mid-century
  (2030-2050) under the RCP4.5 and RCP8.5 scenarios."
```

Tip: CoT justifications are invaluable for validating extraction quality, identifying potential misinterpretations, and understanding why models might be struggling with particular papers or information types.

Cost and Rate Management

`prismAId` provides several features to help manage costs and API rate limits.

Token and Request Rate Limiting

Configuration: Rate Limiting Settings

```
[project.llm.1]
provider = "OpenAI"
api_key = ""
model = "gpt-4o-mini"
temperature = 0.01
tpm_limit = 100000  # Tokens per minute limit
rpm_limit = 60      # Requests per minute limit
```

These settings help you:

- **Avoid Provider Rate Limiting**: Stay below provider-enforced limits
- **Control Costs**: Spread API usage over time to manage budget
- **Balance Resource Usage**: Prevent spikes in API consumption

Example Provider Rate Limits

```
Provider  | Model             | Default TPM  | Default RPM
----------|-------------------|--------------|------------
OpenAI    | gpt-3.5-turbo     | 60,000       | 3,500
OpenAI    | gpt-4o            | 10,000       | 500
Anthropic | claude-3-haiku    | N/A          | 5,000
GoogleAI  | gemini-1.5-flash  | 4,000,000    | N/A
Cohere    | command-r         | 100,000      | 1,000
```

Note: Each provider has different rate limit structures. OpenAI limits by TPM (tokens per minute), Anthropic by RPM (requests per minute), and others may have tiered systems. Check the provider's documentation for current limits.

Warning: These limits can change as providers update their services. Always check the most recent documentation from each provider for current rate limits.

Automatic Cost Minimization

`prismAId` can automatically select the most cost-effective model by leaving the model field empty:

Configuration: Cost Minimization

```
[project.llm.1]
provider = "OpenAI"
api_key = ""
model = ""  # Empty value enables automatic model selection
temperature = 0.01
tpm_limit = 0
rpm_limit = 0
```

How automatic model selection works: 1. `prismAId` calculates the token count for each document 2. It determines which provider models can handle the document within token limits 3. Among eligible models, it selects the most cost-efficient option 4. Different documents may be processed by different models based on size

Tip: Automatic model selection is particularly useful for heterogeneous document collections where some papers may be too large for smaller models.

Estimating and Controlling Costs

To estimate the costs of your systematic review:

Command: Dry Run with Cost Estimation

```
# Set log_level to "medium" or "high" in your config
# Run the review with a small sample of papers (3-5)
./prismaid -project your_project.toml

# Review the console output for token usage and cost
    estimates
# Extrapolate to your full dataset
```

Example Cost Summary Output

```
Processing completed.
-------------------------------------
COST SUMMARY
-------------------------------------
Provider: OpenAI
Model: gpt-4o-mini
Files processed: 5
Total input tokens: 23,456
Total output tokens: 2,189
Estimated cost: $0.17

Extrapolated cost for 100 files: ~$3.40
-------------------------------------
```

Reminder: API pricing may change over time. Always check current pricing from your providers before estimating costs for large projects.

Advanced Prompt Engineering

The quality of your systematic review results largely depends on how effectively you structure your prompts. Here are advanced strategies for optimizing extraction through prompt engineering.

Optimizing Prompt Components

Each component of the prompt structure serves a specific purpose and can be optimized:

Configuration: Advanced Prompt Components

```
[prompt]
persona = "You are an expert environmental scientist
    specializing in climate change impacts on agriculture,
    with experience conducting systematic reviews according
    to PRISMA guidelines."
task = "Analyze the scientific paper provided and extract
    specific information about climate change impacts on crop
     yields, methodology used, and adaptation strategies
    discussed."
expected_result = "You should output a JSON object containing
     values for each key specified in the review structure.
    Values must adhere exactly to the prescribed formats and
    vocabulary."
definitions = "'Crop yield' refers to harvestable production
    per unit of land area. 'Statistical significance' means p
     < 0.05 unless otherwise specified in the paper. '
    Adaptation strategies' include any interventions meant to
     reduce negative climate impacts on agriculture."
example = "For instance, if the paper states 'wheat yields
    decreased by 5.2% (p < 0.01) per degree Celsius warming,
    with irrigation mitigating 40% of losses', you would
    extract: {\"crop_type\": \"wheat\", \"climate_factor\":
    \"temperature increase\", \"yield_impact\": \"-5.2% per
    °C\", \"statistical_significance\": \"yes\", \"
    adaptation_strategies_discussed\": \"yes\", \"
    adaptation_effectiveness\": \"40% loss reduction\"}"
failsafe = "If specific information is not clearly stated in
    the document, do not speculate or infer beyond reasonable
     scientific interpretation. Respond with an empty string
    for missing data. If the paper doesn't address the topic
    at all, use 'not addressed' for categorical fields."
```

Key strategies for each component:

- **Persona**: Include relevant expertise and methodological background to establish appropriate context.
- **Task**: Be specific about analytical depth and the nature of extraction required.

- **Expected Result**: Define the exact output format and emphasize adherence to specified value constraints.
- **Definitions**: Provide domain-specific terminology explanations, especially for potentially ambiguous concepts.
- **Example**: Show realistic examples that cover different data patterns and edge cases.
- **Failsafe**: Include clear guidelines for handling uncertainty, ambiguity, and missing information.

Note: Research suggests that more detailed and domain-specific prompts generally produce more accurate extractions, particularly for specialized scientific concepts.

Advanced Review Structure Patterns

Beyond basic information extraction, you can implement more sophisticated patterns in your review structure:

Configuration: Advanced Review Structure Patterns

```
# Hierarchical extraction
[review.1]
key = "methodology_type"
values = ["experimental", "observational", "modeling", "
    review", "mixed", "other", ""]

[review.2]
key = "experimental_design"
values = ["randomized controlled trial", "non-randomized
    controlled trial", "before-after", "other", "not
    applicable", ""]

# Conditional extraction
[review.3]
key = "sample_size_reported"
values = ["yes", "no"]

[review.4]
key = "sample_size_value"
values = [""]

# Relational extraction
[review.5]
key = "crop_types_studied"
values = [""]

[review.6]
key = "primary_crop"
values = [""]

# Confidence assessment
[review.7]
key = "statistical_methods_quality"
values = ["high", "medium", "low", "not applicable", "cannot
    determine", ""]
```

Advanced patterns include:

- **Hierarchical Extraction**: Extract general categories first, then specific details based on those categories.
- **Conditional Extraction**: Use yes/no fields to establish presence, then extract specific values only if present.
- **Relational Extraction**: Capture relationships between different entities (e.g., primary crop among multiple crops studied).
- **Confidence Assessment**: Include quality assessments of methodological elements.

Tip: When using advanced patterns, ensure your prompt includes clear instructions on how these patterns relate to each other, particularly for conditional extractions.

Iterative Prompt Refinement

The most effective prompts are developed through an iterative process:

Iterative Prompt Development Process

```
1. Start with a basic prompt and review structure
2. Test on 3-5 representative papers
3. Analyze results for:
   - Incorrect extractions
   - Missing information
   - Ambiguous responses
   - Inconsistent formatting
4. Identify patterns in errors or weaknesses
5. Refine prompt components and review structure
6. Test again on the same papers plus 2-3 new ones
7. Repeat until results match expectations
8. Document all prompt versions and their performance
```

Warning: Changing prompts mid-review can introduce inconsistency. Once you begin your full review, avoid modifying the prompt unless absolutely necessary. If changes are required, consider re-processing previously analyzed papers.

Workflow Integration and Automation

`prismAId` can be integrated into broader research workflows and automated pipelines.

Programmatic Integration

Each programming language interface allows for integration with existing research workflows:

Python Integration Example

```python
import prismaid
import os
import glob
import pandas as pd
from datetime import datetime

def run_full_workflow(source_dir, output_dir, config_template
    , api_key):
    """Run a complete prismAId workflow from download to
        analysis"""

    # Create timestamped run directory
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    run_dir = os.path.join(output_dir, f"review_run_{
        timestamp}")
    os.makedirs(run_dir, exist_ok=True)

    # Set up subdirectories
    pdf_dir = os.path.join(run_dir, "pdfs")
    txt_dir = os.path.join(run_dir, "texts")
    results_dir = os.path.join(run_dir, "results")
    os.makedirs(pdf_dir, exist_ok=True)
    os.makedirs(txt_dir, exist_ok=True)
    os.makedirs(results_dir, exist_ok=True)

    # 1. Download papers from URL list if available
    if os.path.exists(os.path.join(source_dir, "papers_urls.
        txt")):
        print("Downloading papers from URL list...")
        prismaid.download_url_list(
            os.path.join(source_dir, "papers_urls.txt"),
            output_dir=pdf_dir
        )

    # 2. Convert PDFs to text
    print("Converting PDFs to text...")
    prismaid.convert(pdf_dir, "pdf", output_dir=txt_dir)

    # 3. Update config file with correct paths and API key
    with open(config_template, 'r') as f:
        config = f.read()

    # Update paths and API key
    config = config.replace("INPUT_DIR_PLACEHOLDER", txt_dir)
    config = config.replace("RESULTS_FILE_PLACEHOLDER", os.
        path.join(results_dir, "review_results"))
    config = config.replace("API_KEY_PLACEHOLDER", api_key)

    config_path = os.path.join(run_dir, "review_config.toml")
    with open(config_path, 'w') as f:
        f.write(config)

    # 4. Run the review
    print("Running systematic review...")
    prismaid.review(config)

    # 5. Generate summary report
    print("Generating summary report...")
    try:
        results_file = glob.glob(os.path.join(results_dir, "
            review_results.*"))[0]
        if results_file.endswith('.csv'):
            results = pd.read_csv(results_file)
        else:  # JSON
            results = pd.read_json(results_file)

        # Generate simple summary statistics
        summary = {}
        for column in results.columns:
```

Batch Processing Strategies

For large-scale reviews, batch processing can improve efficiency and manageability:

Bash Script for Batch Processing

```
#!/bin/bash
# batch_review.sh - Process a large corpus in manageable
    batches

if [ "$#" -ne 4 ]; then
    echo "Usage: $0 <input_dir> <output_dir> <batch_size> <
        config_template>"
    exit 1
fi

INPUT_DIR=$1
OUTPUT_DIR=$2
BATCH_SIZE=$3
CONFIG_TEMPLATE=$4

# Create output directory
mkdir -p "$OUTPUT_DIR"

# Count total files
TOTAL_FILES=$(find "$INPUT_DIR" -type f -name "*.txt" | wc -l
    )
echo "Total files to process: $TOTAL_FILES"

# Calculate number of batches
BATCHES=$(( (TOTAL_FILES + BATCH_SIZE - 1) / BATCH_SIZE ))
echo "Will process in $BATCHES batches of approximately
    $BATCH_SIZE files each"

# Process in batches
for BATCH in $(seq 1 $BATCHES); do
    echo "Processing batch $BATCH of $BATCHES"

    # Create batch directory
    BATCH_DIR="$OUTPUT_DIR/batch_$BATCH"
    BATCH_INPUT="$BATCH_DIR/input"
    BATCH_OUTPUT="$BATCH_DIR/output"
    mkdir -p "$BATCH_INPUT"
    mkdir -p "$BATCH_OUTPUT"

    # Get files for this batch
    find "$INPUT_DIR" -type f -name "*.txt" | sort | head -n
        $BATCH_SIZE | \
        xargs -I{} cp {} "$BATCH_INPUT"

    # Remove processed files from consideration for next
        batch
    find "$INPUT_DIR" -type f -name "*.txt" | sort | head -n
        $BATCH_SIZE | \
        xargs -I{} rm {}

    # Create batch-specific config
    BATCH_CONFIG="$BATCH_DIR/config.toml"
    cat "$CONFIG_TEMPLATE" | \
        sed "s|INPUT_DIR_PLACEHOLDER|$BATCH_INPUT|g" | \
        sed "s|OUTPUT_FILE_PLACEHOLDER|$BATCH_OUTPUT/results|
            g" > "$BATCH_CONFIG"

    # Run review for this batch
    echo "Starting review for batch $BATCH"
    ./prismaid -project "$BATCH_CONFIG"

    echo "Completed batch $BATCH"
done

echo "All batches processed. Combining results..."

# Combine results (assuming CSV output)
HEADER=$(head -n 1 "$OUTPUT_DIR/batch_1/output/results.csv")
echo "$HEADER" > "$OUTPUT_DIR/combined_results.csv"
```

Reminder: For very large reviews (hundreds or thousands of papers), consider implementing a database backend to store intermediate results and enable incremental processing.

Integration with Research Workflows

`prismAId` can be integrated with broader research tools and workflows:

- **Reference Management**: Integrate with Zotero, Mendeley, or EndNote for literature organization
- **Statistical Analysis**: Export results to R, SPSS, or specialized meta-analysis software
- **Collaborative Research**: Share configurations and results through version control systems
- **Publication Workflows**: Generate formatted tables and figures for manuscript inclusion

R Script for Meta-Analysis Integration

```r
# Load libraries
library(prismaid)
library(meta)
library(dplyr)
library(readr)

# Run prismAId review
toml_content <- paste(readLines("meta_analysis_config.toml"),
     collapse = "\n")
RunReview(toml_content)

# Load and process results
results <- read_csv("results/meta_analysis_results.csv")

# Extract effect sizes and variance
meta_data <- results %>%
  # Convert extracted text to proper numeric format
  mutate(
    effect_size = as.numeric(gsub("[^0-9.-]", "", effect_size
        )),
    sample_size = as.numeric(gsub("[^0-9]", "", sample_size))
        ,
    p_value = as.numeric(gsub("[^0-9.]", "", p_value))
  ) %>%
  # Filter out rows with missing data
  filter(!is.na(effect_size) & !is.na(sample_size)) %>%
  # Calculate standard error from p-values and effect sizes
     where possible
  mutate(
    standard_error = case_when(
      !is.na(p_value) ~ abs(effect_size) / qnorm(1 - p_value
          /2),
      TRUE ~ NA_real_
    )
  ) %>%
  # Only keep rows with complete data needed for meta-
     analysis
  filter(!is.na(standard_error))

# Perform meta-analysis
if(nrow(meta_data) >= 3) {  # Need at least 3 studies for
    meaningful meta-analysis
  meta_analysis <- metagen(
    TE = meta_data$effect_size,
    seTE = meta_data$standard_error,
    studlab = meta_data$filename,
    sm = "SMD",  # Standardized Mean Difference
    method.tau = "REML",  # Restricted Maximum-Likelihood
        estimator
    hakn = TRUE  # Hartung-Knapp adjustment
  )

  # Print results
  summary(meta_analysis)

  # Create forest plot
  pdf("results/forest_plot.pdf", width=10, height=8)
  forest(meta_analysis,
         sortvar = meta_data$effect_size,
         prediction = TRUE,
         print.tau2 = TRUE,
         print.I2 = TRUE)
  dev.off()

  # Test for publication bias
  pdf("results/funnel_plot.pdf", width=8, height=8)
  funnel(meta_analysis, studlab = TRUE, contour = TRUE)
  dev.off()
```

Conclusion

Advanced features of `prismAId` significantly enhance its capabilities beyond basic systematic reviews. Ensemble reviews provide increased confidence and uncertainty quantification. Debugging features help identify and resolve issues in the extraction process. Cost management strategies optimize resource utilization, while advanced prompt engineering improves the quality and reliability of extractions. Finally, programmatic integration and automation enable seamless incorporation of `prismAId` into comprehensive research workflows.

By mastering these advanced features, researchers can conduct more sophisticated, reliable, and efficient systematic reviews that maintain the highest standards of methodological rigor while leveraging the power of artificial intelligence.

Tip: Start with simpler configurations and gradually incorporate advanced features as you become more familiar with `prismAId`. Each feature adds power but also complexity, so build your expertise incrementally.

Part V

Troubleshooting & FAQs

Troubleshooting Common Issues

This chapter addresses common issues you might encounter when using `prismAId` and provides practical solutions to resolve them quickly.

Installation Problems

Binary Installation Issues

- **Issue: "Permission denied" error when running the executable**

Solution: Fix Permission Issues

```
# For Linux/macOS
chmod +x prismaid

# For Windows (if using PowerShell with execution
    policy restrictions)
Set-ExecutionPolicy -Scope Process -ExecutionPolicy
    Bypass
```

- **Issue: Security warnings when running on macOS**

- **Issue: Missing dependencies message**

Tip: On macOS, you may need to right-click the executable and select "Open" the first time you run it. After confirming once, you can run it normally.

Note: The standalone binaries should include all required dependencies. If you see dependency errors, you may have downloaded an incomplete file. Try downloading again or check for platform-specific versions.

Package Installation Issues

- **Issue: Python package installation fails**

Solution: Python Installation Issues

```
# Ensure pip is up-to-date
python -m pip install --upgrade pip

# Install with verbose output to see errors
pip install prismaid --verbose

# If C compiler errors occur, install build tools
# For Windows:
pip install --upgrade setuptools wheel
# For Linux:
sudo apt-get install build-essential python3-dev
```

- **Issue: R package installation problems**

Solution: R Installation Issues

```
# Make sure you have the necessary system libraries
# For Linux (Ubuntu/Debian):
# sudo apt-get install r-base-dev libcurl4-openssl-dev
     libssl-dev

# Try installing from source
install.packages("prismaid", repos = "https://open-and
    -sustainable.r-universe.dev",
                type = "source")

# Check for error messages
warnings()
```

- **Issue: Julia package fails to build**

Solution: Julia Installation Issues

```
# Update registry
using Pkg
Pkg.Registry.update()

# Try with build flag
Pkg.add("PrismAId"; build=true)

# Check Julia version (requires 1.6+)
versioninfo()
```

Warning: Package installation problems are often environment-specific. If you continue to experience issues, consider using the standalone binary instead, which doesn't require compiling or managing dependencies.

Configuration Errors

TOML File Syntax Problems

- **Issue: "Failed to parse TOML" or similar error**

Common TOML Syntax Errors and Fixes

```
# INCORRECT: Missing quotes around strings with
    special characters
input_directory = C:\path\with\backslashes

# CORRECT: Use quotes and forward slashes or escaped
    backslashes
input_directory = "C:/path/with/forward/slashes"
# OR
input_directory = "C:\\path\\with\\escaped\\
    backslashes"

# INCORRECT: Malformed arrays
values = ["yes" "no"]

# CORRECT: Use commas between array elements
values = ["yes", "no"]

# INCORRECT: Forgetting to close a section
[project.llm.1
provider = "OpenAI"

# CORRECT: Include the closing bracket
[project.llm.1]
provider = "OpenAI"
```

- **Issue: Nested sections not recognized correctly**

Tip: Use a TOML validator like toml-online.com to check your configuration file for syntax errors before running `prismAId`.

Reminder: TOML sections must be properly nested. For example, '[project.llm.1]' is correct, but '[project][llm][1]' would be incorrect.

Path-Related Problems

- **Issue: "Directory not found" or "File not found" errors**

Solution: Verify Paths

```
# Check if the directory exists
# Windows
dir "C:\path\to\check"

# Linux/macOS
ls -la /path/to/check

# Create directories if missing
# Windows
mkdir "C:\path\to\create"

# Linux/macOS
mkdir -p /path/to/create
```

- **Issue: Path encoding problems with non-ASCII characters**

Tip: Always use absolute paths in your configuration file to avoid path resolution issues. Relative paths can cause problems if you run `prismAId` from different directories.

Note: If your paths contain non-ASCII characters (like accented letters or non-Latin scripts), make sure your configuration file is saved with UTF-8 encoding.

API Key Configuration Issues

- **Issue: "API key not found" or authentication errors**

Solution: Check API Key Setup

```
# Check if environment variable is set
# Windows
echo %OPENAI_API_KEY%

# Linux/macOS
echo $OPENAI_API_KEY

# Set environment variable if needed
# Windows
set OPENAI_API_KEY=your-api-key

# Linux/macOS
export OPENAI_API_KEY=your-api-key
```

- **Issue: API key works in environment but not in configuration**

Warning: Check for extra spaces or invisible characters in your API key. Copy-pasting from certain sources can introduce these. Try re-typing the key manually if problems persist.

Common Fixes for Each Tool

Download Tool Issues

- **Issue: Zotero downloads failing**

Solution: Zotero Configuration Fixes

```
# Correct format for Zotero collection path
# INCORRECT
group = Systematic Review/Climate Papers

# CORRECT
group = "Systematic Review/Climate Papers"

# For group collections, use the group name
group = "Group Name/Collection Name"
```

Note: Zotero collection names are case-sensitive. Ensure your collection path exactly matches what appears in your Zotero library.

- **Issue: URL downloads timing out or failing**

Solution: URL Download Issues

```
# Check connectivity to the URL
curl -I https://example.com/paper.pdf

# Try downloading with increased timeout
# (modify the source URL file to include fewer URLs
    per run)
./prismaid -download-URL papers_batch1.txt
```

Reminder: Some publishers block automated downloads. Consider using Zotero integration or manually downloading papers from repositories with strict access controls.

Convert Tool Issues

- **Issue: PDF conversion producing empty or gibberish text**

Solution: Check PDF Quality

```
# View PDF information
pdfinfo problem_paper.pdf

# For scanned PDFs, you need OCR software first
# prismAId can't extract text from image-based PDFs
```

Warning: The Convert tool cannot extract text from scanned image PDFs or PDFs with strong copy protection. Use OCR software like Adobe Acrobat or Tesseract first, or manually transcribe critical sections.

- **Issue: PDF conversion missing text or with garbled formatting**

Tip: PDFs with complex layouts (multiple columns, text boxes, etc.) may not convert perfectly. Consider manually editing the extracted text files to fix critical sections or remove unnecessary content.

Review Tool Issues

- **Issue: Review stops after processing only a few files**

Solution: Check Rate Limits and API Status

```
# Set log level to high in your configuration
# Then run the review to see detailed error messages
./prismaid -project your_project.toml

# Check provider status pages
# OpenAI: https://status.openai.com/
# GoogleAI: https://status.cloud.google.com/
```

Note: If hitting rate limits, configure appropriate 'tpm_limit' and 'rpm_limit' values in your configuration. This will automatically pace the requests to stay within provider limits.

- **Issue: Reviews producing unexpected or empty results**

Solution: Prompt Debugging

```
# Enable Chain-of-Thought justification to see model
    reasoning
[project.configuration]
cot_justification = "yes"

# Enable higher log level to see full prompts and
    responses
log_level = "high"
```

Reminder: Check the model's justifications and logs to understand why it's producing unexpected results. You may need to refine your prompt or review structure.

Performance and Resource Issues

- **Issue: Reviews running very slowly**

Solution: Performance Optimization

```
# Balance rate limits for better performance
[project.llm.1]
provider = "OpenAI"
api_key = ""
model = "gpt-3.5-turbo"  # Use faster model for
    initial testing
temperature = 0.01
tpm_limit = 60000  # Set appropriate limits based on
    your plan
rpm_limit = 3000
```

Tip: For large reviews, consider using batch processing as shown in the Advanced Features chapter, processing subsets of your documents in parallel if your API plan allows.

- **Issue: Out of memory errors with large documents**

Warning: Very large documents may exceed the context window of LLMs or cause memory issues. Consider splitting extremely large papers into smaller sections or removing non-essential content before processing.

Jupyter Notebook Issues

- **Issue: Python kernel crashing in Jupyter when running reviews**

Solution: Jupyter Notebook Integration

```
# Use the workaround for versions <= 0.6.6
import pty
import os
import time
import select

def run_review_with_auto_input(input_str):
    master, slave = pty.openpty()  # Create a pseudo-
        terminal

    pid = os.fork()
    if pid == 0:  # Child process
        os.dup2(slave, 0)  # Redirect stdin
        os.dup2(slave, 1)  # Redirect stdout
        os.dup2(slave, 2)  # Redirect stderr
        os.close(master)
        import prismaid
        prismaid.RunReviewPython(input_str.encode("utf
            -8"))
        os._exit(0)

    else:  # Parent process
        os.close(slave)
        try:
            while True:
                rlist, _, _ = select.select([master],
                    [], [], 5)
                if master in rlist:
                    output = os.read(master, 1024).
                        decode("utf-8", errors="ignore
                        ")
                    if not output:
                        break  # Process finished

                    print(output, end="")

                    if "Do you want to continue?" in
                        output:
                        print("\n[SENDING INPUT: y]")
                        os.write(master, b"y\n")
                        time.sleep(1)
        finally:
            os.close(master)
            os.waitpid(pid, 0)  # Ensure the child
                process is cleaned up

# Load your review (TOML) configuration
with open("config.toml", "r") as file:
    input_str = file.read()

# Run the review function
run_review_with_auto_input(input_str)
```

Note: This issue occurs when the kernel cannot handle interactive prompts. The workaround intercepts and automatically answers confirmation prompts.

Seeking Further Help

If you continue to experience issues after trying the solutions in this chapter:

- **Check Documentation**: Review the online documentation at open-and-sustainable.github.io/prismaid/
- **GitHub Issues**: Search existing issues or create a new one at github.com/open-and-sustainable/prismaid/issues
- **Matrix Support Room**: Join the prismAId Support Room to discuss your issue with the community

Reminder: When seeking help, always include your `prismAId` version, operating system, error messages, and a minimal reproducible example of your issue.

Gathering System Information for Support

```
# Get prismAId version
./prismaid --version

# For Python package
python -c "import prismaid; print(prismaid.__version__)"

# System information
# Windows
systeminfo | findstr /B /C:"OS Name" /C:"OS Version"

# Linux
uname -a
cat /etc/os-release

# macOS
sw_vers
```

Remember that `prismAId` is an open-source project. Detailed bug reports help improve the software for everyone, and community contributions to fix issues are always welcome.

Frequently Asked Questions

This chapter addresses common questions about `prismAId`, its capabilities, integration with other tools, and performance optimization.

General Questions

What types of systematic reviews is prismAId suitable for?

`prismAId` is designed to handle a wide range of systematic review types across various disciplines. It works well for:

- Literature reviews in scientific fields
- Scoping reviews to map available evidence
- Meta-analyses when combined with statistical tools
- Policy reviews and evidence summaries
- Qualitative evidence syntheses

Note: While prismAId can extract information from any text-based literature, it performs best when the information being sought is explicitly stated in the text rather than requiring complex inference or domain-specific interpretation.

Can I use prismAId with non-English literature?

Yes, but with some limitations. `prismAId`'s underlying LLMs support multiple languages with varying degrees of proficiency. However:

- English literature typically yields the most reliable results
- Major European languages (French, German, Spanish) generally work well
- Other languages may have reduced extraction accuracy
- The Convert tool may struggle with non-Latin character sets

Tip: For multilingual reviews, consider testing the extraction quality on a sample of non-English papers before proceeding with the full review. You may need to adjust your prompts to account for language-specific considerations.

Is prismAId compliant with systematic review reporting guidelines?

`prismAId` is designed to support guideline-compliant reviews but doesn't enforce reporting requirements itself:

- It facilitates PRISMA-compliant reviews by enabling structured data extraction
- The methodology is transparent and reproducible, supporting various reporting frameworks
- You must still ensure your final report addresses all required elements from guidelines like PRISMA, MOOSE, or ENTREQ

Example Citation for prismAId in Methods Section

```
In our systematic review, data extraction was performed using
     prismAId
version X.X.X (Boero, 2024), an open-source tool that uses
    large
language models to extract structured information from
    scientific
literature. We used [MODEL NAMES] with a temperature setting
    of
[VALUE] to extract data according to our pre-defined protocol
    .
To ensure extraction quality, we manually validated [XX%] of
extracted data points, finding [XX%] agreement between AI-
    extracted
and manually verified information.

Boero, R. (2024). prismAId - Open Science AI Tools for
    Systematic,
Protocol-Based Literature Reviews. Zenodo.
https://doi.org/10.5281/zenodo.11210796
```

Integration with Other Software

Can I use prismAId with Zotero/Mendeley/EndNote?

Yes, `prismAId` offers integration options with reference managers:

- **Zotero**: Direct integration via the Download tool, which can access papers from your Zotero collections using the API
- **Mendeley/EndNote**: No direct API integration, but you can export papers to a folder and then use `prismAId` to process them

Reminder: When using the Zotero integration, ensure papers have attached PDFs in your Zotero library. prismAId cannot download papers that don't have attachments.

Can I use prismAId with statistical software for meta-analysis?

Yes, `prismAId` works well with statistical tools for meta-analysis:

- Export results as CSV or JSON
- Import into R (with packages like `meta` or `metafor`)
- Import into STATA, SPSS, or specialized meta-analysis software
- Use with Python's `metapsy` or similar packages

R Code for Importing prismAId Results for Meta-Analysis

```r
# Load libraries
library(readr)
library(dplyr)
library(meta)

# Import prismAId results
results <- read_csv("results.csv")

# Prepare data for meta-analysis
# Assuming prismAId extracted effect sizes, sample sizes, and
    p-values
meta_data <- results %>%
  # Clean and convert text to numeric values
  mutate(
    effect_size = as.numeric(gsub("[^0-9.-]", "", effect_size
        )),
    std_error = as.numeric(gsub("[^0-9.]", "", std_error)),
    sample_size = as.numeric(gsub("[^0-9]", "", sample_size))
  ) %>%
  # Remove rows with missing data
  filter(!is.na(effect_size) & !is.na(std_error))

# Run meta-analysis
if(nrow(meta_data) >= 3) {
  ma <- metagen(
    TE = meta_data$effect_size,
    seTE = meta_data$std_error,
    studlab = meta_data$filename,
    sm = "SMD"
  )

  # Print results
  summary(ma)

  # Create forest plot
  forest(ma)
}
```

Can I use prismAId with qualitative analysis software?

Yes, `prismAId` can be used alongside qualitative analysis tools:

- Export results to CSV/JSON and import into NVivo, ATLAS.ti, or MAXQDA

- Use `prismAId` for initial coding and then refine in specialized software

- Combine AI-assisted extraction with manual qualitative analysis

Tip: For qualitative reviews, consider using more open-ended extraction fields in your review structure and then conducting more detailed thematic analysis in specialized software.

Can I integrate prismAId into automated research workflows?

Yes, `prismAId` can be integrated into automated research workflows:

- Use the API from Go, Python, R, or Julia in scripted workflows

- Incorporate into continuous integration pipelines

- Schedule batch processing of papers

- Chain with other research tools using scripts

Python Script for Automated Workflow

```
import prismaid
import pandas as pd
import matplotlib.pyplot as plt
import os
from datetime import datetime

# Create dated output directory
today = datetime.now().strftime("%Y-%m-%d")
output_dir = f"systematic_review_{today}"
os.makedirs(output_dir, exist_ok=True)

# 1. Download new papers from URL list
url_file = "new_papers.txt"
download_dir = f"{output_dir}/pdfs"
os.makedirs(download_dir, exist_ok=True)
prismaid.download_url_list(url_file, download_dir)

# 2. Convert PDFs to text
text_dir = f"{output_dir}/texts"
os.makedirs(text_dir, exist_ok=True)
prismaid.convert(download_dir, "pdf", text_dir)

# 3. Update configuration with new paths
with open("template_config.toml", "r") as f:
    config = f.read()

config = config.replace("INPUT_DIR_PLACEHOLDER", text_dir)
config = config.replace("OUTPUT_FILE_PLACEHOLDER", f"{
    output_dir}/results")

config_path = f"{output_dir}/config.toml"
with open(config_path, "w") as f:
    f.write(config)

# 4. Run the review
prismaid.review(config)

# 5. Generate visualizations from results
results = pd.read_csv(f"{output_dir}/results.csv")

# Example: Create a bar chart of findings
plt.figure(figsize=(10, 6))
results['climate_factor'].value_counts().plot(kind='bar')
plt.title('Distribution of Climate Factors in Literature')
plt.tight_layout()
plt.savefig(f"{output_dir}/climate_factors.png")

# Log completion
with open(f"{output_dir}/log.txt", "w") as f:
    f.write(f"Automated review completed at {datetime.now()}\
        n")
    f.write(f"Processed {len(results)} papers\n")
```

Performance Optimization

How can I optimize prismAId for speed?

To improve processing speed:

- **Use faster models**: Models like `gpt-3.5-turbo`, `claude-3-haiku`, or `gemini-1.5-flash` process text more quickly than their larger counterparts
- **Reduce text length**: Remove non-essential sections (references, acknowledgments) from papers
- **Batch processing**: Process papers in parallel using multiple instances
- **Optimize rate limits**: Set appropriate `tpm_limit` and `rpm_limit` values based on your API plan

Configuration: Speed Optimization

```
[project.llm.1]
provider = "OpenAI"
api_key = ""
model = "gpt-3.5-turbo"  # Faster model
temperature = 0.01
tpm_limit = 60000  # Adjust based on your tier
rpm_limit = 3000   # Adjust based on your tier
```

Warning: Faster models may have lower accuracy for complex extraction tasks. Always validate a sample of results when optimizing for speed.

How can I optimize prismAId for accuracy?

To improve extraction accuracy:

- **Use more capable models**: Models like `gpt-4o`, `claude-3-opus`, or `gemini-1.5-pro` typically provide more accurate extractions
- **Refine prompts**: Create clear, specific prompts with good examples
- **Use ensemble reviews**: Combine multiple models to improve reliability
- **Lower temperature**: Set temperature to 0.01-0.1 for more deterministic responses
- **Break down complex extractions**: Use multiple simpler reviews instead of one complex review

Configuration: Accuracy Optimization

```
[project.llm.1]
provider = "OpenAI"
api_key = ""
model = "gpt-4o"  # More capable model
temperature = 0.01  # Minimal randomness
tpm_limit = 10000
rpm_limit = 500

[prompt]
# More detailed prompt components
persona = "You are an expert scientist with extensive
    experience conducting systematic reviews following PRISMA
     guidelines. You have specific expertise in identifying
    methodological details and extracting precise information
     from scientific papers."
# ... other detailed prompt components
```

How can I optimize prismAId for cost efficiency?

To minimize costs:

- **Use automatic model selection**: Leave the model field empty (‘model = ””) to let `prismAId` choose the most cost-effective model
- **Trim input texts**: Remove unnecessary sections like references, acknowledgments, and front matter
- **Focus reviews**: Extract only the specific information you need rather than general information
- **Test on samples**: Validate your approach on a small sample before processing the entire dataset

Configuration: Cost Optimization

```
[project.llm.1]
provider = "OpenAI"
api_key = ""
model = ""  # Empty for automatic selection
temperature = 0.01
tpm_limit = 0
rpm_limit = 0
```

Tip: For cost-sensitive projects, run a small test first with 5-10 papers and examine the log output (set ‘log_level = ”medium”’) to see token usage and estimated costs per paper.

What are typical processing speeds and costs?

Performance metrics vary based on paper length, model choice, and extraction complexity:

Typical Performance Metrics

```
Model                | Avg. Time Per Paper | Approx. Cost Per
    Paper
---------------------|---------------------|----------------------

gpt-3.5-turbo        | 5-15 seconds        | $0.01-0.05
gpt-4o-mini          | 10-30 seconds       | $0.01-0.08
gpt-4o               | 15-45 seconds       | $0.05-0.25
claude-3-haiku       | 5-20 seconds        | $0.01-0.08
claude-3-opus        | 20-60 seconds       | $0.1-0.5
gemini-1.5-flash     | 3-10 seconds        | $0.0025-0.01
gemini-1.5-pro       | 10-30 seconds       | $0.005-0.03

Note: Based on average 10-15 page scientific papers with
    default settings.
Actual times and costs will vary based on paper length and
    complexity.
```

Methodological Questions

How accurate is information extraction with prismAId?

The accuracy of `prismAId` extractions depends on several factors:

- **Prompt quality**: Well-crafted prompts improve accuracy
- **Information type**: Explicit data is extracted more accurately than implicit information
- **Model capability**: More advanced models generally provide higher accuracy
- **Document quality**: Clean, well-structured texts yield better results

Note: In validation studies, accuracy typically ranges from 85-95% for straightforward information (e.g., methodology type, sample size) to 70-85% for more complex or subjective information (e.g., study limitations, conceptual frameworks).

Reminder: Always validate a subset of extractions manually to assess accuracy for your specific review task. Consider using ensemble reviews for high-stakes systematic reviews.

How should I validate prismAId results?

Best practices for validation include:

- Manually check a random sample (10-20%) of papers
- Use dual extraction (AI + human) for critical information
- Run ensemble reviews with multiple models and examine disagreements
- Calculate inter-rater reliability between AI and human extractions

- Document validation process and results in your methods section

Python Code for Calculating Cohen's Kappa with Human Validation

```
import pandas as pd
from sklearn.metrics import cohen_kappa_score

# Load AI-extracted data
ai_results = pd.read_csv("prismaid_results.csv")

# Load human validation data (same papers, same fields)
human_results = pd.read_csv("human_validation.csv")

# Ensure the same papers are being compared
common_papers = set(ai_results['filename']).intersection(set(
    human_results['filename']))
ai_subset = ai_results[ai_results['filename'].isin(
    common_papers)]
human_subset = human_results[human_results['filename'].isin(
    common_papers)]

# Sort both dataframes by filename to align rows
ai_subset = ai_subset.sort_values('filename').reset_index(
    drop=True)
human_subset = human_subset.sort_values('filename').
    reset_index(drop=True)

# Calculate agreement for each extracted field
agreement_results = {}
for field in ['methodology_type', 'sample_size', '
    climate_factor', 'crop_type']:
    if field in ai_subset.columns and field in human_subset.
        columns:
        try:
            # Calculate simple agreement percentage
            agreement = sum(ai_subset[field] == human_subset[
                field]) / len(ai_subset)

            # Calculate Cohen's Kappa for categorical
                variables
            kappa = cohen_kappa_score(ai_subset[field],
                human_subset[field])

            agreement_results[field] = {
                'agreement_percentage': agreement * 100,
                'cohens_kappa': kappa
            }
        except:
            print(f"Error calculating agreement for {field}")

# Print results
for field, metrics in agreement_results.items():
    print(f"{field}:")
    print(f"  Agreement: {metrics['agreement_percentage']:.1f
        }%")
    print(f"  Cohen's Kappa: {metrics['cohens_kappa']:.3f}")

    # Interpret Kappa value
    kappa = metrics['cohens_kappa']
    interpretation = ""
    if kappa < 0.20: interpretation = "Poor agreement"
    elif kappa < 0.40: interpretation = "Fair agreement"
    elif kappa < 0.60: interpretation = "Moderate agreement"
    elif kappa < 0.80: interpretation = "Substantial
        agreement"
    else: interpretation = "Almost perfect agreement"

    print(f"  Interpretation: {interpretation}")
    print()
```

Can prismAId completely replace manual review?

No, `prismAId` should be viewed as an assistive tool rather than a complete replacement for human judgment:

- It excels at reducing the time-intensive aspects of systematic reviews
- Human oversight remains essential for quality control and interpretation
- Domain expertise is still needed to contextualize findings and draw conclusions
- Complex or nuanced information may require human validation

Warning: Complete automation of systematic reviews without human supervision is not recommended, particularly for reviews that inform clinical practice, policy decisions, or other high-stakes applications.

Advanced Usage Questions

Can I use prismAId with custom or proprietary LLMs?

Currently, `prismAId` supports five major commercial LLM providers (OpenAI, GoogleAI, Anthropic, Cohere, DeepSeek). Support for custom or self-hosted models is not built into the main distribution.

Tip: As an open-source project, prismAId can be extended to support additional LLM providers. If you have programming experience, you could fork the repository and add support for your preferred models by extending the interface code.

Can I use prismAId for sensitive or confidential data?

You should exercise caution when using `prismAId` with sensitive data:

- All text sent to LLM providers may be processed on their servers
- Check each provider's privacy policy and data usage terms
- Consider data residency and compliance requirements (GDPR, HIPAA, etc.)
- For highly sensitive content, consider redacting identifying information before processing

Warning: When using commercial LLM APIs, the text of your documents is sent to third-party servers. While providers typically have privacy safeguards, you should never process confidential or personally identifiable information without ensuring compliance with relevant regulations and obtaining appropriate permissions.

How can I contribute to prismAId development?

As an open-source project, `prismAId` welcomes contributions:

- Report bugs or request features through GitHub issues
- Contribute code improvements via pull requests
- Help improve documentation or examples

- Share validation studies or use cases

Contributing to prismAId

```
# Fork the repository on GitHub
# Clone your fork
git clone https://github.com/your-username/prismaid.git

# Create a feature branch
git checkout -b feature/your-feature-name

# Make your changes
# Test your changes

# Commit with a descriptive message
git commit -m "Add feature: description of your changes"

# Push to your fork
git push origin feature/your-feature-name

# Create a pull request on GitHub
```

Reminder: Before contributing code, familiarize yourself with the project's contribution guidelines and code of conduct in the repository.

Will my API usage for prismAId affect my other projects using the same API keys?

Yes, any `prismAId` usage will count toward your overall API usage limits with each provider:

- API calls made by `prismAId` will appear in your provider dashboard
- Usage counts toward any quotas or rate limits on your account
- Billing is based on total token usage across all applications using your key

Tip: For organizations with multiple projects using the same LLM providers, consider creating separate API keys for different projects to track usage more effectively and implement budget controls.

Future Development Questions

What new features are planned for prismAId?

While specific roadmap details may change, planned enhancements include:

- Support for additional LLM providers and models
- Enhanced visualization and analysis tools
- Improved OCR integration for image-based PDFs
- More sophisticated ensemble methods

- Interactive review refinement capabilities

Note: As an open-source project, prismAId's development direction is influenced by community needs and contributions. You can suggest features or vote on existing proposals through the GitHub repository.

How can I stay updated on prismAId developments?

To stay informed about new releases and features:

- Follow the GitHub repository
- Join the Matrix Announcements Room
- Subscribe to release notifications
- Check the documentation website periodically

Following prismAId Updates

```
GitHub Repository: https://github.com/open-and-sustainable/
    prismaid
Matrix Announcements: https://matrix.to/#/#prismAId-
    announcements:matrix.org
Documentation: https://open-and-sustainable.github.io/
    prismaid/
```

Reminder: Major version updates may include changes to configuration structure or API methods. Always review the changelog before upgrading.

Conclusion

This FAQ covers common questions about `prismAId`, but the field of AI-assisted systematic reviews is rapidly evolving. For the most current information, consult the online documentation, join the community support channels, or review recent publications about `prismAId` methodology.

If your question isn't addressed here, consider posting in the Matrix Support Room or opening a discussion on the GitHub repository to engage with the community of users and developers.