Peerly: An agentic co-reviewer for technical manuscripts

Arnab Bhattacharya

Task 1: Defining your Problem and Audience

Problem: Researchers lack access to an interactive writing assistant that provides immediate, high-quality *peer-level* feedback during technical manuscript writing.

Researchers and academics face significant challenges when writing technical manuscripts, often submitting work with clarity issues, methodological gaps, citation inconsistencies, or insufficient domain rigor that could have been identified and addressed beforehand. Traditional peer review is time-intensive, costly, and provides feedback only after substantial time investment, while self-review is limited by authors' blind spots and lack of access to comprehensive best practices from their target venues.

Task 2: Propose a Solution

Peerly addresses this problem by providing immediate, structured feedback on research manuscripts before submission. Rather than rewriting papers using AI, Peerly empowers writers to improve their work through high-quality suggestions that understand semantic meaning, domain coherence, and methodological rigor. The system helps researchers identify and fix problems iteratively and early during the writing process.

While at an early stage of its production, Peerly seeks to provide a clean, distraction-free interface similar to Grammarly's inline suggestion model: users see their document on the left with highlighted sections indicating potential issues, while the right panel displays categorized suggestions (clarity, rigor, methodology, citations etc.) with severity indicators (info, warning, error). Each suggestion includes a specific line reference, explanation of the issue, concrete improvement recommendations, and relevant guidelines or best practices from the knowledge base.

2.1 Multi-agent Architecture

Currently, Peerly employs a *multi-agent architecture with specialized reasoning and function calling*. It comprises of the following agents:

1. **Clarity Agent**: Uses agentic reasoning to analyze each document section for communication issues, identifying unclear jargon, confusing explanations, and structural problems by comparing content against clarity guidelines retrieved from a knowledge base (RAG). The agent employs a reflection loop to self-validate suggestions, filtering out false positives before presenting feedback.

2. **Rigor Agent**: Applies agentic reasoning to evaluate methodological soundness, experimental design, semantic coherence in math or statistics heavy domain concepts, and statistical validity by retrieving rigor-specific guidelines (RAG) and, when needed, invoking Tavily search (Function calling) to validate claims against external sources. This agent reasons about whether methods are appropriate for research questions, whether limitations are adequately discussed, and whether results support conclusions drawn.

3. **Orchestrator Agent:** Serves as a meta-reasoning layer that validates suggestions from both specialized agents, resolves conflicts when clarity and rigor suggestions overlap or contradict, prioritizes feedback based on severity and impact, and ensures the final suggestion set is coherent and actionable. This prevents overwhelming users with redundant or contradictory advice.

Why use agents? The agentic reasoning approach is critical because manuscript review requires contextual judgment rather than simple pattern matching. Agents must understand domain semantics, evaluate tradeoffs (e.g., technical precision vs. accessibility), and reason about whether retrieved guidelines actually apply to the specific paper context. The reflection and orchestration loops ensure high-precision feedback by having agents critique their own suggestions and cross-validate findings, mimicking the multi-reviewer deliberation process in human peer review.

2.2 Technical Stack

- **LLM Models:**
    - *GPT-4o* for rigor analysis and orchestration agents. Chosen for superior reasoning capabilities needed to evaluate complex methodological soundness and make final validation decisions across competing agent suggestions.
    - *GPT-4o-mini* for clarity agent. Selected for cost-effectiveness and speed while maintaining sufficient quality for pattern-matching tasks like identifying jargon and unclear explanations.

- **Embedding Model:** OpenAI text-embedding-3-large provides high-dimensional (3072) embeddings with strong semantic understanding crucial for matching paper sections to relevant best practices and guidelines in the knowledge base.

- **Orchestration:** LangGraph to enable building complex, stateful multi-agent workflows with conditional routing, parallel execution, and human-in-the-loop capabilities, essential for coordinating specialized clarity and rigor agents with reflection loops.

- **Vector Database:** Qdrant (via Dockerized Containers)

- **Retrieval Enhancement:**
    - Cohere Rerank v3.5 for contextual compression
    - Tavily Search: Provides external web search for rigor agent to validate methodological claims and find recent best practices not present in the static knowledge base.

- **Chunking Strategy:**
    - RecursiveTextCharacterSplitter
    - Semantic Chunking (default)

- **Backend:** FastAPI
- **Frontend**: React + TypeScript (ongoing)
- **Monitoring:** TBD - integrate with Langsmith for tracing
- **Serving and Inference**: OpenAI API offering reliable, scalable inference with competitive pricing and access to latest models (ongoing)

Task 3: Dealing with Data

3.1 Internal Knowledge Base (PDF Guidelines):

- Clarity Guidelines (used by clarity agent):
    - *"How to Write a Clear Math Paper"* (stored in `app/resources/clarity_docs/`) - provides academic writing best practices for mathematical clarity, explanation techniques, and audience-appropriate communication.
- Rigor Guidelines (used by rigor agent)**:**
    - *"Mathematical Writing"* by Knuth (stored in `app/resources/rigor_docs/`) - contains methodological standards, proof techniques, experimental design principles, and statistical rigor requirements.
- General Writing Guidelines (not used currently but will be in future):
    - *"Primer on Math Writing"* (stored in `app/resources/`) - comprehensive guide covering both clarity and rigor aspects of technical writing.

- Usage: These PDFs are processed using PyMuPDF, chunked semantically, embedded via OpenAI embeddings, and stored in Qdrant (deployed as a dockerized container) for retrieval during agent analysis.

3.2 Tavily Search API

- External web search API for real-time retrieval of authoritative sources.
- Usage: Rigor agent invokes Tavily when evaluating claims about domain-specific standards (e.g., "FDA Phase III trial requirements", "when to use Bonferroni correction") that may not exist in static PDF guidelines or require up-to-date information.
- Rate-limited to 2 calls per section to control costs and latency.

3.3 Default Chunking Strategy

We use semantic chunking (with a percentile breakpoint threshold and enforced min and max size constraints) as the default strategy because:

- Academic writing has an inherently semantic structure —ideas flow in logical units (definitions, theorems, examples, arguments) that don't align with arbitrary character counts.
- When agents retrieve a chunk that starts mid-argument or mid-example, the retrieved context lacks necessary setup, making guidelines harder to apply correctly.
- Overlapping chunks create duplicate near-identical retrievals, wasting context window space and potentially confusing the LLM with repetitive information.
- Semantic chunking preserves *logical completeness* — each chunk represents a coherent idea unit. When the rigor agent retrieves guidance about "sample size requirements," it gets the full explanation with context, examples, and caveats in one chunk, rather than fragmented pieces.

Task 4: Building an End-to-End Agentic RAG Prototype

Peerly implements a LangGraph-based multi-agent workflow that orchestrates three specialized agents with RAG capabilities:

Pipeline:

- *Document Parsing* → User manuscript is parsed into sections by `SectionAnalyzer` (heading-based segmentation)
- *Parallel Agents* → Clarity and Rigor agents process each section independently with RAG-enhanced reasoning (Rigor can also access Tavily for enhanced context)
- *Reflection Loops* → Each agent self-validates suggestions to filter false positives

- *Orchestration Agent* → Orchestrator agent validates, deduplicates, and prioritizes final suggestions
- *Response Generation* → Structured JSON response with categorized, line-specific feedback

The prototype runs entirely locally with Qdrant (Docker), FastAPI backend (local/Docker), and OpenAI/Cohere APIs for inference/reranking, enabling rapid iteration and testing without cloud deployment complexity.

Task 5: Creating a Golden Test Data Set

5.1 Dataset Generation

I used the *EvolInstruct* paradigm to generate high-quality golden evaluation dataset for the Clarity and Rigor agents through a 3-step pipeline:

- **Step 1: Generate Seed**
    - Extract 10 guideline chunks from Qdrant (clarity and rigor PDFs)
    - Use GPT-4o (temp=0.7) to generate 2 seed examples per chunk
    - Each seed contains: flawed paper section (`reference_question`), relevant guideline (`reference_context`), expected suggestion (`reference_answer`
    - Output: ~20 seed examples per agent

- **Step 2: Evolve**
    - Select top 10 seeds from Step 1
    - Apply 4 evolution operators (customized prompts) per seed to create diverse variants:
        - Increased realism: make examples sound more like real papers
        - Add distractor: embed flaws in correct content to test precision
        - Increase subtlety: Make violations borderline
        - Combine issues: Add multiple co-occurring issues
    - Use GPT-4o(temp=0.8) for creative variation
    - Output: ~50 candidates per agent type (10 original + 40 evolved)

- **Step 3: Filter to Final Golden Dataset**
    - Use GPT-4o as LLM-judge to score each candidate on 6 criteria (1-5 scale):
        - Realism (20%): Sounds like real academic paper?
        - Clarity of Issue (15%): Issue clearly identifiable?
        - Pedagogical Value (20%): Teaches something valuable?
        - Actionability (20%): Suggestion specific and actionable?

        - Guideline Alignment (15%): Guideline matches the issue?
        - Overall Quality (10%): Holistic assessment
    - Select top 10 by weighted score ensuring diversity across issue types and domains
    - Output: **10 golden evaluation examples** per agent type (Clarity and Rigor)
        - Clarity agent's eval dataset: *golden_clarity_10.csv*
        - RIgor agent's eval dataset: *golden_rigor_10.csv*

5.2 Evaluation and Performance Assessment from Baseline Results

We test each agent's baseline retrieval strategy (naive retrieval for k=8, 10, where k is number of retrieved docs) using their respective golden evaluation dataset.

Note:

- For evaluations, *we will consider RAG evaluations for the clarity and rigor agents only* - we will omit the orchestrator's assessment for now as it is more of a meta reasoning agent than an agentic RAG system
- I am omitting the faithfulness metric here as that will be a more relevant metric for the overall end-to-end system (final output from the orchestrator).

Retriever	**Answer relevancy**	**Context Precision**	**Context Recall**
Naive (k=8)	0.0753	0.1250	1.0000
Naive (k=10)	0.1885	0.1000	1.0000

Table 1: Baseline performance for clarity agent using the golden_clarity_10.csv eval data

Retriever	**Answer relevancy**	**Context Precision**	**Context Recall**
Naive (k=8)	0.7436	0.2500	1.0000
Naive (k=10)	0.7436	0.2300	1.0000

Table 2: Baseline performance for rigor agent using the golden_rigor_10.csv eval data

5.3 Conclusions from Baseline Assessment

Strengths:

- High recall (1.0) - The system finds relevant information when it exists
- Rigor agent can find relevant answers
- Results are stable across k values

Weaknesses:
- Low precision is an issue - 75-90% of retrieved context is noise
- Clarity agent's answer quality is not great - cannot handle low-precision retrieval

Task 6: The Benefits of Advanced Retrieval

6.1 Advanced Retrieval Strategies Tested and Motivation

- Cohere Rerank (Two-stage retrieval with semantic reranking)
    - This should potentially improve precision in both agents while maintaining contextual relevance and semantic understanding
- BM25 Retrieval (Keyword-based lexical search combined with semantic search)
    - Keyword matching might potentially improve recall for technical terminology while maintaining semantic understanding. The success for this may depend on the diversity of RAG's vectorDB data that covers topics from multiple domains/technical fields.
- Semantic Chunking
    - Creates more contextually coherent chunks based on semantic boundaries rather than fixed sizes - which is important to determine the relevant "guidelines" from the pdf data used by each agent - and should provide better-quality context units retrievers to evaluate and select.
    - This is not a retrieval strategy but rather a chunking strategy - I will use it as default strategy for splitting documents from here on

6.2 Evaluation Testing using the Advanced Retrieval Strategies

Process steps:
- Semantic Chunking used for each evaluation experiment (more extensive comparisons with recursive splitting will be done at a future time)
- For both the clarity and rigor agents:
    - Cohere rerank (v3.5) and BM25 was run for k=8 and k=10
    - Agent's RAG performance evaluated on their respective golden dataset
    - Record the RAGAS quality metrics (precision, recall, relevance) separately
    - Metric plots and tables generated separately

The evaluation results will be discussed in the next section.

Task 7: Assessing Performance

Note : All evaluation results below are with semantic chunking.

7.1 Performance for advanced v/s naive retrievers for Clarity agent

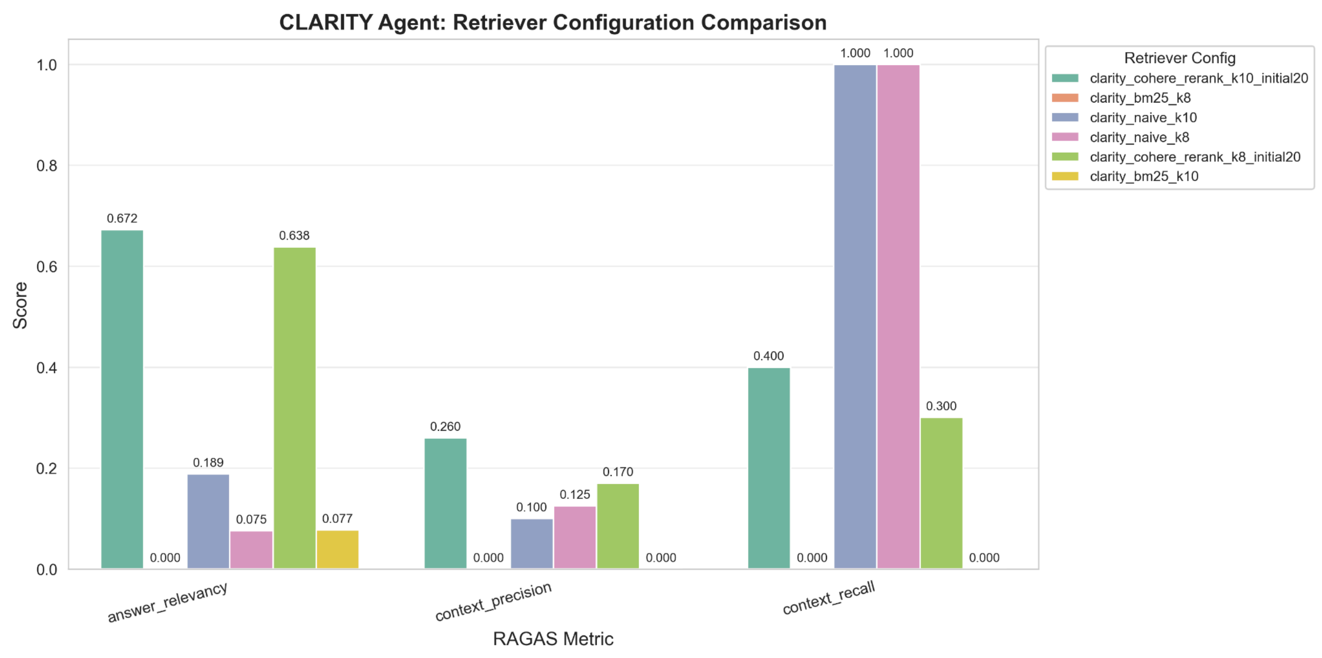

Figure: Comparison of evaluation metrics for clarity agent

Key Insights:

- Rerank is the best in terms of answer relevance, which means they consistently answered what was asked
    - Rerank with k=10 shows >300% improvement over Naive k=10
- Again Rerank outperforms in context precision
    - Increasing k from 8→10 improves rerank performance by > 52%
- Naive has the best recall but provides unusable answers for both values of k (~0.07)
- BM25 remains ineffective at both k values

Suggestion: Rerank is a solid choice for the clarity agent!

7.2 Performance for advanced v/s naive retrievers for Rigor agent

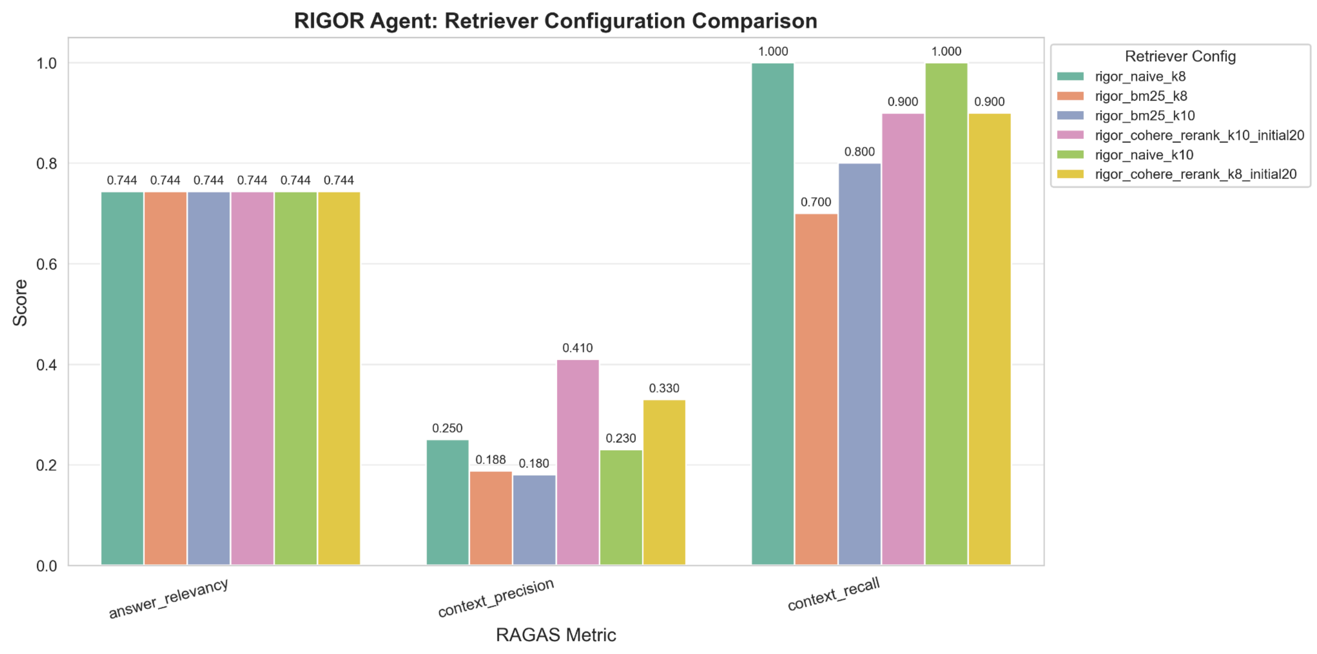

Figure: Comparison of evaluation metrics for rigor agent

Key Insights:
- Rerank outperforms others in terms of context precision
  - For k=10, precision improves by > 78% for Rerank v/s Naive
  - Increasing k from 8→10 improves rerank performance by > 52%
- Naive has the best recall but rerank is not far behind
- BM25 performance is better than that in clarity agent, but still falls behind the Rerank and Naive

Suggestion: Rerank is a solid choice for the rigor agent too!

It is important to note that
- Clarity requires *conceptual understanding, semantic similarity, contextual relevance*
- Rigor requires *factual accuracy, comprehensive coverage, precise relevance*

Cohere Rerank provides both semantic understanding and precision through two-stage retrieval.

7.3 Next Steps and Future Directions

- Develop agents for other responsibilities, such as, ethical standards, best practices, citation relevance.
    - They will all be managed by the orchestrator agent
- Create more granular golden datasets for different types of technical paper sections and associated issues (e.g. data for “Introduction”, “Methodology”, “Model” etc.)
- More efficient context management strategies
    - e.g. caching sub-agents responses for different sections will aid in faster retrieval as sections are incrementally updated by the user
- Calculate cost and latency metrics for end-to-end application (i.e., with Orchestrator in the loop)
- Multi-agent end-to-end evaluations - possibly use llm-as-judge metrics as well as quality metrics focused on the generation
- UI/UX improvements
- Testing with different reasoning models
- Using cloud-based VectorDB for scalability
- Improving test-time compute - the goal is to improve latency so suggestions are created quickly without degrading performance (e.g. hallucinations)