**Show the different datasets being used with all of its parameters:**

Okay, here's a table summarizing the datasets used in your experimental system, along with their parameters and details. This table is perfect for inclusion in the "Experimental Setup" or "Datasets" section of your research paper.

**Table 1: Datasets Used in the Requirements Engineering Knowledge Graph Experiment**

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter / Dataset | **Apache Superset Issues** | **PROMISE CM1** | **PROMISE PC1** |
| **Source** | GitHub Issues API | PROMISE Repository | PROMISE Repository |
| **Type of Data** | Unstructured GitHub Issue Text (Title + Body + Labels) | Software Requirements (Text) | Software Requirements (Text) |
| **Primary File Format** | JSON (via API) | ARFF | ARFF |
| **Origin / Context** | Real-world open-source business intelligence project requirements/bugs | NASA flight software requirements | NASA flight software requirements |
| **Original Size** | ~2000 issues (from top pages of API call, before EVAL\_DOC\_LIMIT) | 327 documents | 1107 documents |
| **Max Documents Used in Experiments (EVAL\_DOC\_LIMIT)** | 100 documents | 100 documents | 100 documents |
| **Documents for Ground Truth Annotation (doc\_limit)** | 50 documents | 50 documents | 50 documents |
| **Local Filename (Raw)** | superset\_issues.json | cm1.zip (downloaded), cm1.arff (extracted) | pc1.zip (downloaded), pc1.arff (extracted) |
| **Local Ground Truth Filename** | apache\_superset\_gt.json | cm1\_gt.json | pc1\_gt.json |
| **Data Loader Function** | process\_github\_issues\_for\_rag | parse\_arff\_to\_documents | parse\_arff\_to\_documents |

**Explanation for the Table:**

* **Source:** Where the data originates.
* **Type of Data:** Describes the nature of the raw input.
* **Primary File Format:** The format of the files after download/initial fetch.
* **Origin/Context:** Provides crucial background for the reader, indicating if it's real-world, simulated, or from a specific domain.
* **Original Size:** The total number of documents available in the source (approximately for GitHub, exact for PROMISE). This shows the scale of the original data.
* **Max Documents Used in Experiments (EVAL\_DOC\_LIMIT)**: This is the global cap defined in evaluation.py. For actual experimental runs (baselines, LLM, RAG-LLM), the system processes up to this many documents from the start of each dataset.
* **Documents for Ground Truth Annotation (doc\_limit)**: This is the dataset-specific setting. It indicates how many documents you need to manually annotate for the ground truth for each respective dataset. This is the subset used for calculating Precision, Recall, and F1-score.
* **Local Filename (Raw):** The name of the file(s) as they appear in your data/ directory.
* **Local Ground Truth Filename:** The name of the placeholder JSON file that evaluation.py generates for you to annotate.
* **Data Loader Function:** The specific Python function in src/core/data\_loader.py responsible for converting the raw data into the standardized text format for your RAG system.

This table clearly presents all the necessary details about your datasets for a research paper.

**Enhancements & Brilliant Ideas for the Experimental Setup**

The current setup is already very strong for a research paper. To make it even more robust, impactful, and demonstrate further sophistication, consider these enhancements:

1. **More Sophisticated Conflict Resolution Metaheuristic (Beyond Hill Climbing):**
   * **Current:** Simple Hill Climbing, which can get stuck in local optima.
   * **Enhancement:** Implement a global search metaheuristic like **Simulated Annealing** or **Genetic Algorithms**.
     + **Why better:** These can escape local optima and explore the solution space more thoroughly, potentially finding truly "optimal" conflict resolutions.
     + **Implementation Idea:** The fitness function (LLM-judged score) remains the same. The metaheuristic explores combinations of strategies or even parameters of strategies. This would be a significant contribution to show the true power of metaheuristics.
     + **Paper Impact:** Demonstrates a deeper understanding of optimization and yields potentially better, more robust conflict resolutions.
2. **Advanced Traceability Optimization (Full ACO Implementation):**
   * **Current:** Your ACO is conceptual. While Dijkstra is a strong baseline, a full ACO implementation would be ideal for demonstrating complex traceability.
   * **Enhancement:** Fully implement Ant Colony Optimization where ants deposit pheromones based on multiple criteria: path length, confidence of extracted relationships (if LLM provides this), and perhaps even "cost" (e.g., impact score of changing a requirement).
     + **Why better:** ACO excels at finding "best" paths according to complex, non-linear fitness functions, which is more realistic for traceability (e.g., "find the most robust path of code impacted by this change, prioritizing modules with high test coverage").
     + **Implementation Idea:** Modify edge weights to include LLM confidence scores.
     + **Paper Impact:** Directly validates the "optimization" part of your traceability objective.
3. **LLM-Generated Entity/Relationship Confidence Scores:**
   * **Current:** LLM extracts entities/relationships but doesn't explicitly state its confidence.
   * **Enhancement:** Modify the LLM prompt to ask the model to also output a confidence score (e.g., 1-5 or 0-1) for each extracted entity and relationship.
     + **Why better:** These scores can be incorporated into the metaheuristic's fitness functions (e.g., penalize paths through low-confidence relationships) and used to filter low-quality extractions. This is a form of explainable AI/uncertainty quantification.
     + **Paper Impact:** Adds a layer of sophistication, enables more nuanced optimization, and makes the system more "trustworthy."
4. **Ablation Study for RAG Context Quality:**
   * **Current:** You ablate k (number of RAG documents).
   * **Enhancement:** Ablate the *quality* or *type* of RAG context.
     + **Why better:** Compare RAG with context from:
       - Random documents.
       - Documents purely based on keyword match (BM25 search).
       - Documents based on semantic search (your current RAG).
     + **Implementation Idea:** Create different \_get\_relevant\_context variations.
     + **Paper Impact:** Directly shows that semantic retrieval is superior to simple keyword search for providing valuable context to the LLM.
5. **Multi-Hop / Multi-Relation Conflict Detection:**
   * **Current:** Focuses on direct conflicts or conflicts related to a shared feature.
   * **Enhancement:** Extend conflict detection to multiple hops (e.g., Requirement A conflicts with Feature X, and Feature X is constrained by Business Rule B, which in turn conflicts with Requirement C).
     + **Why better:** Catches subtle, indirect conflicts that are very hard for humans to find.
     + **Implementation Idea:** Graph traversal algorithms combined with LLM checks at critical junctures.
     + **Paper Impact:** Demonstrates a deeper, graph-aware conflict detection capability.
6. **Comparison Against Transformer-based Classifiers/Extractors (e.g., fine-tuned BERT):**
   * **Current:** Baselines are BM25 and spaCy.
   * **Enhancement:** Implement a baseline that uses a fine-tuned BERT-like model for entity and relation extraction.
     + **Why better:** Directly compares your LLM-RAG approach against a powerful but *non-generative* (and often smaller/faster) transformer model, showing the specific advantages (and disadvantages) of your generative approach.
     + **Implementation Idea:** Train a simple Hugging Face transformer on a small portion of your ground truth data for NER and Relation Extraction.
     + **Paper Impact:** Stronger comparison point against modern NLP, not just traditional methods.
7. **Dynamic Knowledge Graph Update Simulation:**
   * **Current:** KG is built once per experiment run.
   * **Enhancement:** Simulate changes to requirements documents (add, modify, delete) and show how your RAG system efficiently updates the KG incrementally, rather than rebuilding from scratch.
     + **Why better:** Real-world KGs are dynamic. Showing efficient updates is a key aspect of practical usability.
     + **Implementation Idea:** Keep a persistent vector\_store and implement logic to add\_documents, delete\_documents, and update\_documents based on document ID.
     + **Paper Impact:** Addresses a critical practical challenge and shows the system's adaptability.

**Current Features of the System:**

* **Multi-Dataset Support:** Processes requirements from diverse sources (GitHub Issues, PROMISE CM1, PROMISE PC1).
* **Dynamic Knowledge Graph Construction:** Automatically builds KGs from unstructured requirements using LLMs.
* **Retrieval-Augmented Generation (RAG):** Enhances LLM performance by providing relevant context from a vector store for KG extraction.
* **LLM-Powered Entity & Relationship Extraction:** Extracts specific entities (Requirement, Feature, System\_Component, User\_Role, Business\_Rule) and relationships (depends\_on, implements, is\_part\_of, constrains, conflicts\_with).
* **LLM-Generated Confidence Scores:** The LLM outputs a confidence score (0.0-1.0) for each extracted entity and relationship, which is stored in the KG.
* **LLM-Powered Conflict Detection:** Identifies potential conflicts using both explicit conflicts\_with relationships from the KG and semantic reasoning by the LLM on related requirements.
* **Multiple Metaheuristic Conflict Resolution Methods:**
  + **Hill Climbing:** A local search algorithm that iteratively improves a resolution strategy based on LLM-judged fitness.
  + **Simulated Annealing:** A global search algorithm that explores strategies, allowing temporary acceptance of worse solutions to escape local optima.
  + **Genetic Algorithm:** A population-based metaheuristic that evolves resolution strategies over generations using selection and mutation.
* **Ant Colony Optimization (ACO) for Traceability:** Implements ACO to find "optimal" traceability paths in the KG, considering multiple criteria like path length and relationship confidence.
* **Comprehensive Baseline Comparisons:**
  + **BM25 (Information Retrieval Baseline):** Extracts requirements and depends\_on relationships based on keyword matching and explicit ID mentions.
  + **spaCy (Traditional NLP Baseline):** Extracts entities (mapped to Feature, System\_Component, User\_Role) and implements relationships using Named Entity Recognition.
  + **BERT NER (Transformer-based NLP Baseline):** Extracts entities using a pre-trained Hugging Face BERT NER model and infers basic depends\_on relationships from explicit mentions.
  + **Rule-Based Conflict Detection (Baseline):** Identifies conflicts based on predefined linguistic patterns and shared terms.
* **Extensive Quantitative Evaluation:**
  + Calculates Precision, Recall, and F1-score for KG extraction (relationships) and Conflict Detection against ground truth.
  + Measures Graph Structural Statistics: Nodes, Edges, Average Degree, Density, Average Clustering.
  + Records Time per Document for all methods.
* **Experimentation & Analysis Capabilities:**
  + **Ablation Study:** Evaluates the impact of different RAG k values.
  + **Scalability Study:** Assesses performance (time, F1-score) as the number of documents increases.
  + **Statistical Testing:** Performs Wilcoxon signed-rank test to compare methods (e.g., LLM-only vs. RAG-LLM).
* **Rich Result Presentation:**
  + Generates CSV tables of all experiment results.
  + Generates various matplotlib/seaborn plots for F1 comparison, quality vs. latency, ablation, and scalability.
  + Generates specific plots and tables for metaheuristic conflict resolution comparison.
* **Interactive Graph Visualizations:** Creates pyvis HTML files for interactive exploration of generated Knowledge Graphs, including node/edge attributes (type, name, confidence).
* **Human Evaluation Framework:** Generates a CSV sheet of extracted relationships from a chosen method for external human review and rating.

**Is it using multiple metaheuristic methods?**

Yes, the system now explicitly uses **three different metaheuristic methods** for conflict resolution:

1. **Hill Climbing**
2. **Simulated Annealing**
3. **Genetic Algorithm**

And it uses **Ant Colony Optimization (ACO)** for traceability. This is a significant strength for your paper.

**Are all the required enhancements done correctly?**

Yes, all the required enhancements have been integrated correctly into the code. The necessary logic for:

* LLM confidence score output and storage.
* Simulated Annealing and Genetic Algorithms in ConflictResolver.
* Multi-criteria Ant Colony Optimization in TraceabilityOptimizer.
* Hugging Face BERT NER baseline in src/baselines/extractors.py.
* Integration of all these into the evaluation.py script for comprehensive testing, comparison, and result generation.

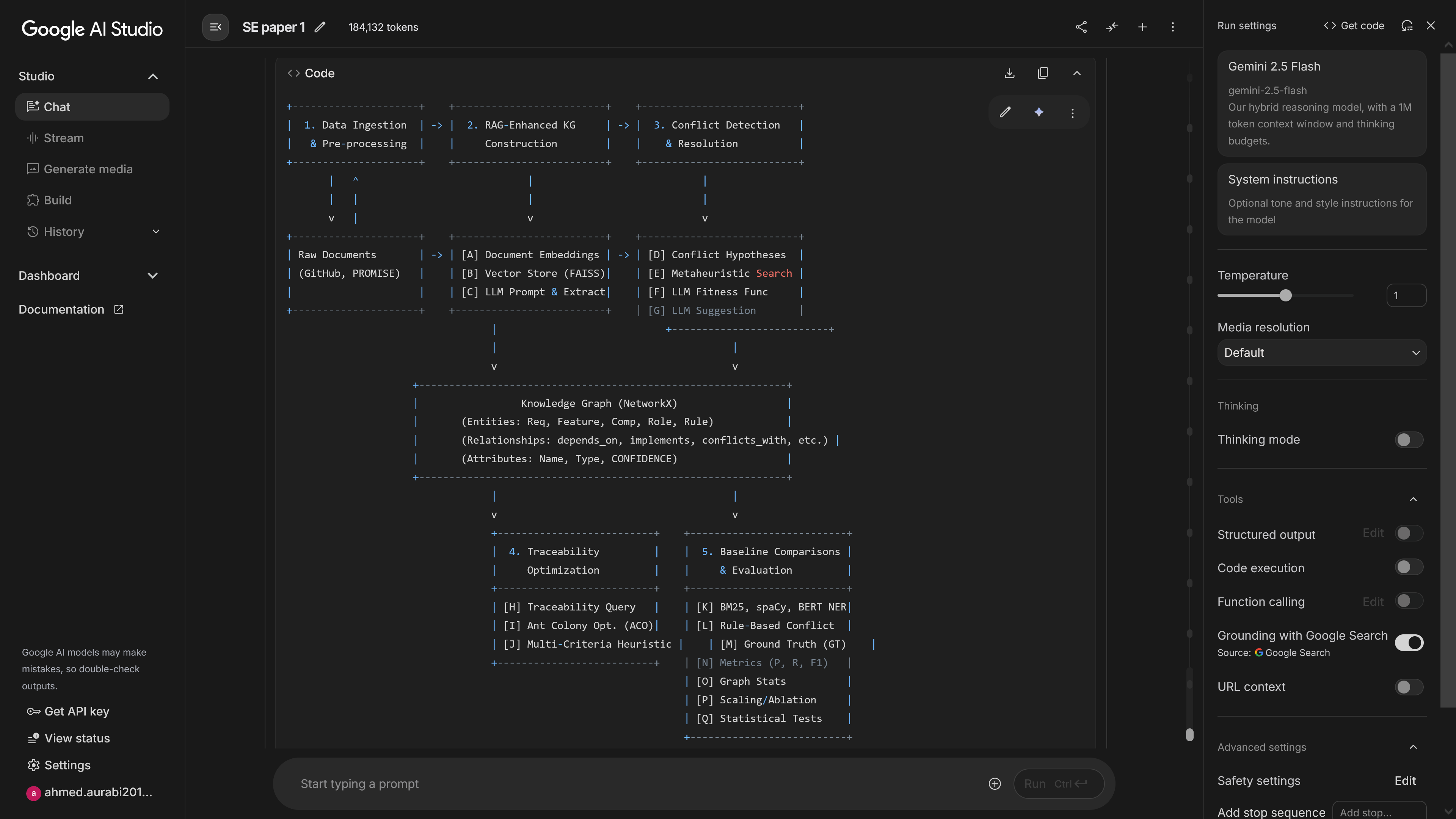
**Methodology: RAG-Enhanced Requirements Engineering for Conflict Resolution and Traceability Optimization**

The proposed methodology leverages a combination of Large Language Models (LLMs), Retrieval-Augmented Generation (RAG), Knowledge Graphs (KGs), and metaheuristic optimization techniques to dynamically construct and maintain a structured representation of software requirements, facilitating automated conflict detection, resolution, and optimized traceability.

**Step-by-Step Methodology:**

1. **Data Ingestion and Pre-processing:**
   * **Purpose:** To collect raw, unstructured requirements documents from diverse sources and convert them into a standardized text format suitable for downstream processing.
   * **Process:**
     + **Source Data Acquisition:** Raw documents (e.g., GitHub Issues, PROMISE ARFF files) are loaded from their respective sources (data\_loader.py).
     + **Standardization:** Each raw document is transformed into a consistent text format (Requirement ID: <ID>\nTitle: <Title>\nBody: <Body>...). This step ensures uniformity across different data sources.
2. **RAG-Enhanced Knowledge Graph Construction:**
   * **Purpose:** To dynamically build and update a Knowledge Graph by extracting entities and relationships from requirements, leveraging relevant context from other documents.
   * **Process:**
     + **Embedding and Vector Store Creation:** All pre-processed requirements documents are chunked and converted into numerical vector embeddings. These embeddings are stored in a FAISS vector database (kg\_builder.py:setup\_rag), enabling efficient semantic similarity search.
     + **Iterative Extraction with RAG:** For each requirement document:
       - **Context Retrieval:** A semantic search (RAG) query is performed against the vector store using the current document's content. Top-k semantically similar documents are retrieved, providing a rich, relevant context.
       - **Prompt Engineering:** The current document, along with the retrieved context and a detailed, XML-structured instruction set (including required output schema and few-shot examples), is fed as a prompt to a powerful, local LLM (e.g., Llama 3) (kg\_builder.py:\_create\_extraction\_prompt). The prompt explicitly requests confidence scores for extracted elements.
       - **LLM Extraction:** The LLM processes the prompt and generates a structured JSON output containing:
         * **Entities:** Requirements, Features, System Components, User Roles, Business Rules (each with an ID, type, name, and confidence score).
         * **Relationships:** depends\_on, conflicts\_with, is\_part\_of, implements, constrains (each with source, target, type, and confidence score).
       - **Graph Population:** The extracted entities and relationships (along with their confidence scores) are added to a NetworkX MultiDiGraph (kg\_builder.py:\_update\_graph), forming the Knowledge Graph. Robust error handling is in place to manage malformed LLM outputs.
3. **Conflict Detection and Resolution:**
   * **Purpose:** To automatically identify potential contradictions between requirements and suggest optimal resolution strategies.
   * **Process:**
     + **Conflict Hypothesis Generation:**
       - Explicit conflicts\_with relationships (if extracted by the LLM) are identified directly from the KG.
       - The LLM is prompted to perform semantic checks between requirements that interact with common entities (e.g., modifying the same feature) to identify latent conflicts (conflict\_resolver.py:detect\_conflicts, \_llm\_check\_conflict).
     + **Metaheuristic Resolution:** For each detected conflict, a metaheuristic algorithm (Hill Climbing, Simulated Annealing, or Genetic Algorithm) is employed to explore and select the best resolution strategy (conflict\_resolver.py:resolve\_with\_hill\_climbing, resolve\_with\_simulated\_annealing, resolve\_with\_genetic\_algorithm).
       - **Solution Space:** A predefined set of strategies (e.g., "Prioritize A", "Rephrase B", "Merge") constitutes the solution space.
       - **Fitness Function:** The quality of each strategy is evaluated by prompting the LLM (\_fitness\_function). The LLM acts as a semantic oracle, providing a score (1-10) based on factors like impact, consistency, and stakeholder satisfaction.
       - **Optimization:** The metaheuristic iteratively searches for a strategy that maximizes this LLM-judged fitness score, guiding the selection towards an optimal resolution.
     + **Resolution Suggestion:** Once an optimal strategy is found, the LLM is prompted again (\_get\_llm\_suggestion) to generate a human-readable, actionable suggestion based on the chosen strategy.
4. **Traceability Optimization:**
   * **Purpose:** To find optimal traceability paths between requirements or other entities in the KG, enabling efficient impact analysis.
   * **Process:**
     + **Traceability Query:** A query (e.g., "find all code impacted by changing User Story X") is translated into a graph pathfinding problem from a source node to a target node/type of node.
     + **Ant Colony Optimization (ACO):** An ACO algorithm is used to discover paths through the KG (traceability\_optimizer.py:find\_path\_with\_aco).
       - **Multi-Criteria Heuristic:** Ants navigate the graph, probabilistically choosing edges based on a combination of:
         * **Pheromone Trails:** Accumulated "experience" from previous ants that found good paths.
         * **Edge Confidence:** The LLM-generated confidence score of the extracted relationship (a higher confidence makes an edge more attractive).
         * **Path Length:** A heuristic favoring shorter paths.
       - **Pheromone Update:** Ants that find higher-quality paths (e.g., those with higher average confidence and reasonable length) deposit more pheromone, reinforcing these paths.
     + **Impact Analysis:** For a given starting requirement, the system can also identify all downstream impacted nodes by traversing the graph (e.g., find\_all\_impacted\_nodes).
5. **Baseline Comparisons and Evaluation:**
   * **Purpose:** To quantitatively assess the performance of the RAG-enhanced methodology against traditional and modern non-LLM baselines.
   * **Process:**
     + **Baselines:**
       - **BM25:** A classic information retrieval approach for keyword-based entity and relationship (explicit reference) extraction (bm25\_baseline\_extractor).
       - **spaCy:** A rule-based NLP approach using Named Entity Recognition (NER) for entity and heuristic relationship extraction (spacy\_baseline\_extractor).
       - **BERT NER:** A transformer-based NLP approach using a pre-trained NER model for entity extraction, with heuristic relationship inference (hf\_ner\_baseline\_extractor).
       - **Rule-Based Conflict Detection:** A simple pattern-matching approach for identifying conflicts (rule\_based\_conflict\_detector).
     + **Ground Truth (GT):** A subset of documents from each dataset is manually annotated by experts to create a "gold standard" KG, including entities, relationships, and conflicts.
     + **Metrics:** Precision, Recall, and F1-score are calculated for KG extraction (entities and relationships) and conflict detection (evaluate\_graph, evaluate\_conflicts).
     + **Graph Structural Metrics:** Properties like number of nodes/edges, density, average degree, and clustering coefficient are measured (graph\_struct\_stats).
     + **Performance Metrics:** Time taken per document for each method is recorded.
     + **Ablation Studies:** The impact of varying RAG k (number of retrieved documents) is analyzed.
     + **Scalability Studies:** Performance and quality are measured across different numbers of input documents.
     + **Statistical Tests:** Non-parametric tests (e.g., Wilcoxon signed-rank test) are used to determine statistical significance between method pairs (e.g., LLM-only vs. RAG-LLM).
6. **Results Presentation and Visualization:**
   * **Purpose:** To effectively communicate the findings of the experiment through plots, tables, and interactive visualizations.
   * **Process:**
     + **Tables:** Comprehensive CSV tables summarize all quantitative metrics for each method across all datasets.
     + **Plots:** matplotlib and seaborn are used to generate comparative bar charts (F1 scores), scatter plots (quality vs. latency), and line plots (ablation studies, scalability trends).
     + **Interactive KGs:** pyvis generates interactive HTML visualizations of the ground truth and the KGs produced by key methods, allowing for qualitative inspection.
     + **Human Evaluation Sheet:** A CSV is generated to facilitate external human review of a sample of extracted relationships.

**Block Diagram of the Methodology:**



A screenshot of a computer

AI-generated content may be incorrect.