Repository-to-Knowledge-Graph Conversion and GraphRAG

1. Repository/File to Knowledge Graph

When a repository or file is provided, it is parsed and analyzed to extract its structural and semantic components. The process involves:

- **Parsing and Static Analysis**: Source code is parsed using language-specific parsers to build Abstract Syntax Trees (ASTs) and extract entities such as functions, classes, modules, and their relationships (calls, imports, inheritance, etc.).
- **Documentation and Test Extraction**: Documentation files and test cases are also parsed to identify logical sections, test coverage, and links to code entities.
- **Chunking**: The codebase is divided into meaningful chunks:
    - **AST-based Chunking**: Code is chunked at the function, class, or module level, preserving logical boundaries and context.
    - **Sliding Window with Overlap**: For very large files, a sliding window (e.g., 100-200 lines with 20-40 line overlap) is used to maintain context across chunk boundaries.
    - **Hybrid Chunking**: For mixed content (code + docs), combine AST-based chunking for code and section-based chunking for documentation.
- **Graph Construction**: Each chunk becomes a node in the knowledge graph. Edges are created to represent relationships such as function calls, imports, test coverage, documentation links, and dependencies. The graph is continuously updated as the codebase evolves.

2. Establishing GraphRAG (Graph Retrieval-Augmented Generation)

- **Indexing and Embedding**: Each chunk/node is embedded using language models to enable semantic search and retrieval.
- **Graph-Driven Retrieval**: When the LLM is prompted (for root cause analysis, fix generation, etc.), it queries the knowledge graph to retrieve the most relevant chunks and their direct/indirect relationships (e.g., dependencies, callers, callees, related tests).
- **Context Assembly**: The retrieved subgraph (relevant nodes and their context) is assembled and provided as input context to the LLM, ensuring that the model has both the semantic and structural information needed for deep reasoning.

3. Benefits of This Approach

- **Scalability**: Chunking and graph-based retrieval allow the system to handle very large codebases efficiently, retrieving only the most relevant context for each query.
- **Explainability**: The knowledge graph provides a transparent, navigable structure that explains why certain code or documentation was retrieved and used by the LLM.
- **Precision**: By leveraging both semantic similarity and explicit code relationships, the system delivers highly targeted and context-rich information to the LLM, improving the quality of analysis and generated fixes.
- **Continuous Adaptation**: As the codebase changes, the knowledge graph and embeddings are updated, ensuring that retrieval and reasoning always reflect the latest state.