

SynapseDoc: Automated Documentation with a Multi-LLM-as-Judges Evaluation Framework for Critical Systems

Ikbel Ghrab¹ , Ines Abdeljaoued-Tej² 

1. Engineering School of Statistics and Information Analysis, University of Carthage, Tunis, Tunisia. E-mail: ikbel.ghrab@essai.ucar.tn
2. Corresponding author, Engineering School of Statistics and Information Analysis, University of Carthage, Tunis, Tunisia. E-mail: ines.tej@essai.ucar.tn

Article Info

Article type:
Research Article

Article history:
Received February 12, 2026
Received in revised form April 12, 2026
Accepted June 2, 2026
Available online June 8, 2026

Keywords:
automated documentation
multi-judge evaluation
prompt engineering
graph-based pipeline
semantic code analysis
software maintainability

ABSTRACT

Objective. This study introduces SynapseDoc, an automated documentation generation system for critical software systems, coupled with a novel multi-LLM-as-Judges evaluation framework. The research aims to automate the conversion of source code into pedagogically structured tutorials and establish a scalable, unbiased method for assessing the quality of AI-generated documentation.

Methods. The system is built on a modular pipeline architecture using the PocketFlow orchestration framework. It processes code through sequential nodes for repository fetching, abstraction identification, relationship analysis, chapter ordering, content generation, and final assembly. The core innovation is the evaluation framework, which employs four different Large Language Models as independent judges. They assess generated documentation across nine weighted metrics (including Accuracy, Faithfulness, Completeness, Clarity, and Hallucination) using a rigorous scoring methodology that includes normalization, weighted aggregation, and content density analysis.

Results. The framework was tested on PyMedPhys, a real-world open-source medical physics library. Quantitative evaluation revealed a clear performance hierarchy among eight state-of-the-art LLMs. The Gemini family outperformed others significantly. These results demonstrate the critical impact of model selection on documentation quality, particularly for healthcare applications where factual reliability is paramount, and validate the effectiveness of the multi-judge approach in providing a balanced, nuanced assessment that prioritizes accuracy and faithfulness alongside structural integrity.

Conclusion. This work demonstrates that reliable, automated code documentation is achievable by integrating diverse LLMs for generation and evaluation. Through SynapseDoc and a multi-judge framework with hallucination penalties, we established a methodology for selecting optimal models in safety-critical contexts. Gemini emerged as the best choice for healthcare applications, while the modular framework enables ongoing model selection as LLMs evolve. This approach sets a benchmark for AI-generated documentation, paving the way for more intelligent software engineering tools. Future work includes scaling to diverse codebases, IDE integration, and incorporating multimodal elements.

Cite this article: Ghrab, I., & Abdeljaoued-Tej, I. (2026). SynapseDoc: Automated Documentation with a Multi-LLM-as-Judges Evaluation Framework for Critical Systems. *Management Science and Information Technology*, XX(X), pages. <https://doi.org/10.22034/ISS.2026.10440.1071>



© Author(s) retain the copyright.
Publisher: International Scientific Services (ISS)
DOI: <https://doi.org/10.22034/ISS.2026.10440.1071>

1. Introduction

The manual creation and maintenance of source code documentation is a significant operational bottleneck in software engineering, often requiring weeks of developer effort. In safety-critical fields such as healthcare, fintech, and public-sector systems, clear and comprehensive documentation is not optional but mandatory to ensure reliability, transparency, and regulatory compliance. With advancing digitalization, software increasingly drives critical tasks and processes, including in medical device manufacturing, where it falls under the manufacturer's quality management system (QMS). Indeed, the growing use of big data and the need to ensure data reliability throughout operational and software chains are now critical, as illustrated in the medical domain (Musah, 2025). This challenge appears in logistics chain classification, where multiple roles and concurrent processes create identification ambiguity, for data collection, software processing, or analytical robustness (Tyapukhin, 2025). Such ambiguity leads to unexpected maintenance costs, with documentation debt recognized as a major cause of team slowdowns due to incomplete or outdated documentation (Charalampidou, 2019). It is precisely in this context that SynapseDoc intervenes: by automating reliable source code documentation, our tool reduces uncertainty, improves maintenance, and addresses the operational performance and agility challenges raised by management science and information technology.

Beyond facilitating maintainability and knowledge transfer, documentation serves as a core instrument of governance, enabling traceability of design decisions, data flows, risk mitigation strategies, and validation procedures across the system lifecycle (Sanders, 2008; Dagenais, 2010; Van der Spek, 2011; Chakrabarty, 2024; Tan, 2024). Emerging regulatory frameworks (such as the EU AI Act, the General Data Protection Regulation (GDPR), and sector-specific standards including ISO/IEC 42001 for AI management systems and ISO 13485 for medical devices) explicitly require structured technical documentation to guarantee auditability, explainability, accountability, and human oversight. Inadequate or outdated documentation may therefore lead not only to technical debt but also to compliance risks, legal exposure, and reduced stakeholder trust (Jelodar, 2025). While AI-based documentation tools have recently emerged, they frequently generate superficial summaries, lack methodological rigor, and provide limited mechanisms for quality assurance or objective evaluation (Mintlify, 2024). This need for standardized evaluation protocols is further highlighted by recent surveys on the performance of LLMs in code generation and analysis tasks (Chen et al., 2025). Our work proposed a modular pipeline based on PocketFlow that transforms raw source code into professional-grade tutorials, supported by a novel multi-judge scoring system designed to ensure consistency, depth, and evaluative reliability.

The evolution of language models has progressed through several key phases to address these documentation challenges. In the 1990s, early Statistical Language Models (Chen & Goodman, 1999; Rosenfeld, 2000; Chu et al., 2024) used n-gram probabilities based on Markov assumptions. While effective initially, they struggled with data sparsity and a rapid increase in parameters as n grew. Neural Language Models (Bengio et al., 2003) addressed these issues by introducing distributed word representations and feedforward neural networks. However, they lacked mechanisms for modeling sequence dynamics. Recurrent Neural Networks (Mikolov et al., 2010; Sherstinsky, 2020) were introduced to capture temporal dependencies, but suffered from vanishing gradients and difficulty with long-term context. LSTM (Hochreiter & Schmidhuber, 1997) and GRU (Chung et al., 2014) architectures improved this with gating mechanisms that preserved longer-range information.

A major breakthrough came with the Transformer architecture (Vaswani et al., 2017), which replaced recurrence with self-attention, enabling efficient parallelization and global context understanding. This paved the way for pre-trained models like BERT (Devlin et al., 2019) and GPT (Radford et al., 2018), which use self-supervised learning to acquire generalizable language representations. Modern Large Language Models (LLMs) such as GPT, LLaMA, DeepSeek, and Gemini build on these foundations, scaling Transformer architectures to billions of parameters and demonstrating capabilities like in-context learning and chain-of-thought reasoning (Zhao et al., 2023). However, leveraging LLMs effectively in real-world applications requires more than prompt engineering; it involves orchestration: managing interactions with data, tools, and humans across structured workflows. This has led to frameworks designed to simplify the development of LLM-powered systems. The proliferation of LLMs has catalyzed

a paradigm shift across specialized sectors, with applications ranging from financial assistance (Landolsi & Abdeljaoued-Tej, 2025) to help desk solutions based on Retrieval-Augmented Generation (RAG) (Tabib et al., 2026). These models are also used in document management (Baklouti et al., 2025), image editing (Charaa et al., 2025), and healthcare, including automated scoring of neuropsychological tests (Hamdi et al., 2025). Concurrently, Vision-Language Models have contributed to advances in text extraction and facial recognition for identity document processing (Benali et al., 2024). As these systems become integral to high-stakes environments, the manual creation of source code documentation has emerged as a critical bottleneck.

2. Materials and Methods

Modern Large Language Models (LLMs) are constructed upon the Transformer architecture, which leverages attention mechanisms to prioritize relevant input features during processing. The development of these models typically follows a three-phase pipeline consisting of self-supervised pretraining for foundational language understanding, supervised fine-tuning (SFT) to enable instruction following, and reinforcement learning from human or AI feedback (RLHF/RLAIF) to ensure alignment with specific values and safety standards. Effectively integrating these models into real-world applications necessitates LLM orchestration, which refers to the structured coordination of components that extend models beyond simple prompt-response interactions. Unlike one-shot prompting, orchestration introduces a high-level system supporting multi-step reasoning, memory management, and external tool integration, allowing LLMs to function as intelligent agents capable of handling evolving tasks and collaborating within guided workflows.

While established frameworks such as LangChain or LangGraph offer extensive integrations for building such workflows, they often introduce significant operational overhead and architectural complexity. Consequently, this study utilizes PocketFlow, a minimalist orchestration framework, designed for building complex AI workflows with minimal overhead, emphasizing simplicity, modularity, and vendor neutrality (Zhang et al., 2025). With a core codebase of approximately 100 lines and no external dependencies, it enables declarative workflow design and flexible integration with diverse LLMs. The architectural backbone of this framework is based on Nested Directed Graphs (NDG), a design that facilitates iterative logic and modular nesting while maintaining constant-time ($O(1)$) operations for flow transitions. This lightweight approach ensures precise control and rapid prototyping, making it ideal for scalable and collaborative human-AI development tasks.

The proposed system implements this orchestration to convert raw source code into structured, professionally detailed tutorials through a modular, multi-stage processing pipeline. A central component of this methodology is the innovative multi-LLM-as-judges evaluation framework, which addresses the inherent difficulty of validating generative outputs. This approach employs multiple independent LLMs to rigorously assess the quality of the generated documentation across defined metrics, including accuracy, coherence, completeness, and faithfulness. By utilizing diverse models as evaluators, the framework ensures a multi-perspective validation process that mitigates individual model biases and provides a robust, unbiased assessment of the final documentation assets.

2.1 Overview and System Architecture

The primary objective of SynapseDoc is to streamline the creation of technical documentation by automatically transforming source code repositories into professional, beginner-friendly tutorials. The system is designed to ingest code from either remote GitHub URLs or local file systems, systematically analyzing the codebase to identify key architectural concepts and relationships. By synthesizing these insights into structured Markdown files, the platform provides a clear and pedagogical overview of the software structure, bridging the gap between raw implementation and user comprehension.

To manage this complexity, the system employs a modular pipeline architecture built on the PocketFlow framework. This design decomposes the documentation task into a series of discrete nodes, where each component performs a specific functional role before passing its output to the next stage. Such a modular approach promotes system maintainability and scalability, allowing developers to modify or extend individual processing steps with minimal

impact on the overall workflow. This architecture ensures that the system remains flexible enough to adapt to various programming languages and repository structures.

A core innovation of this architecture is its deep integration with Large Language Models, which perform the sophisticated work of source code understanding and semantic relationship extraction. These models go beyond simple pattern matching to identify high-level abstractions, ensuring that the final tutorial content is conceptually aligned with the software logic. By leveraging the generative capabilities of these models, SynapseDoc produces documentation that is both technically precise and accessible to a broad audience, maintaining a human-readable quality that traditional automated tools often lack. An essential component enabling this capability is a carefully designed prompt engineering strategy. Rather than issuing generic instructions, the system constructs context-aware prompts that embed structured repository metadata, abstraction indices, relationship graphs, and selectively truncated source code excerpts. Each prompt explicitly assigns the model a professional role (such as senior software architect, senior system analyst, or instructional designer) to guide reasoning style, output structure, and level of abstraction. Output constraints (e.g., strict YAML schemas or Markdown formatting rules) are systematically enforced to ensure determinism and machine-parseable responses. By controlling context scope, limiting code excerpts for clarity, and embedding validation requirements directly into instructions, the pipeline reduces ambiguity and improves reproducibility. This structured prompting methodology transforms the LLM from a general-purpose text generator into a specialized architectural analysis engine embedded within the documentation workflow.

The full processing cycle is implemented through six sequential nodes, which coordinate the transition from raw code extraction to the final synthesis of a well-structured tutorial (see Figure 1). Each node, such as FetchRepo, IdentifyAbstractions, and AnalyzeRelationships, encapsulates a clearly defined task and contributes its intermediate results to a shared state that persists throughout the execution.

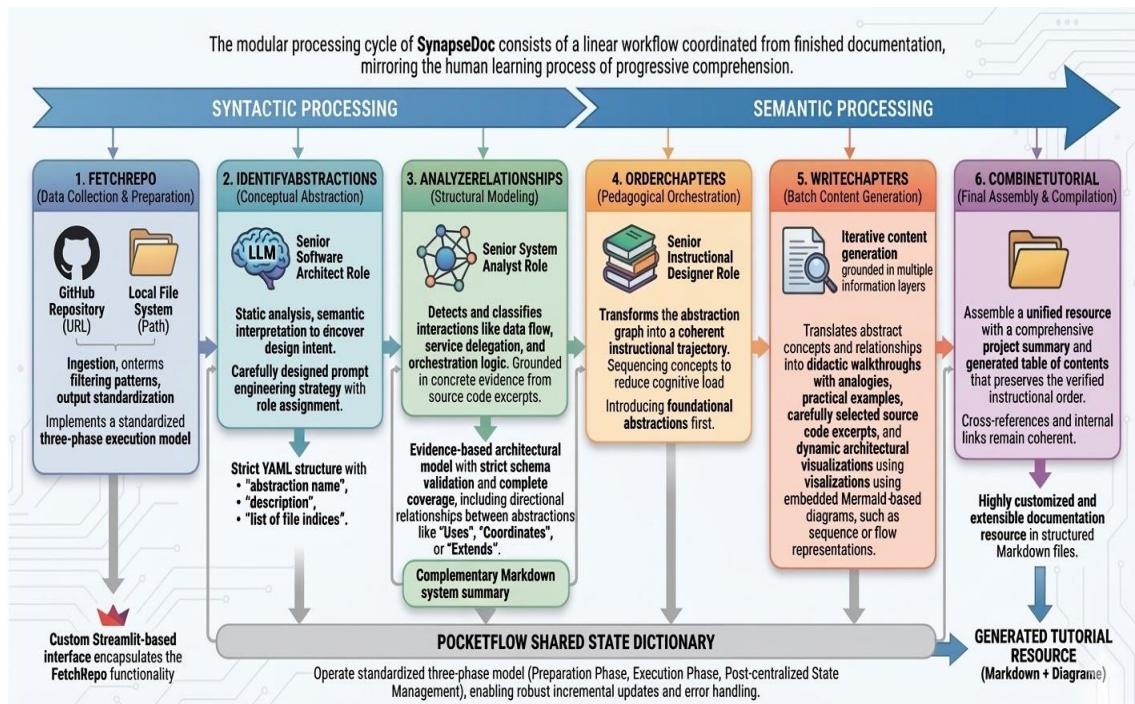


Figure 1. Modular processing cycle of SynapseDoc

This centralized state management ensures data consistency across all processing stages and facilitates a robust flow of information, allowing for a highly customized and extensible documentation pipeline.

2.2 Data Collection and Preparation: The FetchRepo Node

The foundational phase of the Documentation-Codebase-Knowledge pipeline is centered on data collection and preparation, a process managed exclusively by the FetchRepo node. This component functions as the primary ingestion point, responsible for retrieving, filtering, and structuring source code files from diverse repositories to establish a standardized input for downstream analysis. By supporting both local file system directories and remote GitHub repositories, the node provides significant flexibility in data source selection while ensuring consistent output formatting across different input modalities.

In alignment with the PocketFlow framework, the FetchRepo node implements a distinct three-phase execution model consisting of prep for configuration validation, exec for core processing, and post for result integration. This architectural separation enhances modularity and error handling by operating on a shared state dictionary that facilitates seamless data flow between pipeline components. The node prioritizes source agnosticism, allowing subsequent stages to process information uniformly regardless of its origin, while comprehensive filtering mechanisms minimize computational overhead by focusing analysis solely on relevant code artifacts.

The node retrieves its operational parameters from the shared state dictionary, which is typically initialized through command-line arguments or user inputs. The input specification requires exactly one mutually exclusive source identifier, either a GitHub repository URL or a local directory path, to enforce a clear distinction between data origins. To refine the analysis, the system employs a dual-pattern filtering system consisting of `include_patterns` for explicit file type selection and `exclude_patterns` to prevent the inclusion of auxiliary directories such as tests or metadata. Furthermore, size and access controls, including maximum file size limits and optional GitHub tokens for private repository access, ensure performance stability and prevent the processing of non-textual content.

Project identification is maintained through an optional project name parameter, which the node automatically infers from the source path or URL if not explicitly provided. To enhance accessibility and support non-technical users, a custom Streamlit-based interface was developed to encapsulate the FetchRepo functionality. This interface provides an intuitive form where users can specify repository details and dynamically apply filtering constraints through user-friendly controls, bridging the gap between complex backend orchestration and practical application usability.

2.3 Operational Workflow and Implementation

The operational workflow of the system is structured around a modular architecture comprising six discrete processing components, each designed to execute a specific stage of the documentation lifecycle. The process begins with FetchRepo, which ingests code from local directories or GitHub while applying rigorous filtering patterns to exclude irrelevant artifacts. This is followed by IdentifyAbstractions, where a large language model acting as a senior architect extracts five to ten core concepts and provides beginner-friendly analogies. The AnalyzeRelationships component then functions as a system analyst to model interactions between these abstractions, ensuring every component is represented in the architectural graph. To optimize the learning curve, OrderChapters sequences these concepts to minimize cognitive load, while WriteChapters generates the primary Markdown content, including Mermaid diagrams and concise code snippets. Finally, CombineTutorial assembles these outputs into a navigable resource featuring a table of contents and comprehensive project summaries.

The FetchRepo component, as the entry point of this pipeline, operates through a standardized three-phase model to ensure data integrity and source agnosticism. During the Preparation Phase, defined by the prep method, the system validates input parameters and confirms the source specification to ensure that exactly one GitHub URL or local directory is provided. This phase also derives project metadata and packages configuration settings into a preparation result dictionary to prevent runtime errors. The Execution Phase, handled by the exec method, then performs the actual data retrieval. For GitHub repositories, it utilizes specialized utilities to recursively crawl files via the GitHub API,

handling authentication and rate limiting. For local sources, it traverses the file system while respecting symbolic links and permissions. Both methods standardize the output by mapping relative file paths to their respective content, ensuring a uniform addressing scheme regardless of the ingestion modality.

The Post-Processing Phase, or post method, concludes the ingestion cycle by converting the gathered data into a standardized list of tuples, represented formally as:

$$\text{shared}[\text{"files"}] = [(\text{path}_1, \text{content}_1), \dots, (\text{path}_n, \text{content}_n)] \quad (1)$$

This structure ensures that each path_i is a normalized relative path and content_i represents the raw textual content. By centralizing this data in the shared pipeline state, the component eliminates the need for source-specific handling in downstream stages. This design allows subsequent elements, such as `IdentifyAbstractions`, to seamlessly consume the standardized output, thereby enhancing the overall flexibility, reliability, and scalability of the documentation pipeline.

2.4 Documentation Generation: From Code to Chapters

Following the successful completion of data collection and preparation by the `FetchRepo` component, the Documentation-Codebase-Knowledge pipeline transitions into its core analytical and generative phase. This sophisticated documentation process employs a sequence of five specialized stages (`IdentifyAbstractions`, `AnalyzeRelationships`, `OrderChapters`, `WriteChapters`, and `CombineTutorial`) that systematically leverage Large Language Models (LLMs) to transform raw source code into comprehensive, pedagogically structured tutorial documentation. This phase represents the intellectual foundation of the pipeline, implementing a systematic approach to code comprehension that mirrors human learning processes: identifying core concepts, understanding their relationships, organizing knowledge hierarchically, and presenting information in accessible narrative form.

The documentation generation architecture implements a progressive refinement model where each stage builds upon the outputs of its predecessors, creating increasingly sophisticated representations of the codebase's conceptual structure. This approach ensures that the final documentation maintains both technical accuracy and educational coherence, addressing the fundamental challenge of bridging the gap between complex implementation details and accessible learning materials.

2.4.1 Conceptual Foundation and Abstraction Identification

At the core of the documentation pipeline lies the task of establishing a conceptual understanding of the codebase. This begins with `IdentifyAbstractions`, which initiates the analytical process by detecting the system's essential abstractions, such as classes, modules, core functionalities, and architectural patterns. Unlike static code analysis tools that focus on syntax, this stage leverages semantic interpretation to uncover the codebase's design intent. To achieve this, the system constructs a concise, structured context tailored for interaction with a large language model (LLM). This context includes the project name and an indexed overview of the codebase, where each source file is represented by its relative path and a content preview. The index is then transmitted to the LLM through the `call_llm` utility, accompanied by a carefully designed prompt that instructs the model to behave as a senior software architect. The model is required to return a list of identified abstractions in a strict YAML format. Each entry must include: an abstraction name; a description; and a list of file indices identifying its occurrences in the codebase.

The decision to enforce YAML as the output format is deliberate and methodologically justified. YAML (YAML Ain't Markup Language) is a human-readable data serialization standard designed for both clarity and machine processing. Its indentation-based hierarchical structure naturally represents relationships between abstractions and associated files while remaining easy to validate and parse programmatically. YAML is extensively used in modern software engineering ecosystems, including configuration management in Kubernetes and automation frameworks such as Ansible, demonstrating its robustness and industry acceptance for structured system descriptions. Enforcing a

strict YAML schema in LLM outputs reduces ambiguity, improves determinism, and facilitates automated validation and downstream processing, thereby strengthening the reliability and reproducibility of the documentation pipeline.

Once the response is received, the system rigorously validates the output and verifies that each entry adheres to the expected schema. To ensure clarity and reduce redundancy, semantically similar abstractions are merged by comparing names and keywords, consolidating their associated file indices. The final curated list is stored in the shared state, forming a semantic blueprint for all downstream stages.

2.4.2 Relationship Analysis and Structural Modeling

Following the identification of core abstractions, the next stage deepens the architectural understanding of the system by modeling how these components interact to form a coherent and operational whole. The *AnalyzeRelationships* phase fulfills this objective by detecting, classifying, and structuring the architectural connections that underpin the codebase's behavior. Rather than merely listing dependencies, this step seeks to infer meaningful interaction patterns (such as data flow, orchestration logic, service delegation, and responsibility ownership), thereby transforming a static set of abstractions into a dynamic architectural model. To conduct this analysis, the system assembles a structured contextual input composed of the validated abstraction list and carefully selected source code excerpts. These excerpts serve as implementation-level evidence, grounding the model's reasoning in concrete logic (e.g., function calls, imports, class instantiations, or shared interfaces). This evidence-based prompting strategy reduces speculative inference and enhances semantic precision.

The LLM is then invoked with a carefully engineered prompt instructing it to take on the role of a senior system analyst, emphasizing architectural rigor, clarity of reasoning, and consistency in structural interpretation. The model is required to generate two complementary outputs:

- **System Summary:** A high-level architectural overview of the project's purpose and operational logic, written in Markdown format. This summary synthesizes the system's intent, core responsibilities, and overarching design philosophy.
- **Relationship List:** A strictly formatted YAML structure enumerating directional relationships between abstractions. Each entry specifies a source component, a target component, and a clearly labeled interaction type (e.g., Uses, Coordinates, Extends, Manages, Provides data to, or Depends on).

A critical constraint enforced at this stage is complete coverage: every identified abstraction must participate in at least one relationship, either as a source or a target. This guarantees architectural connectivity and prevents isolated components from being omitted in the structural representation. The YAML output is subjected to rigorous schema validation to ensure syntactic correctness and structural consistency.

If validation fails or if coverage constraints are not satisfied (specifically in the case of isolated utility classes or independent modules), the system avoids forcing functional hallucinations. Instead, a 'Logical Categorization Fallback' is triggered. This mechanism automatically assigns the isolated component a 'membership' relationship to the overarching system summary (e.g., 'Module X belongs to Utility Layer') rather than a false operational dependency (e.g., 'Module X interacts with Module Y'). This ensures 100% architectural connectivity for the final tutorial structure without compromising the technical faithfulness of the codebase's operational logic. Once validated, the relationship graph and the generated system summary together establish the structural and narrative backbone of the project.

This enriched architectural model not only clarifies component interactions but also serves as the conceptual foundation for subsequent tutorial generation, diagram synthesis, and higher-level documentation artifacts. Once the system's abstractions and their interrelationships have been formally identified, the pipeline transitions from structural analysis to pedagogical orchestration. This stage ensures that technical knowledge is not merely documented, but communicated in a logically progressive and cognitively accessible manner. The *OrderChapters* component fulfills this role by transforming the abstraction graph into a coherent instructional trajectory. To determine an optimal learning sequence, the system provides the LLM with both the validated abstraction metadata and the previously constructed relationship graph. The model is explicitly prompted to assume the role of a senior instructional designer, combining

architectural awareness with educational strategy. Rather than relying on arbitrary ordering, the LLM analyzes dependency directionality, conceptual prerequisites, and interaction density to propose a chapter structure that introduces foundational abstractions first, followed by progressively more complex or dependent components. This ensures that readers encounter core concepts before advanced coordination logic, thereby reducing cognitive overload. The proposed ordering undergoes strict validation to confirm that every abstraction index appears exactly once, guaranteeing structural completeness and eliminating duplication.

With a validated pedagogical sequence established, the WriteChapters component transforms each abstraction into a fully developed tutorial chapter. Implemented as a BatchNode, it iterates systematically over the ordered list, generating a dedicated Markdown chapter for every abstraction. Each generation step is context-aware and grounded in multiple information layers:

- The abstraction's name and formal description.
- The overarching system summary to preserve global coherence.
- Carefully selected source code excerpts (capped at 20 lines to maintain clarity and readability).
- Relevant relationships with other abstractions, ensuring architectural continuity across chapters.

The LLM is guided to adopt a didactic tone that balances technical precision with accessibility. Complex implementations are decomposed into smaller conceptual units, followed by structured, step-by-step walkthroughs that connect theory to implementation. When appropriate, intuitive analogies and practical examples are introduced to anchor abstract concepts in familiar mental models. To reinforce structural understanding and support visual learners, the generated content incorporates Mermaid-based diagrams (such as sequence diagrams, flow representations, or high-level architectural schematics) embedded directly within the Markdown output. These visual elements complement the textual explanation, enabling readers to grasp both the static organization and the dynamic interactions of the system. Through this stage, the pipeline evolves from architectural modeling to structured knowledge transmission, producing documentation that is not only technically accurate but also pedagogically optimized for progressive comprehension.

2.4.3 Final Assembly and Compilation

In the concluding stage of the pipeline, CombineTutorial consolidates all previously generated artifacts into a unified, structured, and fully navigable documentation resource (see Figure 2). This phase marks the transition from distributed content generation to cohesive compilation, ensuring logical continuity and structural integrity across the entire tutorial.

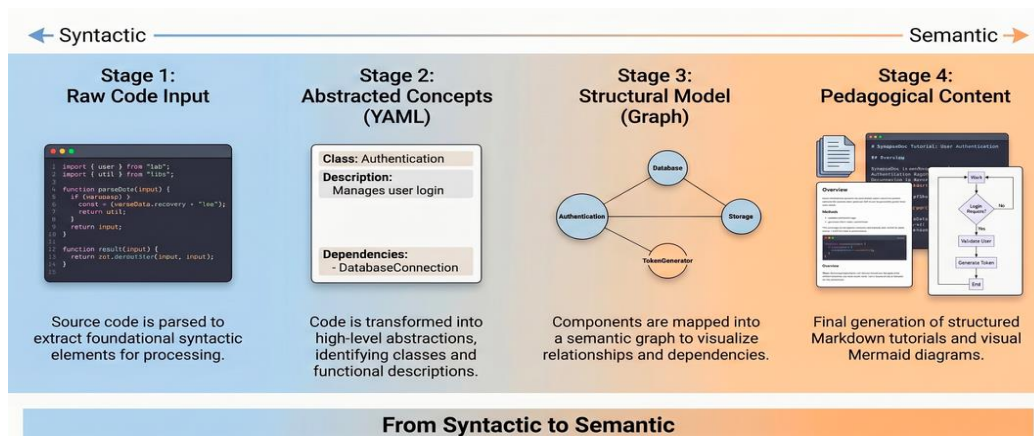


Figure 2. Progression from raw code syntax to meaningful human-centric knowledge

The component constructs a primary entry-point file that serves as the central index of the tutorial. This file begins with the validated project summary, offering readers an immediate high-level overview of the system's purpose and architectural organization. It then generates a table of contents that links to each chapter according to the verified instructional order established earlier in the pipeline. This guarantees alignment between the pedagogical sequencing and the final document structure. All individual Markdown chapters are written to the specified output directory using a consistent naming convention that preserves their validated order. Cross-references and internal links remain coherent, maintaining the conceptual flow developed throughout earlier stages. Once the files are successfully generated, the system records the path to the assembled tutorial in the shared state. This update serves as a completion signal for the pipeline, confirming that the documentation generation process has concluded and that the full tutorial is available as a single, organized resource.

3. Evaluation

Selecting an appropriate large language model (LLM) for automated code documentation requires both rigorous evaluation and cross-model consistency. To address this, we developed a scalable benchmarking framework and applied it to a real-world healthcare software repository: PyMedPhys, an open-source medical physics library (Biggs et al., 2022). This project's multidisciplinary nature, combining clinical computation, data management, and medical device interfacing, provides a robust and domain-relevant foundation for evaluating LLM documentation capabilities.

Eight LLMs across four major model families (Gemini, GPT, Qwen, and LLaMA) were employed to generate documentation for the PyMedPhys codebase. To determine which of these eight models is best suited for powering our documentation generation tool, we needed a reliable evaluation methodology that could assess the quality of their respective outputs.

Our evaluation process initially began using a conventional LLM-as-a-Judge approach, where a single LLM was used to assess the quality of the generated documentation. However, relying on a solitary judge model may introduce model-specific biases and limit the reliability of the evaluation. To address this, we draw on recent research investigating the factuality evaluation capabilities of LLMs and their potential as reliable judges (Fu et al., 2023), leading us to adopt a Multi-LLM-as-Judges ensemble strategy. From the original eight generation models, we selected four to serve as evaluators: one from each major model family (Gemini, GPT, Qwen, and LLaMA). These judge models independently reviewed the documentation produced by all eight generation models, assigning scores according to predefined evaluation metrics.

This ensemble approach offers several advantages. By incorporating diverse model perspectives, we minimize the influence of any single model's scoring bias and enhance the robustness of our assessments. The cross-family selection of judges ensures that evaluations reflect different architectural paradigms and analytical tendencies, enabling consensus-based quality estimation across diverse LLM architectures. Significant inter-model disagreements are automatically flagged for further analysis, providing insights into areas where evaluation criteria may be ambiguous or where documentation quality varies substantially across dimensions.

3.1 Evaluation Metrics

To comprehensively assess the quality of LLM-generated documentation, we defined nine core criteria designed to capture every facet of technical and pedagogical quality. These metrics are organized into three conceptual categories: Technical Accuracy, Structural Integrity, and Content Reliability, each addressing distinct dimensions of documentation effectiveness. All metrics are evaluated on a standardized scale, with scores aggregated through weighted fusion across the four judge models.

3.1.1 Technical Accuracy

This category measures the factual correctness and technical utility of the generated documentation, ensuring that developers can rely on it for implementation and troubleshooting.

- **Accuracy** assesses whether the documentation correctly describes the behavior, inputs, outputs, and side effects of the code. Inaccurate documentation can lead to misuse of functions and subtle bugs, making this metric paramount for clinical software where patient safety may depend on correct implementation.
- **Code Example Quality** evaluates the relevance, correctness, and clarity of any code snippets included in the documentation. High-quality examples should be runnable, illustrate typical use cases, and follow best practices.
- **Mermaid Diagram Quality** measures the effectiveness of any visual diagrams generated to illustrate workflows, data flow, class relationships, or algorithmic processes. For complex medical physics concepts such as DICOM RT treatment planning or radiation dose calculation pipelines, well-structured diagrams can convey relationships more efficiently than text alone. This metric considers diagram accuracy, clarity, and appropriate complexity.
- **Completeness** evaluates whether the documentation covers all essential aspects of the code, including parameters, return values, exceptions, edge cases, and dependencies. Incomplete documentation forces developers to read the source code directly, undermining the purpose of automated documentation generation.

3.1.2 Structural Integrity

While technical accuracy ensures the content is correct, structural integrity determines how effectively that content can be accessed and understood. This category examines documentation organization and presentation.

- **Coherence** measures the logical flow and consistency of the documentation. Well-structured documentation should present information in a logical order, maintain consistent terminology throughout, and avoid contradictory statements. For multi-function modules within PyMedPhys, coherence also encompasses how well the documentation explains relationships between related components.
- **Relevance** assesses whether the documentation focuses on information that is actually useful to its intended audience, primarily clinical physicists and researchers working with PyMedPhys. This metric penalizes content that, while factually correct, does not serve the reader's immediate needs. Unlike hallucination (which involves factual fabrication), relevance addresses pertinence, including information that is true but unnecessary can obscure critical details and reduce documentation efficiency.

3.1.3 Content Reliability

Beyond correctness and organization, documentation must be trustworthy and accessible. This category addresses the fundamental question of whether users can depend on the information provided.

- **Faithfulness** evaluates whether the documentation accurately reflects the actual implementation without speculation or unsupported claims (Malin, B. et al., 2025). Faithful documentation stays grounded in what the code actually does, rather than what it might ideally do or what similar libraries implement. This is particularly important for legacy code or functions with historical quirks that must be accurately documented.
- **Clarity** measures how easily the documentation can be understood by its target audience. Clear documentation uses appropriate language, explains complex concepts without oversimplification, and avoids ambiguous phrasing. For PyMedPhys's multidisciplinary audience (which includes clinical physicists, software developers, and researchers), clarity requires balancing technical precision with accessibility.
- **Hallucination Detection** identifies and penalizes fabricated information not grounded in the actual codebase. Hallucinations represent the most severe failure mode for LLM-generated documentation, as they

introduce misinformation that can mislead developers and potentially impact clinical safety (Khan, 2024). Examples include documenting non-existent parameters, inventing behavior for edge cases, or describing features that do not exist in the code. Due to the critical importance of factual reliability in healthcare contexts, Hallucination carries a substantially heavier penalty than any other metric in our aggregation process.

Each of these nine metrics is evaluated independently by all four judge models, with scores recorded and normalized for subsequent aggregation. The multi-perspective assessment across diverse LLM families helps mitigate individual model biases that might favor certain writing styles or penalize particular documentation characteristics. Through this comprehensive metric framework, we aim to identify LLM generators that produce documentation which is not only technically accurate but also structurally sound and reliably trustworthy for clinical software development contexts.

3.2 The Scoring Pipeline

To transform qualitative assessments into quantifiable, comparable scores, SynapseDoc employs a multi-stage scoring pipeline that systematically evaluates documentation quality through consensus-based fusion and penalized weighting. This pipeline processes the raw scores from all four judge models through five sequential stages, each designed to address specific aspects of documentation quality while maintaining mathematical rigor and interpretability.

3.2.1 Consensus and Normalization

For each documentation chapter, the individual scores assigned by the four independent judge models are first aggregated to establish a consensus baseline for each metric. This aggregation mitigates the influence of any single judge's idiosyncratic scoring tendencies and provides a more reliable estimate of true documentation quality.

The raw consensus score for a given metric is computed as the mean of the four judge-assigned scores:

$$\text{RawScore}_m = \frac{1}{4} \sum_{j=1}^4 \text{Score}_{j,m} \quad (2)$$

where $\text{Score}_{j,m}$ represents the score assigned by judge j for the metric m and the RawScore_m is the resulting consensus score for that metric. To ensure consistent weighted aggregation, these raw scores (on a 1–5 scale) are transformed into a standardized 0–1 scale:

$$\text{NormalizedScore}_m = \frac{\text{RawScore}_m - 1}{4} \quad (3)$$

3.2.2 Hierarchical Hallucination Penalty

Given the critical importance of factual reliability in safety-critical healthcare systems, the hallucination metric h undergoes a specialized two-step penalty process designed to disproportionately penalize models that generate fabricated content. This hierarchical approach ensures that hallucination receives substantially greater weight in the final score than its base allocation would suggest.

- **Score Inversion:**

$$\text{InvertedScore}_h = 1 - \text{NormalizedScore}_h \quad (4)$$

- **Penalty Amplification:**

$$\text{PenalizedScore}_h = \text{InvertedScore}_h \times \lambda_h \quad (5)$$

where the multiplier $\lambda_h = 2$ effectively doubles the negative impact of fabricated content on the final score. This amplification ensures that even moderate levels of hallucination meaningfully degrade a model's overall rating, reflecting the unacceptable nature of factual errors in clinical documentation.

3.2.3 Weighted Metric Integration

With all metrics normalized and the hallucination metric appropriately penalized, the processed scores are combined using a calibrated weighting scheme that reflects the relative importance of different quality dimensions. The weighted score is computed as:

$$\text{WeightedScore} = \sum_{m \in M} \text{ProcessedScore}_m \times \omega_m \quad (6)$$

where M represents the set of all evaluation metrics, ProcessedScore_m is the normalized score (or penalized score for hallucination) for metric m , and ω_m is the weight assigned to that metric.

The weights are distributed across three tiers based on each metric's contribution to documentation utility in clinical contexts:

- Core Metrics (Accuracy, Faithfulness, Completeness): 55%
- Secondary Metrics (Clarity, Coherence, Relevance): 30%
- Auxiliary Metrics (Code and Mermaid Quality): 10%
- Hallucination Penalty: 5% (prior to amplification).

This tiered weighting structure prioritizes factual correctness and completeness while still acknowledging the importance of presentation and supporting elements. The hallucination metric, despite its modest base weight, achieves effective parity with core metrics through the amplification mechanism described above.

3.2.4 Content Density Adjustment

Documentation quality depends not only on accuracy and structure but also on substance; the documentation must contain meaningful explanatory content rather than being empty, consisting of only a few sentences, or merely repeating code without explanation. To address this, the framework incorporates an automated content density analysis that evaluates whether each section contains sufficient original, explanatory text. The average section density is computed as:

$$\text{Density}_{\text{avg}} = \frac{\sum_{s=1}^S \text{WordCount}_s}{S} \quad (7)$$

where S represents the total number of sections in the documentation and WordCount_s is the word count for section s . Based on this analysis, a density weight ω_{density} is assigned according to predefined pedagogical thresholds.

Specifically, a weight of 0.7 is applied when $\text{Density}_{\text{avg}} < 50$ (very sparse), 0.8 when $50 \leq \text{Density}_{\text{avg}} < 100$ (sparse), 1.2 when $100 \leq \text{Density}_{\text{avg}} < 200$ (balanced), 1.0 when $200 \leq \text{Density}_{\text{avg}} < 300$ (dense), and 0.9 when $\text{Density}_{\text{avg}} \geq 300$ (very dense). In management contexts, documentation that is too brief poses a risk of omitting safety-critical implementation details. Conversely, to mitigate the risk of AI-generated 'verbosity' or 'fluff', the density factor operates in tension with Clarity and Relevance metrics.

This creates an Optimization Constraint: the model is incentivized to provide sufficient detail (Depth) without introducing redundant information that would obscure the core logic (Precision).

3.2.5 Final Score Computation

The weighted score is adjusted by the structural factor and reconverted to the 1–5 scale:

$$\text{FinalScore}_{1-5} = 1 + (\text{WeightedScore} \times \omega_{\text{density}} \times 4) \quad (8)$$

For a comprehensive assessment at the documentation set level, the final scores for all N chapters are averaged:

$$\text{SetScore}_{\text{avg}} = \frac{1}{N} \sum_{c=1}^N \text{FinalScore}_{1-5,c} \quad (9)$$

where N represents the number of chapters in the documentation set and $\text{FinalScore}_{1-5,c}$ is the final score for chapter c . This scoring methodology introduces several innovations that enhance the reliability and validity of AI-generated documentation evaluation. It adopts a multidimensional quality assessment approach that moves beyond single-metric or binary evaluations, capturing multiple aspects of documentation quality to provide a more comprehensive view of model performance.

Indeed, while our framework primarily relies on a ‘Weighted Aggregation’ to determine the final quality score, we have introduced a secondary statistical check to handle inter-model conflicts (such as a 2-2 tie). To ensure the reliability of the consensus baseline, the system monitors for inter-judge disagreement by calculating the variance σ_m^2 for each metric m across the four scores for each metric:

$$\sigma_m^2 = \frac{1}{4} \sum_{j=1}^4 (\text{Score}_{j,m} - \text{RawScore}_m)^2 \quad (10)$$

where $\text{Score}_{j,m}$ is the raw score from judge j and RawScore_m is the mean consensus score, defined in Section 3.2.1. In our framework, any documentation chapter where $\sigma_m^2 > \tau$ (the disagreement threshold 1.0) is subjected to a ‘Pessimistic Consensus Protocol’. This protocol overrides the automated FinalScore_{1-5} and flags the content for manual audit. By analyzing the variance of the judge panel, SynapseDoc identifies ‘Conflict Zones’ where LLM hallucinations or technical nuances cause model divergence, ensuring that safety-critical documentation is never approved on a simple majority basis if a significant conflict exists.

The framework also incorporates a penalty mechanism for hallucinations, ensuring that models generating inaccurate or fabricated information are appropriately penalized in order to prioritize those that produce reliable and factually consistent outputs. In addition, structural quality is integrated through automated content density analysis, which rewards well-organized and balanced documentation without requiring manual assessment.

The multi-judge evaluation strategy further strengthens robustness by reducing individual model bias and improving reproducibility. The modular design of the framework supports scalability, enabling efficient evaluation of large documentation datasets while maintaining quality standards.

4. Results

The evaluation results reveal clear performance tiers among the eight tested LLMs (see Table 1). The average final scores, computed under our weighted multi-metric system with hallucination penalties, distinctly differentiate model quality and behavior while providing actionable insights for model selection.

Table 1. Overall Scores for Documentation Generation Models (Weighted Evaluation)

Metric	Gemini 2.5-pro	Gemini 2.5-flash	GPT 4.1-nano	GPT 5-nano	Llama 3.2-11b	Llama 4-scout	Qwen3 32b	Qwen3 30b-a3b
Avg. FinalScore*	4.68	4.65	3.15	3.70	2.85	2.70	3.36	3.55
Avg. RawScore	4.52	4.45	3.46	3.89	3.15	2.90	3.44	3.50
Completeness	4.40	4.38	3.70	3.97	3.27	2.80	3.58	3.67
Clarity	4.80	4.36	3.97	4.50	3.92	3.55	4.03	4.17
Code Quality	4.55	4.47	3.28	3.82	2.85	2.57	3.30	3.40
Mermaid Quality	4.67	4.65	3.67	4.36	2.12	1.67	3.10	2.17
Coherence	4.87	4.95	4.07	4.73	4.15	3.80	4.25	4.32
Faithfulness	4.40	4.45	3.58	3.16	2.75	2.50	3.15	3.37
Hallucination	3.52	3.30	1.23	2.43	2.22	2.80	2.20	2.67
Accuracy	4.63	4.55	3.71	3.55	3.05	3.02	3.20	3.42
Relevance	4.88	4.97	3.90	4.53	4.02	3.42	4.15	4.32

*Final scores integrate weighted metrics and hallucination penalties.

4.1 Overall Performance Tiers

- **Top Tier:** The Gemini family dominates the highest performance tier, with gemini-2.5-pro (4.68) and gemini-2.5-flash (4.65) achieving nearly identical top scores. Both models demonstrate strong completeness, exceptional structural clarity, and the lowest hallucination rates among all evaluated models. Their performance is characterized by balanced documentation that is simultaneously comprehensive, accurate, and well-structured, a rare combination that proved decisive in the weighted scoring system.
- **Middle Tier:** A cluster of models occupies the middle tier, led by gpt-5-nano (3.70) and followed by qwen3-30b-a3b (3.55). These models maintain solid coherence and relevance despite exhibiting some trade-offs: gpt-5-nano shows tendencies toward verbosity that occasionally impacts faithfulness, while the Qwen variant delivers consistent if unremarkable performance across most metrics. qwen3-32b (3.36) and gpt-4.1-nano (3.15) occupy the lower end of this tier, with the GPT-4.1 variant's score substantially depressed by hallucination issues.
- **Lower Tier:** The LLaMA family comprises the lowest tier, with llama-3.2-11b (2.85) and llama-4-scout (2.70) trailing significantly behind all other models. Both variants struggle with code quality, Mermaid diagram generation, and factual stability, deficiencies that compound under the weighted penalty system. The 1.98-point spread between the top-performing Gemini model and the lowest LLaMA variant reflects substantial qualitative variation across the evaluated systems, underscoring the importance of careful model selection for documentation generation tasks.

4.2 Weighted vs. Raw Score Analysis

Comparing weighted final scores against unweighted raw averages provides critical insights into how the penalty mechanisms and metric weighting influence rankings, and reveals each model's specific strengths and weaknesses under rigorous evaluation.

- **Gemini Models:** Both Gemini variants exhibit weighted scores that exceed their raw averages (4.68 vs. 4.52 for pro; 4.65 vs. 4.45 for flash), indicating robust performance across heavily weighted core metrics and minimal hallucination penalties. This positive differential suggests balanced documentation quality that aligns well with our prioritization of accuracy and faithfulness over superficial fluency.

- **GPT Models:** The two GPT variants display markedly different behaviors. gpt-5-nano maintains relative stability (3.70 weighted vs. 3.89 raw), with the modest decrease attributable primarily to its elevated verbosity triggering the content density penalty in some sections. In contrast, gpt-4.1-nano suffers a sharp decline from 3.46 raw to 3.15 weighted, a drop of 0.31 points driven predominantly by two factors: severe hallucination issues (1.23, the lowest score across all models for this metric) and over-dense content distribution that incurred density penalties despite adequate raw scores in other dimensions.
- **LLaMA Models:** Both LLaMA variants consistently score below their raw averages after weighting (llama-3.2-11b: 2.85 vs. 3.15; llama-4-scout: 2.70 vs. 2.90). This negative differential reflects compounding penalties from factual inaccuracies (faithfulness scores below 3.0), poor code example quality, and near-total failure in Mermaid diagram generation (1.67–2.12). The weighted system correctly identifies that these models produce documentation that appears acceptable in raw averages but fails on dimensions critical for clinical software contexts.
- **Qwen Models:** The Qwen variants remain balanced, with minimal differences between weighted and raw scores (within 0.1 points). This consistency confirms their "normal" output profile: adequate across most dimensions but not exceptional in either direction. Neither model excels sufficiently to challenge the top tier nor fails badly enough to join the lower tier.

4.3 Metric-Level Analysis

4.3.1 Core Metrics (Combined Weight: 55%)

The core metrics (Accuracy, Faithfulness, and Completeness) exerted the strongest influence on final rankings, as designed.

- Gemini-2.5-pro achieved consistently high marks across all three: Accuracy (4.63), Faithfulness (4.40), and Completeness (4.40). gemini-2.5-flash followed closely with a nearly identical profile (4.55, 4.45, 4.38), demonstrating that the flash variant sacrifices little factual quality despite its lighter architecture.
- gpt-5-nano performed respectably on Accuracy (3.55) and Completeness (3.97) but exhibited weaker Faithfulness (3.16), indicating a tendency to include plausible-sounding but unsupported statements. This uneven factual grounding explains its placement below the Gemini tier despite strong performance on structural metrics.
- gpt-4.1-nano showed a curious inversion: higher Faithfulness (3.58) than Accuracy (3.71) or Completeness (3.70), suggesting conservative generation that avoids speculation but also omits necessary detail.
- Both LLaMA models scored below 3.0 across Faithfulness and Completeness (2.75–2.80 range), with Accuracy similarly depressed (3.02–3.05). These scores reflect weak grounding in the actual codebase and fragmented document structure that fails to capture complete function specifications.
- The Qwen variants maintained moderate reliability in the 3.2–3.7 range across all core metrics, consistent with their middle-tier classification. qwen3-30b-a3b marginally outperformed its 32B counterpart, suggesting that architecture matters more than parameter count for documentation tasks.

4.3.2 Secondary Metrics (Combined Weight: 30%)

Secondary metrics (Clarity, Coherence, and Relevance) capture the structural and semantic quality that determines how easily documentation can be used.

- Both Gemini variants again led this category, with gemini-2.5-flash achieving near-perfect scores for Coherence (4.95) and Relevance (4.97), the highest individual metric scores recorded in the entire evaluation. This indicates exceptional ability to organize information logically and focus on content that matters to clinical physicists.

- gpt-5-nano produced coherent and logically structured outputs (4.73 Coherence) but exhibited a "content-dense" generation style that occasionally buried critical information under excessive detail, reflected in its slightly lower Clarity (4.50) relative to Coherence.
- gpt-4.1-nano maintained adequate secondary scores (3.90–4.07 range) despite its core metric deficiencies, demonstrating that even models with factual instability can produce well-structured text, an important reminder that structural quality cannot substitute for factual reliability.
- The Qwen models delivered clear, relevant documentation (Clarity 4.03–4.17, Relevance 4.15–4.32) with solid coherence (4.25–4.32), confirming their reputation as reliable middle-ground performers.
- The LLaMA family displayed inconsistent clarity and lower relevance scores (3.42–4.02), reflecting less controlled output structure and tendency to include tangential information that, while factually correct, does not serve the reader's immediate needs.

4.3.3 Auxiliary Metrics (Combined Weight: 10%)

Auxiliary metrics capture technical and structural fidelity through Code Example Quality and Mermaid Diagram Quality, dimensions that separate exceptional documentation from merely adequate alternatives.

- The Gemini models again dominated, with near-identical code quality scores (≈ 4.5) and the highest Mermaid diagram quality (≈ 4.6). Their ability to generate accurate, runnable code examples and clear visual diagrams of clinical workflows (e.g., DICOM RT treatment planning sequences) adds substantial practical value for users.
- gpt-5-nano followed closely with strong Mermaid quality (4.36) and solid code examples (3.82), indicating good diagrammatic alignment despite occasional verbosity in explanatory text.
- gpt-4.1-nano showed a sharp drop in code quality (3.28) relative to its successor, suggesting improvements in the 5-series for technical example generation.
- The LLaMA models demonstrated clear limitations, particularly in diagram rendering (1.67–2.12), scores that indicate near-total failure to generate usable Mermaid diagrams. Code example quality was similarly depressed (2.57–2.85), with examples often incomplete, non-runnable, or demonstrating poor practices.
- The Qwen models achieved reasonable but unremarkable technical precision (code 3.30–3.40, Mermaid 2.17–3.10), with the 30B variant notably weaker on diagram generation despite comparable text performance.

4.3.4 Hallucination Assessment

Hallucination scores provide the most critical measure of factual stability for clinical software documentation, where fabricated information could have patient safety implications.

- Gemini maintained moderate-to-low hallucination rates (3.30–3.52), indicating occasional minor fabrications but no systematic tendency to invent functionality. This reliability underlies their top-tier positioning despite not achieving the absolute highest raw scores in every category.
- GPT variants showed substantial variation. gpt-4.1-nano suffered from significant factual drift (1.23), the lowest hallucination score recorded, frequently documenting non-existent parameters or inventing behavior for edge cases. This single deficiency explains its dramatic weighted score decline and renders it unsuitable for clinical applications despite adequate structural metrics. gpt-5-nano improved substantially (2.43) but still exhibits more hallucination than Gemini models.
- The LLaMA and Qwen families clustered around moderate values (2.2–2.8), reflecting general reliability but occasional factual inconsistencies. Neither family achieves the factual precision required for top-tier clinical documentation, though they may suffice for less safety-critical applications.

These hallucination penalties explain much of the gap between raw and weighted outcomes for models with otherwise acceptable raw averages, particularly for gpt-4.1-nano, whose 1.23 hallucination score triggered the full force of the hierarchical penalty mechanism.

4.4 Judge Consensus and Framework Reliability

The multi-LLM judging framework demonstrated strong reliability, with high inter-model agreement across all four evaluators. All judges independently produced the same relative ranking of generator models: Gemini variants at the top, GPT and Qwen in the middle tier, and LLaMA models at the bottom, confirming that the observed performance differences reflect genuine quality variation rather than evaluator bias or architectural idiosyncrasy. Beyond confirming rankings, each model family exhibited characteristic behavioral patterns consistently identified by all judges: Gemini variants produced balanced, factually grounded outputs with minimal trade-offs across metrics; GPT models delivered structurally excellent documentation but showed a recurring faithfulness gap, trading factual precision for fluency; LLaMA variants were consistently deficient across nearly all dimensions, particularly in diagram generation where scores approached total failure; and Qwen models maintained moderate, evenly distributed performance with no extreme strengths or weaknesses. Statistical analysis revealed minimal inter-judge variance, with strongest agreement on objective metrics such as Code and Mermaid Quality, and slightly lower but still robust consensus on more interpretive metrics like Relevance and Clarity. Where significant disagreements occurred (primarily around borderline cases in the middle tier), the framework automatically flagged these for qualitative review as designed. The high consensus across architecturally diverse judges confirms that the ensemble strategy successfully mitigates individual model bias, while the finding that balanced generators like Gemini showed the highest inter-judge agreement and lowest inter-metric variance underscores that consistent quality across all dimensions, rather than excellence in any single area, produces the most reliable documentation for clinical applications.

5. Discussion

While this study provides a robust foundation for evaluating LLM-generated documentation, several limitations must be acknowledged. The evaluation was conducted on a single project within the healthcare domain (PyMedPhys) and relied exclusively on freely accessible LLM APIs. These constraints restricted experimentation with high-performance commercial models and prevented large-scale evaluations across diverse programming languages, frameworks, and project complexities. The findings, therefore, may not fully generalize to other domains such as financial software, embedded systems, or safety-critical infrastructure, which may require more specialized documentation strategies and domain-informed model behavior.

Additionally, the current evaluation framework depends solely on automated LLM-based assessment without the inclusion of human expert review. Integrating domain specialists (such as clinical physicists for healthcare software or regulatory experts for compliance-critical systems) could provide richer contextual insight and improve the interpretive reliability of results. The absence of such expert input remains a key limitation, particularly for assessing documentation quality in specialized or high-stakes fields where nuanced understanding of domain terminology and workflow logic is essential.

The models tested in this study were general-purpose LLMs without domain-specific fine-tuning. This likely limited their ability to produce documentation that is both contextually accurate and compliant with domain regulations. In healthcare settings, for instance, achieving high-quality documentation requires not only linguistic precision but also a solid understanding of medical terminology, regulatory requirements (e.g., FDA 21 CFR Part 820), and clinical workflow logic. The performance gap between top-tier models and their competitors suggests that architectural choices and training methodologies matter more than raw parameter count, but even the best general-purpose models may benefit from targeted domain adaptation.

Despite these limitations, the proposed multi-LLM evaluation framework represents a foundational contribution toward systematic, cross-domain benchmarking of documentation generation quality. The inter-judge consensus and performance tiers validate the methodology's reliability and suggest several promising directions for future research:

- **Broader Validation:** Expanding evaluations to encompass diverse programming languages, frameworks, and large-scale open-source systems (e.g., Apache projects, Linux kernel modules) would improve generalizability and cross-domain robustness. Applying the framework to non-healthcare domains would test whether the observed model rankings hold across different documentation contexts.
- **Domain Fine-Tuning:** Adapting and fine-tuning LLMs for specialized domains, particularly healthcare, could enhance factual precision, regulatory compliance, and contextual awareness. Incorporating domain-specific corpora and standards such as HL7 FHIR, DICOM RT, or FDA quality system regulations may enable generation of compliant, domain-aware documentation that surpasses general-purpose capabilities.
- **Expert-Informed Evaluation:** Integrating domain experts into the assessment process would serve multiple purposes: validating LLM-based evaluations against human judgment, refining scoring methodologies based on expert feedback, and curating high-quality training data for future judge model development. A hybrid human-AI evaluation framework could combine scalability with nuanced understanding.
- **Healthcare Specialization:** Prioritizing documentation tasks related to clinical workflows, safety reviews, and regulatory audits could accelerate adoption in medical settings. The development of healthcare-specific evaluation models trained on domain literature (journal articles, clinical guidelines, regulatory documents) could further improve assessment precision and relevance.
- **Framework Expansion:** Extending the current evaluation framework with additional domain-sensitive metrics, user feedback integration, and CI/CD compatibility would enable continuous, automated monitoring of documentation quality as both codebases and LLMs evolve. This would transform benchmarking from a one-time activity into an ongoing quality assurance process.

The issues encountered in software documentation are part of a broader problem of data and model governance (Fattah et al., 2025). Documentation automation, such as that provided by SynapseDoc, reduces uncertainty in the software value chain, thereby improving operational performance from an information systems management perspective. Our analysis shows that the computational cost of running four judge LLMs on a typical codebase (e.g., PyMedPhys, ~50k lines of code) is approximately \$1.34 per generation-evaluation cycle using current API pricing. Scaling to 100 repositories per month would represent an insignificant expense for any industrial R&D budget. In contrast, the human cost of manual documentation and review is substantial: developers lose more than eight hours per week due to insufficient documentation. This observation is consistent with recent findings from Anthropic (Shen & Tamkin, 2025), which report that engineers now use AI for about 60% of daily tasks, achieving a 50% productivity boost – a marked increase from 20% the previous year. Notably, 27% of AI-assisted work involved documentation improvements and small quality-of-life fixes. These results align with our own: AI-driven documentation automation directly enhances developer productivity. Consequently, the computational cost of SynapseDoc's multi-LLM judge framework is orders of magnitude lower than the human cost it replaces, offering a clear positive return on investment.

6. Conclusion

We developed SynapseDoc, a documentation generation tool that leverages a LLM to automatically produce code documentation. A critical challenge in building such a tool is selecting the optimal LLM backbone: the model that best follows prompts and instructions while generating accurate, well-structured documentation. To address this, we employed eight LLMs across four major model families to generate documentation for PyMedPhys, an open-source medical physics library, and evaluated their outputs using a novel multi-LLM-as-Judges framework with four independent evaluators.

The evaluation employed nine weighted metrics organized into three conceptual categories (Technical Accuracy, Structural Integrity, and Content Reliability) to assess documentation completeness, clarity, factual accuracy, and coherence. Results revealed that Gemini variants consistently outperformed all other tested models, with gemini-2.5-pro (4.68) and gemini-2.5-flash (4.65) achieving the highest weighted scores through balanced, factually grounded documentation that excelled across all metric categories. Their strong performance across core metrics (Accuracy, Faithfulness, and Completeness) combined with low hallucination rates and exceptional structural clarity, establishes them as the optimal choice for powering SynapseDoc in clinical software contexts.

The multi-LLM judging framework demonstrated strong reliability, with high inter-model agreement across all four evaluators and stable model ordering that confirmed performance differences reflect genuine quality variation rather than evaluator bias. The hierarchical hallucination penalty and content density adjustment mechanisms proved effective at identifying models that produce fluent but factually unreliable documentation, reinforcing the importance of factual stability for clinical applications.

By utilizing PocketFlow for workflow orchestration, we efficiently coordinated the complex pipeline of documentation generation, multi-model evaluation, and systematic aggregation of results, all while maintaining modularity and reproducibility. This architectural approach ensures that as new models emerge, SynapseDoc can be seamlessly updated with the latest optimal LLM without requiring wholesale reconfiguration.

Overall, this study establishes a foundation for scalable, cross-model benchmarking of documentation generation and provides a methodology for selecting the optimal LLM for tools like SynapseDoc. With Gemini identified as the current best-fit model and potential for domain-specific adaptation in healthcare and other regulated fields, this work offers a path toward automated, reliable code documentation that leverages the most capable LLMs for the task.

Author Contributions

All authors contributed equally to the conceptualization of the article and writing of the original drafts.

Data Availability Statement

Data available on request from the authors.

Acknowledgements

The authors would like to thank Ismail Khenissi and Mohamed Dhieb for their continued assistance for this work. The authors are deeply grateful to the anonymous referees, whose thoughtful comments and constructive suggestions have enriched the manuscript.

Ethical considerations

The authors avoided data fabrication, falsification, and plagiarism, and any form of misconduct.

Funding

The study was funded by Easy Transfer company and LR16IPT09 laboratory at Institut Pasteur de Tunis.

Conflict of interest

The authors declare no conflict of interest.

Declaration of Generative AI and AI-assisted technologies in the writing process

The authors used Google Gemini and DeepSeek for language editing, structural organization, and clarity enhancements. No part of the manuscript was generated by AI without human verification. The final content was reviewed and edited by the authors, who assume full responsibility for its integrity and accuracy.

References

- Baklouti, Y. B., Echcherqaoui, T., & Abdeljaoued-Tej, I. (2025). Smart enterprise document access with a chatbot powered by azure openai. *10th International Congress on Information and Communication Technology (ICICT'2025)*. Springer. https://doi.org/10.1007/978-981-96-6932-5_35
- Benali, H., Melliti, M., & Abdeljaoued-Tej, I. (2024). Modèle de langage de vision pour l'extraction de texte en arabe: detection dans des cartes d'identité et reconnaissance faciale. *11ème Conférence Internationale en Automatique & Traitement de Signal (ATS'2024)*.
- Bengio, Y., et al. (2003). A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, 3, 1137–1155.
- Biggs, S., Jennings, M., Swerdloff, S., Chlap, P., Lane, D., Rembish, J., ... & Sobolewski, M. (2022). PyMedPhys: A community effort to develop an open, Python-based standard library for medical physics applications. *Journal of Open Source Software*, 7(78), 4555. <https://doi.org/10.21105/joss.04555>
- Chakrabarty, S., & Pal, S. (2024). Free and customizable code documentation with LLMs: A fine-tuning approach. *Preprint arXiv*. <https://arxiv.org/abs/2412.00726>
- Charaa, T., Hamdeni, T., & Abdeljaoued-Tej, I. (2025). Editing fashion images with precision: A controlled inpainting method. *The International Journal of Multimedia & Its Applications (IJMA)*, 17(3). <https://doi.org/10.5121/ijma.2025.17302>
- Charalampidou, S. (2019). Managing technical debt through software metrics, refactoring and traceability. [Thesis fully internal (DIV), University of Groningen]. University of Groningen. https://research.rug.nl/files/84473801/Chapter_4.pdf
- Chen, S. F., & Goodman, J. (1999). An Empirical Study of Smoothing Techniques for Language Modeling. *Computer Speech & Language*.
- Chen, L., Guo, Q., Jia, H., Zeng, Z., Wang, X., Xu, Y., Wu, J., ... & Zhang, S. (2025). A survey on evaluating large language models in code generation tasks. *Preprint arXiv*. <https://arxiv.org/abs/2408.16498>
- Chu, Z., et al. (2024). History, Development, and Principles of Large Language Models: An Introductory Survey. *Preprint arXiv:2402.06853*.
- Chung, J., et al. (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *Preprint arXiv:1412.3555*.
- Dagenais, B., & Robillard, M. P. (2010). Creating and evolving developer documentation: understanding the decisions of open source contributors. *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering* (p. 127–136). Association for Computing Machinery. <https://doi.org/10.1145/1882291.1882312>
- Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 4171–4186.
- Fattah, M., Ghobadi, N., & Ziari, M. (2025). Emerging Paradigms in Supply Chain Resilience Under Crisis Circumstances: A Roadmap for Future Investigations. <https://doi.org/10.22034/ISS.2025.9049.1038>
- Fu, X.-Y., Laskar, M. T. R., Chen, C., & Tn, S. B. (2023). Are large language models reliable judges? a study on the factuality evaluation capabilities of LLMs. Dans S. Gehrmann et al. (Éds.), *Proceedings of the Third*

- Workshop on Natural Language Generation, Evaluation, and Metrics (GEM)* (p. 310–316). Association for Computational Linguistics. <https://aclanthology.org/2023.gem-1.25/>
- Hamdi, I., Lam-Weil, J., Martz, E., Abdeljaoued-Tej, I., & Capobianco, A. (2025). An automatic scoring method for responses to the hayling test. *Proceedings of the 37th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'2025)*. <https://doi.org/10.1109/ICTAI66417.2025.00181>
- Jelodar, H., Meymani, M., & Razavi-Far, R. (2025). Large language models (LLMs) for source code analysis: Applications, models and datasets. *Preprint arXiv*. <https://arxiv.org/abs/2503.17502>
- Khan, M. F. (2024). Evaluation of quantitative and qualitative metrics for assessing hallucination phenomena in LLMs. *Nuvern Applied Science Reviews*, 8(10), 38–47. <https://nuvern.com/index.php/nasr/article/view/2024-10-16>
- Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), 1735–1780.
- Landolsi, H., & Abdeljaoued-Tej, I. (2025). The future of financial assistance: Leveraging LLMs for personalized, human-like interactions. *International Journal of Web & Semantic Technology (IJWest)*, 16(1). <https://doi.org/10.5121/ijwest.2025.16102>
- Malin, B., Kalganova, T., & Boulgouris, N. (2025). A review of faithfulness metrics for hallucination assessment in large language models. *IEEE Journal of Selected Topics in Signal Processing*. <http://doi.org/10.1109/JSTSP.2025.3579203>
- Mikolov, T., Karafiát, M., Burget, L., Cernocký, J., & Khudanpur, S. (2010). Recurrent neural network based language model. In *Interspeech* (Vol. 2, No. 3, pp. 1045-1048).
- Mintlify. (2024). Beautiful docs for your codebase. <https://mintlify.com/>
- Musah, L. (2025). Enhancing Supply Chain Performance and Agility in the Healthcare Industry Through Big Data Analytics: A Complex Adaptive Systems Theory Approach. *Management Science and Information Technology*, 2(1), 1-19. https://msit.refconf.com/article_2662.html
- Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving language understanding by generative pre-training. *OpenAI*.
- Rosenfeld, R. (2002). Two decades of statistical language modeling: Where do we go from here?. *Proceedings of the IEEE*, 88(8), 1270-1278.
- Sanders, R., & Kelly, D. (2008). Dealing with risk in scientific software development. *IEEE Software*, 25(4), 21–28. <http://doi.org/10.1109/MS.2008.84>
- Shen, J. H., & Tamkin, A. (2026). How AI impacts skill formation. *Preprint arXiv:2601.20245*. <https://arxiv.org/abs/2601.20245>
- Sherstinsky, A. (2020). Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network. *Physica D: Nonlinear Phenomena*, 404, 132306.
- Tabib, M., Abdeljaoued-Tej, I., & Raouafi, H. (2026). Enhancing large language models with retrieval augmented generation for a trusted help desk solution. *Lecture Notes in Networks and Systems (LNSS)*, 1606. https://link.springer.com/chapter/10.1007/978-3-032-03695-7_3
- Tan, W. S., Wagner, M., & Treude, C. (2024). Detecting outdated code element references in software repository documentation. *Empirical Software Engineering*, 29(1), 5. <https://doi.org/10.1007/s10664-023-10397-6>
- Tyapukhin, A. P. (2025). Classification, Digitalization and Structure of Integrated Logistics Chains. *Management Science and Information Technology*, 2(3), 189-215. <https://doi.org/10.22034/ISS.2025.9029.1031>

- Van der Spek, P., Klusener, S., & van de Laar, P. (2010). Complementing software documentation. In *Views on evolvability of embedded systems* (pp. 37-51). Dordrecht: Springer Netherlands. https://doi.org/10.1007/978-90-481-9849-8_3
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 5998–6008
- Zhang, H., Haskell, J., & Frost, Y. (2025). Flow state: Humans enabling AI systems to program themselves. *Preprint arXiv*. <https://arxiv.org/abs/2504.03771>
- Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., ... & Wen, J. R. (2023). A survey of large language models. *Preprint arXiv:2303.18223*, 1(2), 1-124.