IUFlowGen: Converting Procedural Documents into Structured Diagrams

Abstract. Procedural documents are common in domains like technical operations and legal compliance, but their unstructured and complex logic hinders understanding. Converting such text into flowcharts improves clarity and reduces cognitive load, and recent AI advances show strong potential for automating this process. However, current AI systems often fall short in real-world scenarios: they typically generate static outputs without clarification mechanisms, fail to adapt to varying document difficulty, and lack any formal strategy for determining when automation is genuinely helpful. To address these gaps, we argue that effective systems must include interactive clarification features and a principled method for deciding when AI assistance is appropriate. In this paper, we present **IUFlowGen**, an AI-assisted, human-in-the-loop system that combines retrieval-augmented generation and graph-based modeling to plot flowcharts that help users better understand procedural content. To support the intelligent deployment of AI assistance, we also propose a rubric-based framework that quantifies procedural document complexity. Experiments show IUFlowGen effectively handles documents of varying complexity and produces flowcharts aligned with the original procedural intent.

Keywords: Flowchart generation · Procedural documents · Natural language understanding · Graph visualization · Evaluation metrics

1 Introduction

Procedural documents in domains such as engineering and public administration often contain multi-step instructions, conditional logic, and domain-specific terminology written in unstructured natural language. This makes them difficult to interpret. Converting such documents into flowcharts is a proven method to improve clarity, reduce errors, and support collaboration and downstream verification [10].

Recent advances in large language models (LLMs) [15] have shown the potential to reason over text, making it possible to automate the transformation of procedural text into structured diagrams. However, most existing tools rely on summarization or rule-based extraction, which often result in incomplete or logically inconsistent outputs [9, 6, 16]. In addition, these systems are typically limited to solving easy or well-structured cases, lack interactivity, fail to capture the full depth of complex procedural documents, and do not support clarification features that help users understand or refine the generated flowcharts.

In this paper, we apply AI-driven procedures and develop a prototype system—**IUFlowGen**—that supports full flowchart creation with interactivity and

clarification features. The system combines retrieval-augmented generation (RAG) [7], prompt-based LLM reasoning, and graph-based modeling [5] to help users efficiently understand procedural documents. By leveraging AI assistance, users can quickly extract and comprehend all relevant content from complex texts.

To complement the generation system, we also propose a checklist-based complexity metric that estimates the inherent difficulty of a procedural document. This metric enables systems to assess whether AI assistance is warranted, helping allocate resources more efficiently and ensuring that automation is applied when it provides the greatest benefit.

Contributions. This paper makes the following contributions:

- We propose a complexity metric to assess how difficult a procedural document is for humans to keep track of.
- We introduce a method that combines retrieval-augmented generation, prompt-based reasoning, and graph-based modeling to transform procedural documents into structured flowcharts with clarification features.
- We implement this method in **IUFlowGen**, a prototype system that supports interactive querying, full flowchart generation, and adaptive AI support based on document complexity.

Next, we review related work, introduce IUFlowGen's architecture, describe our evaluation design, present experimental results, and conclude with a discussion of findings.

2 Related Works

Recent advances in large language models (LLMs) and transformer-based architectures, such as GPT-3.5 [11], have enabled automatic extraction of procedural logic and diagram generation from natural language. While these models greatly reduce the manual effort required to interpret and structure procedural documents, there remains a critical gap: there is currently no standardized metric to assess how complex a document is, making it difficult to decide when and how AI assistance should be applied efficiently.

Several authors [4, 14, 6, 9, 16] have explored the use of AI to convert natural language text into flowchart-like visualizations. Hu et al. [4] proposed a sequence-to-graph model with reinforcement learning to generate mind maps from short news articles, though limited in domain and applicable only to short inputs. Text2Chart [14] generated statistical charts from analytical narratives but did not support procedural logic or interactivity. Kulkarni et al. [6] combined OCR, summarization via T5 [13], and graph construction, but deviated from formal flowchart conventions and relied on text abstraction. Mhatre et al. [9] used GPT-3.5 [11] to produce flowcharts from simple prompts, yet only handled basic procedures and ignored nested or multi-actor logic. DiagramAgent [16] introduced a multi-agent prompting framework for general diagram generation, but lacked procedural specialization and did not support human-in-the-loop confirmation.

These approaches demonstrate the potential of AI to overcome the limitations of manual reading and interpretation. However, they fall short in supporting document-scale complexity, user interactivity, and clarification. Among them, the closest to our work are [9, 6, 16], though none provide a comprehensive solution integrating procedural fidelity, interactive refinement, and AI-guided understanding.

Several papers have identified a wide range of factors that contribute to procedural document complexity. These can be grouped into four main categories. *Cognitive Complexity* arises from factors such as document length, domain-specific knowledge requirements, role/entity multiplicity, and the need to track entity changes across steps [2, 1]. *Structural Complexity* includes step compounding, complex sequencing (e.g., branches, loops), and inadequate handling of edge cases or alternate paths [12, 1, 18]. *Linguistic/Textual Complexity* is introduced by implicit logical markers and high lexical density, which increase cognitive burden [8, 17]. Finally, *Visual/Design Factors* such as poor chunking or missing layout structure, make it harder for readers to follow procedural logic [12].

While these works offer partial insight into why some documents are harder to understand, there is currently no standardized metric that consolidates these dimensions into a unified complexity score—nor a formal definition of procedural hardship that could guide decisions on when AI assistance is necessary.

In summary, prior research highlights two key gaps. First, although many studies identify individual factors that contribute to procedural difficulty, there is no unified metric to quantify document complexity or to guide when AI assistance should be applied. Second, while several systems attempt to convert natural language into flowcharts, they either lack support for complex procedural logic, interactivity, or clarification features. These limitations motivate our work: we propose a system that not only generates interpretable flowcharts from procedural documents but also introduces a structured metric to assess document complexity and determine when AI assistance is most beneficial.

3 Document Complexity

3.1 Factors Contributing to Procedural Complexity

There are many factors that contribute to the difficulty of reading and interpreting procedural documents. Based on prior literature and analysis, we categorize these factors into four broad types: *Cognitive Complexity*, *Structural Complexity*, *Linguistic/Textual Complexity*, and *Visual/Design Factors*. Each category captures a different aspect of what makes a procedural document hard to understand—ranging from working memory load and control flow intricacy to textual ambiguity and poor formatting.

Table 1 summarizes ten representative criteria that commonly lead to procedural hardship.

These factors highlight the multidimensional nature of procedural complexity. Some relate to surface structure and formatting, while others involve deeper

Table 1. Representative criteria contributing to procedural document complexity.

Criterion	Description	Ref.
Step compound	A single step contains multiple sub-operations, increasing difficulty of execution and mental parsing.	Pal et al. [12]
Complex sequencing	Includes branching logic (e.g., merge, if-else), loops, or conditionals that disrupt linear flow.	Amini et al. [1]
Length	Longer documents increase cognitive load and reduce recall by requiring sustained attention over many steps.	Garbacea et al. [2]
Implicit markers	Missing connectors like "then," "if," or "next" forces readers to infer step relationships.	Ma et al. [8]
Bad visual layout	Poor chunking or formatting (e.g., no bullet points, headings) makes it harder to follow structure.	Pal et al. [12]
Domain knowledge required	Jargon and specialized terminology assume familiarity with the domain.	Garbacea et al. [2]
Role/entity multiplicity	Multiple actors or roles involved in overlapping tasks increases coordination burden.	Amini et al. [1]
Stage tracking & entity-change	Requires monitoring how entities change across steps—who does what, and when.	Amini et al. [1]
Lexical density	Dense concentration of technical terms makes text harder to skim and comprehend.	Wold et al. [17]
Error & edge-case handling	Missing instructions for failures, retries, or exceptions causes ambiguity.	Zhang et al. [18]

reasoning and domain knowledge. In the next section, we build upon these criteria to propose a unified measurement framework that can assess the complexity level of a document and support decisions about whether AI assistance is needed.

3.2 Proposed Metric: Evaluating Procedural Hardship

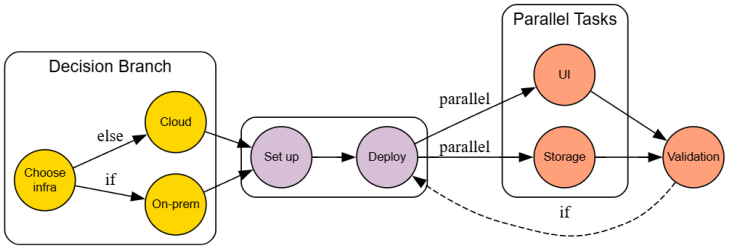

Fig. 1. An illustration of procedural complexity in a document.

Procedural documents often contain numerous actions, diverse actors, domain-specific terminology, and intricate interdependencies between sequential and parallel steps. Fig. 1 illustrates such structural complexity through a dense example involving nested steps, role transitions, and ambiguous control flows. This example exhibits several of the high-impact complexity factors outlined in Table 1, such as *Length*, *Complex sequencing*, and *Error or edge-case handling*.

While prior studies have outlined individual factors that contribute to procedural difficulty, there remains no unified system for quantifying how challenging a document is to comprehend—particularly under traditional, non-AI reading conditions. To address this gap, we introduce a checklist-based metric that evaluates a document's *procedural hardship* based on the presence of ten well-established complexity criteria.

Each criterion is assessed in binary form: it is either present in the document or not. This produces a cumulative hardship score ranging from 0 to 10. Based on internal pilot testing, we observed that users frequently began to experience confusion or hesitation when **three or more criteria** were triggered. This aligns with cognitive theories on overload from simultaneous processing demands, such as information density and branching logic.

Based on this, we define a score-to-level mapping that helps determine when AI assistance may be warranted, as shown in Table 2:

This metric offers more than just a conceptual framework—it provides a lightweight, interpretable method for operationalizing procedural complexity in real-world applications. On the system side, it enables dynamic decisions about when to invoke AI assistance, helping optimize computational resources and user support. On the user side, it sets clear expectations about cognitive load, guiding readers toward appropriate tools or visual aids. By bridging the gap between

Table 2. Hardship score interpretation based on checklist count.

Score Range	Complexity Level	Interpretation
0–2	**Easy**	Linear, short documents with minimal jargon and clear structure.
3–5	**Medium**	Moderately complex: some branching, abstraction, or lexical density.
6–10	**Hard**	Highly complex: long, jargon-heavy, non-linear, vague, or poorly structured.

document features and reader difficulty, the metric establishes a foundation for adaptive, human-centered document processing systems.

4 Proposed Method

This section outlines the system**IUFlowGen**, an AI-powered, human-in-the-loop system for converting unstructured procedural documents into structured, interactive flowcharts. Designed for domains such as technical documentation and compliance, IUFlowGen enhances interpretability by translating complex logic into modular diagrams. The system involves two core actors: the **user**, who submits and manually validates the document, and the **AI engine**, which extracts procedural logic and generates diagrams. Users may include domain experts, technical writers, or educators, who review, query, and refine the AI-generated flowcharts for accuracy and clarity.

The AI engine performs step identification, semantic enrichment, and flowchart rendering based on the Graphviz DOT format. Each procedural step is modeled as a DOT subgraph containing nodes for actors, actions, and entities, with internal logic represented via directed edges. Inter-step relationships are then inferred to connect these subgraphs into a complete flowchart.

These capabilities not only reduce the cognitive burden on users but also improve procedural transparency and downstream task alignment. Moreover, IUFlowGen is intentionally designed to mitigate key sources of procedural complexity as defined in Section 3.1. Each system module corresponds to one or more high-impact hardship factors—such as length, implicit logic, or role multiplicity—identified by our proposed metric. This alignment ensures that the architecture is not only functionally comprehensive but also grounded in an empirical understanding of what makes procedural documents difficult to interpret. The system architecture, illustrated in Fig. 2 , is designed to support the following core functionalities that enable intuitive visual comprehension:

R1. Extract procedural steps and represent them as flowchart nodes.
R2. Identify actors, entities, and attributes for each step.
R3. Model intra-step logic as directed edges.
R4. Identify the inter-step relationships.
R5. Render the procedural flow into a clean, navigable diagram.

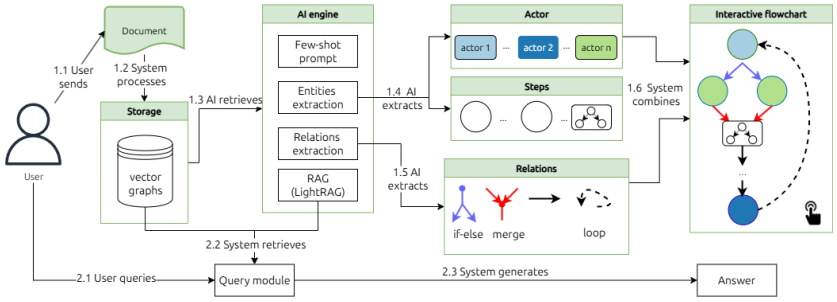

Fig. 2. IUFlowGen's system architecture.

In addition to improving readability, the system also supports interactive querying to enhance user clarification:

R6. Enable clarification querying for manual correction.

The next section details the underlying architecture, key algorithms, and implementation of each module.

4.1 Identifying Procedural Steps

Initially, **IUFlowGen** transforms the uploaded procedural document into vector and graph representations using LightRAG [3], enabling semantically grounded retrieval and reasoning. The document is first segmented into chunks, each encoded into a high-dimensional vector and connected via a graph that captures semantic proximity and contextual links, as shown in step 1.2 in Fig 2.

This dual representation supports accurate step identification: the vector index retrieves relevant segments, while the graph ensures coherence across the document. By leveraging semantic retrieval and graph traversal, the system can detect procedural content even when it is embedded in lengthy or densely worded sentences. This allows the AI model to focus on the underlying procedural intent, regardless of surface-level verbosity or lexical complexity—addressing key difficulties such as *Length* and *Lexical Density* noted in the proposed complexity metric (Section 3.1).

LightRAG ranks and selects segments likely to represent procedural actions using dual retrieval, and a post-processing module standardizes each step for modularity. The result is an ordered list of procedural steps, which provides a clear and interpretable structure for downstream flowchart generation.

Each node represents a distinct procedural step, labeled for clarity and bounded visually. The output is a list of well-defined steps, each represented as a subgraph in Graphviz DOT, forming the foundation for semantic enrichment and structural assembly in later stages, as illustrated in step 1.4 of Fig 2.

4.2 Enriching Procedural Steps with Entities

After identifying core procedural steps, **IUFlowGen** enriches each with semantic details—actors, actions, entities, and contextual information—to satisfy **Requirement R2**. Each step is passed to the *Entity Extractor (EE)* module, which uses LightRAG-based retrieval and chain-of-thought prompting to extract granular elements describing *who does what*, *to what*, and *under what conditions*. These extracted elements are then modeled as structured subgraphs in Graphviz DOT syntax, offering a modular, interpretable representation.

This semantic structuring not only supports visualization but also plays a critical role in maintaining clarity when procedural steps involve multiple participants, interacting components, or evolving states across stages. By explicitly decomposing each step into actors, actions, and associated entities, the system helps disambiguate overlapping responsibilities and surface the dynamic relationships among components over time—especially in domains where roles shift or entities transform across stages.

The construction of each semantic subgraph follows a consistent schema. First, for each procedural step, a subgraph labeled by the step name is created using the format `subgraph cluster_`X, where X represents the step index. Within each subgraph, several types of nodes are added: an `actor_`X node (typically representing the actor, with the default label `"You"`), an `action_`X node corresponding to the main verb of the step, one or more `entity_`Y nodes representing tools or objects involved, and optionally an `info_`Z node that encodes additional contextual details such as conditions or parameters. To maintain consistency and ensure clarity across the entire graph, all node identifiers are uniquely indexed based on their step position, for example, `actor_1` or `action_1`.

Semantic Components: Each procedural step is decomposed into four key semantic elements. The **Actor** represents the agent responsible for performing the action; if not explicitly stated, it defaults to "You". The **Action** is the main verb or operation defining the step's intent. The **Entity** includes tools, data, or objects involved in the step, and multiple entities may be associated with a single action. **Relevant Info** captures optional parameters or conditions that further specify how the action is executed. The result of this extraction is a structured dictionary that maps each step to its associated semantic components, which forms the semantic backbone for modeling logical relationships in the next phase.

4.3 Modeling Intra-Step Logic

With enriched semantic representations in place, IUFlowGen proceeds to identify and encode internal logical relationships within each procedural step to fulfill **Requirement R3**. This involves capturing how actors, actions, entities, and relevant conditions interact within a single step through directed edges in the subgraph. These intra-step dependencies often reflect nuanced operations that are not always expressed linearly in the original text.

To model such internal structure, IUFlowGen leverages syntactic cues, semantic roles, and discourse patterns to infer the flow of control or data among the elements of a step. For example, when a step implicitly involves multiple operations or condition-dependent behaviors (e.g., "Attach the device and configure it if the LED turns green"), the system separates these into semantically distinct but structurally linked nodes, connected via conditional or sequential edges.

This internal subgraph construction ensures that each procedural step—however compressed in phrasing—preserves its full operational meaning when visualized. It accommodates multi-part instructions and latent conditionals by explicitly representing intra-step logic that would otherwise remain buried in dense or compound sentence structures. As a result, users can grasp complex instructions at a glance without needing to mentally unpack the syntax.

Each completed subgraph thus serves not only as a semantic unit but also as a logical capsule, ready for integration into the broader inter-step flowchart assembly process described next.

4.4 Modeling Inter-Step Relationships

To construct a coherent procedural flow, IUFlowGen establishes inter-step relationships that link individual subgraphs into a unified, interpretable sequence. This satisfies **Requirement R4**: Identify inter-step relationships.

The *Inter-Step Relator (ISR)* module integrates multiple signal sources to infer logical transitions between steps: semantic similarity derived from embedding proximity, intra-step graph traversal patterns, and procedural cues retrieved via LightRAG. These sources jointly enable the system to detect a range of transitions, such as:

- **Sequential dependencies**: e.g., Step A → Step B,
- **Conditional branches**: e.g., Step A → Step B *if a condition holds*,
- **Concurrent flows**: for steps that can operate in parallel.

Many procedural documents omit explicit transition markers like "next," "after that," or "if so," requiring inference from context and semantics. IUFlowGen's design compensates for such *implicit markers* by leveraging latent signal patterns that extend beyond surface syntax. In addition, when instructions include vague references to exceptions or error handling (e.g., "retry if failed," "unless otherwise specified"), the module captures these contingencies as conditional or alternate branches within the flow—addressing challenges related to *complex sequencing* and *error and edge-case handling*.

To guarantee syntactic and visual consistency, the inferred edge set undergoes regex-based post-processing. This step corrects malformed DOT syntax, normalizes edge formats, and enforces graph integrity, as illustrated in Step 1.5 of Fig. 2.

The output is a connected set of edges linking modular step subgraphs into a single, structured flowchart. This transformation—from semantically segmented

fragments to an ordered, branching diagram—enables downstream visualization and user comprehension of even highly non-linear or exception-prone procedures.

4.5 Assembling the Final Flowchart

In the final stage, **IUFlowGen** integrates all previously generated components into a complete, syntactically valid DOT representation, satisfying all preceding requirements and fulfilling **Requirement R5**: Render the procedural flow into a clean, navigable diagram.

The *Flowchart Assembler (FA)* module merges intra-step subgraphs and inter-step edges, adding necessary global directives such as `digraph G {}`, `compound=true` (to permit cross-cluster edges), and `rankdir=TB` (to maintain top-to-bottom orientation). Before merging, inter-step connections are rewritten to point to representative actor nodes rather than subgraph clusters directly, preserving semantic intent while maintaining DOT validity via `ltail` and `lhead` annotations.

This stage also implicitly addresses issues of *bad visual layout*, a known contributor to procedural hardship, by enforcing structure-preserving visual alignment and standardized node spacing. The assembler ensures that even highly complex diagrams remain interpretable—avoiding clutter, minimizing edge crossings, and preserving narrative flow.

The result is a logically coherent and visually structured DOT flowchart, suitable for direct rendering, refinement, or downstream integration in interactive tools.

4.6 Clarification Query Module

To enhance interpretability and support user understanding in complex scenarios, **IUFlowGen** integrates a *Clarification Query Module* that satisfies **Requirement R6**: Enable clarification querying for manual correction.

After assembling the flowchart, users can select any node or edge to initiate a query about its semantics, context, or logical connections. When triggered, the system performs a local LightRAG retrieval on the vector-indexed procedural graph, followed by chain-of-thought prompting to generate concise, contextually grounded explanations or refinements. This module acts as a bridge between the system's automated reasoning and the user's domain expertise.

Such interactivity is particularly valuable when the source document contains dense terminology or assumes specialized knowledge—two known complexity factors: *Domain Knowledge Required* and *Lexical Density*. By allowing users to clarify ambiguous or jargon-heavy steps on demand, IUFlowGen reduces misunderstanding and promotes trust in the flowchart's fidelity.

The clarification mechanism thus serves as a safeguard against misinterpretation, offering human-in-the-loop correction for edge cases that fall outside the model's confident reasoning scope. It transforms the static output into an explorable artifact that accommodates diverse user needs and domain expertise.

5 Experiment setup and results

5.1 Experiment Setup

To evaluate the effectiveness of IUFlowGen in aiding procedural document understanding, we conducted a user study with participants divided into two groups: one group used AI assistance via IUFlowGen, while the other completed the tasks without any tool support. Each participant was asked to read and comprehend four procedural documents of increasing complexity, reflecting real-world variation in structure, length, domain-specific jargon, and control flow intricacy.

The documents were arranged from easy to difficult, with complexity increasing in terms of step count, lexical density, and structural nonlinearity. For each document, participants were required to answer a set of predefined comprehension questions that focused on difficult or ambiguous parts of the procedure.

To simulate practical constraints, each reading session was time-limited: 10 minutes for the first two documents and 15 minutes for the last two. Participants in the IUFlowGen group were allowed to interact with the system-generated flowcharts, while the control group relied solely on the raw document text.

Performance was evaluated based on two key metrics: (1) the number of correct answers to the comprehension questions, and (2) the total time taken to respond. This setup allows for both quantitative and qualitative comparisons between traditional reading and AI-assisted understanding under time pressure.

5.2 Experiment Results

Table 3 summarizes the performance of participants across the two experimental groups. The results support our initial hypothesis: participants who used IUFlowGen consistently outperformed those without AI assistance in both comprehension accuracy and response time, particularly as document complexity increased. While both groups performed comparably on simpler documents, the AI-assisted group demonstrated a clear advantage on complex tasks, suggesting that flowchart-based visualization and interactive clarification significantly enhance procedural understanding under time constraints.

Table 3. Comparison of performance between participants with and without AI assistance.

Group	**Average Correct Answers**	**Average Time (minutes)**
With AI Assistance	3.6 / 4.0	10.2
Without AI Assistance	2.7 / 4.0	12.8

Remarks. The results in Table 3 demonstrate a clear benefit from AI assistance. Participants using IUFlowGen not only achieved higher accuracy in answering comprehension questions, but also completed their tasks in less time on average. This performance gap became more pronounced as document complexity

increased, suggesting that AI-supported visualization plays a particularly important role in helping users navigate branching logic, implicit relationships, and dense technical language. These findings support the claim that structured flowcharts and clarification features substantially reduce cognitive load during procedural reading.

6 Discussion

IUFlowGen offers a novel approach to procedural document understanding by combining retrieval-augmented generation (RAG) with graph-based flowchart synthesis. This integration enables the system to convert unstructured procedural text into accurate, interactive diagrams that capture both semantic content and logical structure. Unlike prior systems, IUFlowGen emphasizes human-in-the-loop assistance, modular reasoning, and structured interpretability.

Compared to prior systems, IUFlowGen offers stronger support for procedural reasoning and interactivity. Existing approaches, such as Kulkarni et al. [6], Mhatre et al. [9], and DiagramAgent [16], rely on OCR, summarization, or multi-agent prompting, but often lose procedural detail, lack multi-actor support, or omit interactive refinement. In contrast, IUFlowGen builds intermediate knowledge graphs via LightRAG, extracts procedural steps and relations, and assembles modular DOT subgraphs into coherent flowcharts—without requiring pre-annotated data.

Critically, this paper not only presents a system for flowchart generation, but also introduces a checklist-based metric for assessing the inherent complexity of procedural documents. This metric helps determine when AI assistance is most beneficial and provides a foundation for benchmarking document difficulty across domains.

Together, IUFlowGen and the proposed complexity metric represent a unified approach for translating complex procedural documents into structured flowcharts—offering an efficient, interpretable solution for aiding user understanding in both technical and real-world applications.

7 Conclusion

We have introduced IUFlowGen, a new approach to converting procedural documents into structured flowcharts using AI techniques, while preserving logical clarity and improving interpretability. Our system empowers users to explore, query, and refine procedural logic extracted from complex texts through a human-in-the-loop interface. By supporting local execution and modular components, IUFlowGen offers enhanced privacy and adaptability for sensitive or domain-specific use cases.

Moreover, in the broader context of digital transformation and process automation, our approach could be applied to government workflows, compliance documentation, or internal policy modeling. For example, regulatory guidelines or operational handbooks could be automatically visualized to support auditing,

training, or system integration. To extend usability at scale, future work may investigate more robust prompt control, multilingual support, and the integration of domain-specific validation rules.

References

1. Amini, A., Bosselut, A., Mishra, B.D., Choi, Y., Hajishirzi, H.: Procedural reading comprehension with attribute-aware context flow. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP). p. 122–136. Association for Computational Linguistics (2020), https://arxiv.org/abs/2003.13878
2. Garbacea, C., Bowman, S.R.: Explainable prediction of text complexity. arXiv preprint arXiv:2007.15823 (2020), https://arxiv.org/abs/2007.15823
3. Guo, Z., et al.: Lightrag: Simple and fast retrieval-augmented generation. arXiv preprint arXiv:2410.05779 (2024). https://doi.org/10.48550/arXiv.2410.05779, https://arxiv.org/abs/2410.05779, version 3, revised April 28, 2025
4. Hu, M., et al.: Efficient mind-map generation via sequence-to-graph and reinforced graph refinement. In: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP) (2021). https://doi.org/10.48550/arXiv.2109.02457, https://arxiv.org/abs/2109.02457
5. Jin, B., et al.: Large language models on graphs: A comprehensive survey. arXiv preprint arXiv:2312.02783 (2024). https://doi.org/10.48550/arXiv.2312.02783, https://arxiv.org/abs/2312.02783, to appear in Transactions on Knowledge and Data Engineering (TKDE)
6. Kulkarni, A., Shah, H., D'Mello, L., Shah, K.: Flowchart generation and mind map creation using extracted summarized text. In: Proceedings of the 2023 International Conference on Recent Advances in Science and Engineering Technology (ICRASET). IEEE, B G NAGARA, India (November 2023). https://doi.org/10.1109/ICRASET59632.2023.10420315
7. Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.t., Rocktäschel, T., Riedel, S., Kiela, D.: Retrieval-augmented generation for knowledge-intensive nlp tasks. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL). p. 945-957 (2020). https://doi.org/10.18653/v1/2020.acl-main.85
8. Ma, K., Lin, X., Zhang, Y., Wang, J.J., Liu, Q., Hajishirzi, H.: Coalescing global and local information for procedural text understanding. arXiv preprint arXiv:2208.12848 (2022), https://arxiv.org/abs/2208.12848
9. Mhatre, M., Pandey, A., Rane, H., Sahu, S.: A novel approach for creating flowcharts using generative ai. In: Proceedings of the 2024 Asia Pacific Conference on Innovation in Technology (APCIT). IEEE (July 2024). https://doi.org/10.1109/APCIT62007.2024.10673464
10. Musdizal: The influence of visualization strategy on reading comprehension ability. Jurnal Dimensi **8**(2) (June 2019). https://doi.org/10.33373/dms.v8i2.2162
11. OpenAI: Gpt-3.5 overview. https://platform.openai.com/docs/models/gpt-3-5 (2023), accessed July 6, 2025
12. Pal, K.K., Mishra, S., Alavi, S.M.K., Sharma, A., Baral, C.: Constructing flow graphs from procedural cybersecurity texts. In: Findings of the Association for Computational Linguistics: EMNLP 2021. pp. 2844–2856. Association for Computational Linguistics (2021), https://arxiv.org/abs/2105.14357

13. Raffel, C., et al.: Exploring the limits of transfer learning with a unified text-to-text transformer. arXiv preprint arXiv:1910.10683 (2020). https://doi.org/10.48550/arXiv.1910.10683, https://arxiv.org/abs/1910.10683
14. Rashid, M.M., et al.: Text2chart: A multi-staged chart generator from natural language text. arXiv preprint arXiv:2104.04584 (2021). https://doi.org/10.48550/arXiv.2104.04584, https://arxiv.org/abs/2104.04584
15. Saba, W.S.: Llms' understanding of natural language revealed. arXiv preprint arXiv:2407.19630 (2024). https://doi.org/10.48550/arXiv.2407.19630, https://arxiv.org/abs/2407.19630, version 2, revised August 2, 2024
16. Wei, J., et al.: From words to structured visuals: A benchmark and framework for text-to-diagram generation and editing. arXiv preprint arXiv:2411.11916 (November 2024). https://doi.org/10.48550/arXiv.2411.11916, https://arxiv.org/abs/2411.11916
17. Wold, S., Nygaard, V., Velldal, E., Hohle, P.: Estimating lexical complexity from document-level distributions. arXiv preprint arXiv:2404.01196 (2024), https://arxiv.org/abs/2404.01196
18. Zhang, Z., Zhao, H., Cai, D., Yang, Y., Liu, Q., Zhang, S.: Knowledge-aware procedural text understanding with multi-stage training. arXiv preprint arXiv:2009.13199 (2020), https://arxiv.org/abs/2009.13199