Systems&Control Transactions

Research Article - Peer Reviewed Conference Proceeding
ESCAPE 35 - European Symposium on Computer Aided Process Engineering
Ghent, Belgium. 6-9 July 2025
Jan F.M. Van Impe, Grégoire Léonard, Satyajeet S. Bhonsale, Monika E. Polańska, Filip Logist (Eds.)

PSE PRESS

GRAPSE: Graph-Based Retrieval Augmentation for Process Systems Engineering

Daniel Ovalle[a†], Arpan Seth[b†], John R. Kitchin[a], Carl D. Laird[a], and Ignacio E. Grossmann[a*]

[a] Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, PA 15213, United States
[b] Evonik Corporation, Process Technology and Engineering, Trexlertown, PA 18087, United States
* Corresponding Author: grossmann@cmu.edu.
† These authors contributed equally to this work.

ABSTRACT

Large Language Models have demonstrated potential in accelerating scientific discovery, but they face challenges when making inferences in rapidly evolving and niche domains like Process Systems Engineering (PSE). To address this, we propose a Graph-based Retrieval-Augmented Generation (RAG) pipeline specifically designed for PSE papers. Our pipeline includes custom document parsing, knowledge graph construction, and refinement to enhance retrieval accuracy. We evaluate the effectiveness of our approach using an automatically generated benchmark consisting entirely of PSE-related questions. The results show that our pipeline outperforms both non-RAG and vanilla RAG implementations in terms of relevant document retrieval and overall answer quality. Additionally, our implementation is fully customizable, allowing users to select the papers most relevant to their specific tasks. This framework is openly available, providing a flexible solution for those working in PSE or similar domains.

Keywords: Large Language Model, Retrieval-Augmented Generation, Graph-based Retrieval, Process Systems Engineering

INTRODUCTION

Large Language Models (LLMs) have shown remarkable capabilities, performing well across a wide range of tasks, including accelerating scientific discovery in different fields. More specifically, in the field of Process Systems Engineering (PSE), LLMs have been adapted to various tasks, including flowsheet generation, the prediction of chemical properties, and infeasibility diagnosis in optimization models [1-2]. Still, off-the-shelf LLMs are inherently limited to the information present in their training data, which may become outdated or fail to cover specialized topics comprehensively [3]. This limitation is particularly relevant in both rapidly evolving and highly specialized fields, like PSE [4].

Retrieval-Augmented Generation (RAG) is an advanced framework that enhances the capabilities of LLMs by integrating them with external knowledge retrieval systems. Unlike standard LLMs, which rely solely on pre-trained knowledge, RAG combines parametric memory (the information encoded in the model's weights) with non-parametric memory (external data sources, such as databases or knowledge repositories). A retriever dynamically identifies and retrieves relevant information from these external sources, presenting it to the model alongside user inputs [5]. This approach enables RAG to provide more accurate, factual, and context-specific responses, while allowing the knowledge base to be updated without requiring model retraining. Such dynamic and interpretable knowledge integration makes RAG particularly valuable for addressing complex, knowledge-intensive tasks. Therefore, the PSE community can benefit from a domain-specific RAG to improve standard LLM performance, while ensuring that the responses remain grounded in truthful and updated information.

Graph-based knowledge structures are powerful tools to study complex and interconnected information, such as the one in scientific domains. Graph RAG enhances traditional RAG frameworks by leveraging knowledge graphs to represent and retrieve interconnected information more effectively. Unlike standard indexed databases that overlook relationships across documents, Graph RAG can be used to construct knowledge graphs from academic papers, linking entities, concepts, and their relationships. This approach captures the interconnected nature of scientific knowledge, enabling more accurate and context-aware retrieval [6-7]. For example, Graph RAGs have successfully been used in medicine to improve retrieval accuracy and response quality by modeling relationships among diseases, treatments, and outcomes [8].

In this work, we propose a Graph RAG specifically designed to incorporate domain-specific knowledge from PSE. To achieve this, we develop a pipeline that parses academic papers and constructs tailored knowledge graphs from PSE literature. Additionally, we create a benchmark dataset of PSE-specific questions and answers derived from the utilized corpus to evaluate the accuracy and quality of the retrieved information. While copyright restrictions prevent us from publishing graph databases from the model, the pipeline developed in this work is freely accessible through a GitHub repository. This pipeline allows users to fully replicate and customize their selection of papers, enabling the construction of a Graph-based Retrieval Agent for Process Systems Engineering (GRAPSE) that is precisely aligned with their desired body of literature.

Standard Retrieval-Augmented Generation

RAG is a framework that enables LLMs to infer knowledge from documents that were not directly trained on. This approach leverages the strength of LLMs in abstracting information from text when the relevant content is provided as part of the prompt, often referred to as the context. The primary objective of RAG is to retrieve the most relevant document(s) from a given set based on a user query, ensuring that the retrieved information is passed as context. This enriched prompt, containing both the query and the related context, allows the LLM to perform accurate and informed inference, effectively answering the query [5].

They key part of the RAG framework are embeddings, which are are numeric vector representations of text that encode semantic and contextual information. Words or phrases with similar meanings are positioned closer together in this high-dimensional space, where the distance and angle between vectors represent the degree of similarity. Transformer-based models, unlike traditional embedding methods like TF-DF [9], create contextual embeddings, meaning that the vector representation of a word adapts based on its surrounding context, capturing complex relationships in language.

Once documents have been split, chunks of text are passed through a transformer embedding model, which processes the input and produces high-dimensional vectors that reflect the semantic and contextual meaning of the text. These embeddings are then stored in a vector database, which indexes both the original text and its corresponding vectors. This setup allows for efficient storage and fast retrieval, as the database can perform similarity searches by comparing query embeddings to stored vectors. Typically, a cosine similarity metric is used to determine alignment between vectors.

The chunks corresponding to the vectors with the highest similarity to the embedded query vector are retrieved from the database and provided to the LLM as contextual input. A sketch of a standard RAG methodology is presented in Figure 1.

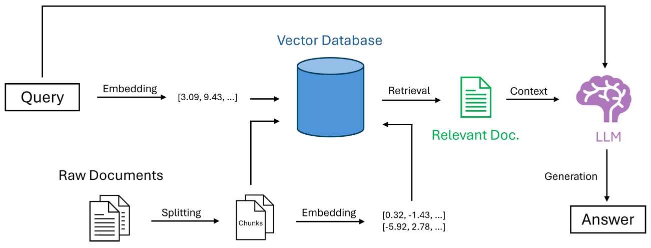


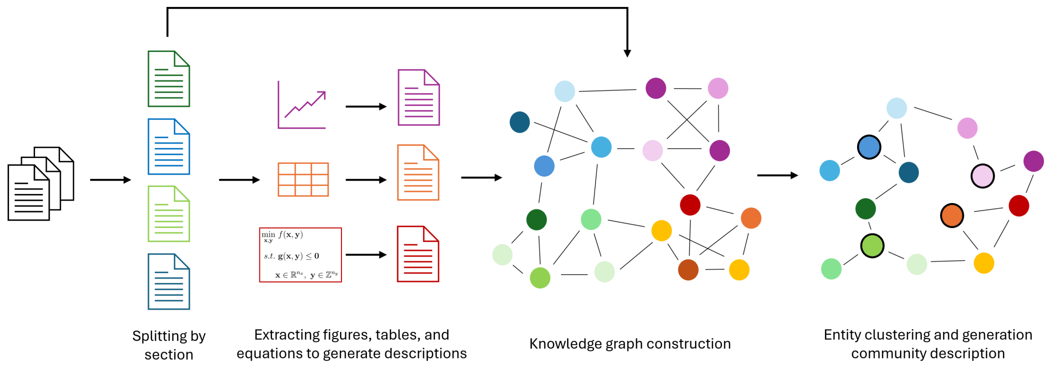
Figure 1. Standard Retrieval Augmented Generation pipeline. Raw documents are divided into chunks based on a predefined criterion, often a fixed character length. Each chunk is then converted into a high-dimensional vector using a transformer-based embedding model and indexed within a vector database. Similarly, the query is embedded using the same model to produce its vector representation. A similarity search is performed between the query vector and the vectors stored in the database. The chunks corresponding to the top k most similar vectors are retrieved and incorporated into the prompt as context. This enriched prompt, combining the original query with the retrieved context, is then passed to an LLM for final answer generation.

METHODOLOGY

The following subsections outline our proposed Graph RAG methodology, detailing the process of transforming raw papers into a refined knowledge graph. For reference, a representation of the entire pipeline is provided in Figure 2.

Document Parsing

The first step in our approach involves the user selecting the academic papers that will form the knowledge base for the retrieval system. Once the papers are chosen, the pipeline begins by processing each paper, breaking it down into smaller units of information commonly referred to as "chunks". Some traditional RAG methodologies often utilize a fixed character-length chunking strategy, where the chunk size is limited by the maximum input length supported by the embedding model, with overlapping regions between consecutive chunks [5]. While this general approach is applicable to



Splitting by section
Extracting figures, tables, and equations to generate descriptions
Knowledge graph construction
Entity clustering and generation community description

Figure 2: Sketch of the proposed Graph RAG pipeline. Raw papers are divided into sections, and each section is analyzed to extract all figures, tables, and equations. A multimodal LLM generates descriptions for these extracted elements, which are then appended to their corresponding sections. From this enriched set of documents, a knowledge graph is constructed, where nodes represent entities identified in the documents, and edges capture their relationships. In the figure, colors denote the topics found in the documents, illustrating the degree of their relation. Similar entities are merged to eliminate duplicates, and a community detection algorithm is applied to group nodes into clusters [7]. Nodes with black borders represent the descriptions generated based on the information within each community. The final knowledge graph is stored in a graph vector database for efficient search and retrieval.

any document, it does not consider the inherent structure of academic papers. Consequently, important ideas may be fragmented across multiple chunks, potentially reducing the retrieval quality.

To address this limitation, our pipeline parses each paper to identify its structural components, such as sections, subsections, and other logical divisions. Instead of relying on arbitrary fixed-size chunking, we allow each section to form its own chunk. This ensures that each chunk represents a self-contained idea or concept, preserving the coherence and contextual integrity of the information.

Academic papers in PSE often include a significant amount of information formatted as equations, tables, and figures. To effectively integrate this knowledge into the retrieval system, we extract these elements and generate detailed text descriptions for each using a multimodal LLM. These descriptions provide contextual information about the purpose, content, and relevance of the equations, tables, and figures, enhancing their interpretability and utility within the knowledge base.

The generated descriptions are not only linked to the specific chunk where the corresponding equation, table, or figure appears but are also associated with all chunks that reference them. For instance, the description of "Figure 1" is attached to both the chunk where the figure resides and all other chunks that mention "Figure 1." This approach ensures that users querying the system can access enriched and cohesive context related to these important elements, regardless of where they are discussed within the academic paper.

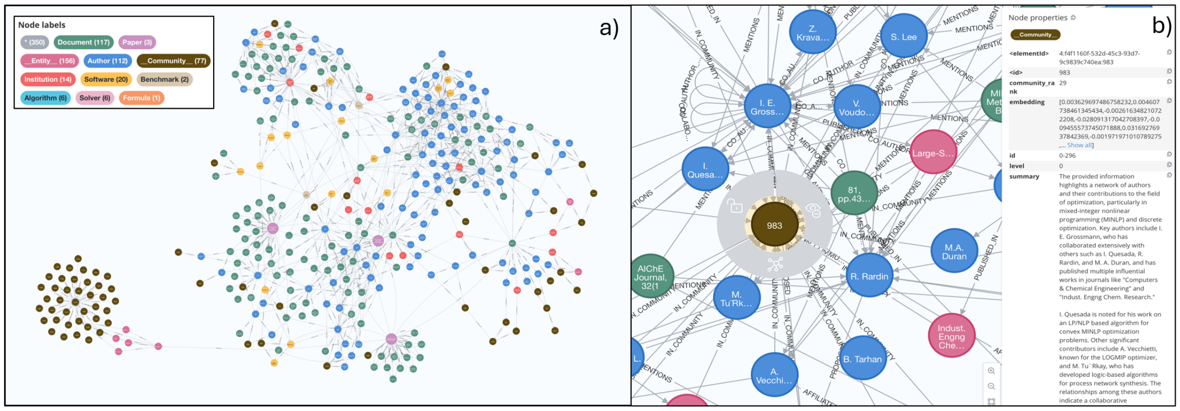
Knowledge Graph Construction

After parsing and processing all the selected papers, the next step of the pipeline is to construct a knowledge graph that integrates and relates the extracted information across the entire body of literature. To achieve this, entities are first identified and extracted from the documents [6]. Tailoring the pipeline to academic literature, entity recognition is constrained to specific categories: papers, solvers, authors, algorithms, institutions, benchmarks, and software. However, this categorization can be customized by users to include additional or alternative categories, allowing greater flexibility and adaptability.

For each identified entity, a detailed description is generated based on the contextual information found in the text. Once these descriptions are established, relationships between entities are inferred and created to form a network [6]. To ensure greater coherence and maintain the integrity of the source information, each chunk from the parsed documents is treated as a separate entity within the graph and linked to the entities extracted from it. The entire knowledge graph construction process is implemented using the Neo4j graph database, enabling efficient storage, querying, and visualization of the interconnected knowledge [10].

Graph Enhancement

To improve the efficiency of the constructed knowledge graph, the pipeline incorporates a step to group similar entities, reducing redundancy and improving query performance. This step ensures that semantically or contextually identical entities, which may be represented with slight variations (e.g., "Ignacio Grossmann"



<elementId>	4:f4f1160f-532d-45c3-93d7-9c9839c740ea:983
<id>	983
community_rank	29
embedding	[0.003629697486758232,0.004607738461345434,-0.002616348210722208,-0.028091317042708397,-0.009455573745071888,0.03169276937842369,-0.001971971010789275,... Show all]
id	0-296
level	0
summary	The provided information highlights a network of authors and their contributions to the field of optimization, particularly in mixed-integer nonlinear programming (MINLP) and discrete optimization. Key authors include I. E. Grossmann, who has collaborated extensively with others such as I. Quesada, R. Rardin, and M. A. Duran, and has published multiple influential works in journals like "Computers & Chemical Engineering" and "Indust. Engng Chem. Research." I. Quesada is noted for his work on an LP/NLP based algorithm for convex MINLP optimization problems. Other significant contributors include A. Vecchietti, known for the LOGMIP optimizer, and M. Tu¨Rkay, who has developed logic-based algorithms for process network synthesis. The relationships among these authors indicate a collaborative

Figure 3: Knowledge graph visualization. Figure a) illustrates the complete knowledge graph built from a large collection of PSE papers, with node colors representing various entity types across the network. Figure b) provides a detailed view of a single community within the graph, showing the relationships among entities in the community and the corresponding community description.

vs. "I.E. Grossmann"), are unified into a single node within the graph.

The grouping process begins with a fuzzy search across the recognized entities to identify potential matches based on similarities in names, descriptions, or other attributes. Once potential matches are identified, a clustering algorithm is applied to group these similar entities systematically [7]. The clustering consolidates entities with high similarity scores while preserving entities that are sufficiently different.

Generating communities within a knowledge graph and creating descriptions of these communities play a crucial role in enabling global inference across the graph as seen in Edge et al. [7]. These community summaries provide a foundation for answering high-level, global questions that go above the scope of individual text chunks. By leveraging the Leiden community detection algorithm, the graph is partitioned into clusters of closely related entities, each meaning to represent a distinct theme or area of knowledge within the corpus.

Once communities are identified, generating comprehensive descriptions of each community enhances the ability of the system to perform global query-focused summarization. Analogue to the approach by Edge et al. [7], we separate queries into two distinct vector searches: one local, targeting raw chunks, and one global, focused on community descriptions. By combining the results of the two searches, we create a single retrieved context that simultaneously accounts for both granular details and high-level abstractions. This approach has proven more effective for answering open-ended questions that require various levels of knowledge abstraction [11]. A visualization of the obtained knowledge graphs is provided in Figure 3.

We employed the OpenAI API framework with *ChatGPT-4o-mini* as the generation model, utilized the *text-embedding-ada-002* model for embedding generation, and leveraged Chroma and Neo4j [10] as vector databases. To make sure GRAPSE is easily accessible, all computations were conducted on MacBook Air M1 (16 GB RAM). For all vector searches across the RAG pipelines, we employed similarity search with a threshold of 0.3. The LLMs were used with default settings, except the temperature was set to 0.0, and the same prompts were used uniformly across all pipelines.

EVALUATION

Evaluating LLM frameworks is a key step in their development, requiring well-defined benchmarks of questions and corresponding metrics. While there are several established benchmarks for LLMs covering topics such as history, reasoning, coding, and math, no such benchmark currently exists for PSE questions. One potential solution is to collect questions from PSE textbooks and tests, but this approach is constrained by copyright issues. Alternatively, manually creating a question set is possible, but it is labor-intensive and would limit users from readily testing their customized pipelines. To address these challenges, we leverage Ragas to automatically generate a benchmark of PSE-specific questions and answers for evaluating retrieval pipeline performance [11]. This method uses the raw documents in the corpus to provide ground truth, enabling reliable assessment of retrieval quality. The metrics employed to evaluate the pipeline are detailed below.

1. Context Recall

Context Recall (CR) evaluates how effectively relevant documents or information are retrieved, ensuring no important details are missed. Higher scores, ranging from 0 to 1, indicate better performance by capturing all pertinent material. This metric simplifies evaluation by breaking the reference answer into individual claims and verifying alignment with the retrieved contexts, eliminating the need for time-intensive direct annotation of reference contexts. CR is calculated using the input, reference answer, and retrieved contexts, emphasizing completeness in retrieval as:

$$CR = \frac{\#\ of\ claims\ in\ reference\ supported\ by\ retrieved\ context}{Total\ \#\ of\ claims\ in\ the\ reference} \quad (1)$$

2. Faithfulness

The Faithfulness (F) metric measures the factual alignment of a response with the retrieved context, with scores ranging from 0 to 1, where higher values indicate greater consistency. A response is considered fully faithful when all its claims are supported by the given context. This metric is calculated by identifying individual claims in the response and verifying their support within the retrieved context, ensuring a precise assessment of factual accuracy as:

$$F = \frac{\#\ of\ claims\ in\ response\ supported\ by\ retrieved\ context}{Total\ \#\ of\ claims\ in\ the\ response} \quad (2)$$

3. Factual Correctness

Factual Correctness (FC) assesses how accurately a generated response aligns with a given reference, assigning a score between 0 and 1, where higher values indicate better factual consistency. The response and reference are broken into individual claims, and their overlap is evaluated using an LLM. The metric is derived as an F1 score, with true positives representing claims in the response that match the reference, false positives as unmatched claims in the response, and false negatives as reference claims missing from the response. Precision and Recall are combined to compute the final F1 score as:

$$FC = 2\,\frac{Precision \cdot Recall}{Precision + Recall} \quad (3)$$

4. Semantic Similarity

Semantic Similarity measures how closely the meaning of a generated response aligns with the ground truth, with scores ranging from 0 to 1, where higher values indicate a stronger semantic match. This metric evaluates response quality by assessing alignment in the semantic embedding space. The ground truth and generated response are transformed into vector representations using the same embedding model, and their similarity is calculated using cosine similarity.

RESULTS

The entire field of PSE cannot be comprehensively represented through a modest selection of papers due to its wide-ranging diversity in research areas [4]. For the purpose of this work, we based our document selection on the references cited in the *Advanced Optimization for Process Systems Engineering* book by Professor Grossmann [12]. This selection includes 48 papers with the full list available in the GitHub repository. While this is not an exhaustive representation of all research areas within PSE, it serves as an illustrative proof of concept for this study. Importantly, our pipeline is designed to be paper-agnostic, allowing users to customize their selection of documents to suit their specific tasks and apply the methodology accordingly. Our resulting benchmark consists of 577 questions.

Metric Comparison

We compared four distinct pipelines. The *azure_openai* pipeline represents a straightforward use of the LLM, equivalent to a single, simple API call. The *langchain_basic* pipeline corresponds to a standard RAG implementation, as illustrated in Figure 1. The *graph_db_vector* pipeline performs vector search within the graph database without any additional enhancements. Lastly, the *grapse* pipeline reflects the complete methodology proposed in this work. The results of this comparison are presented in Figure 4.

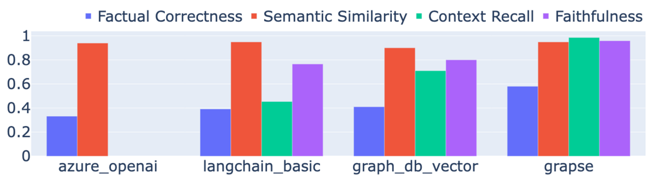
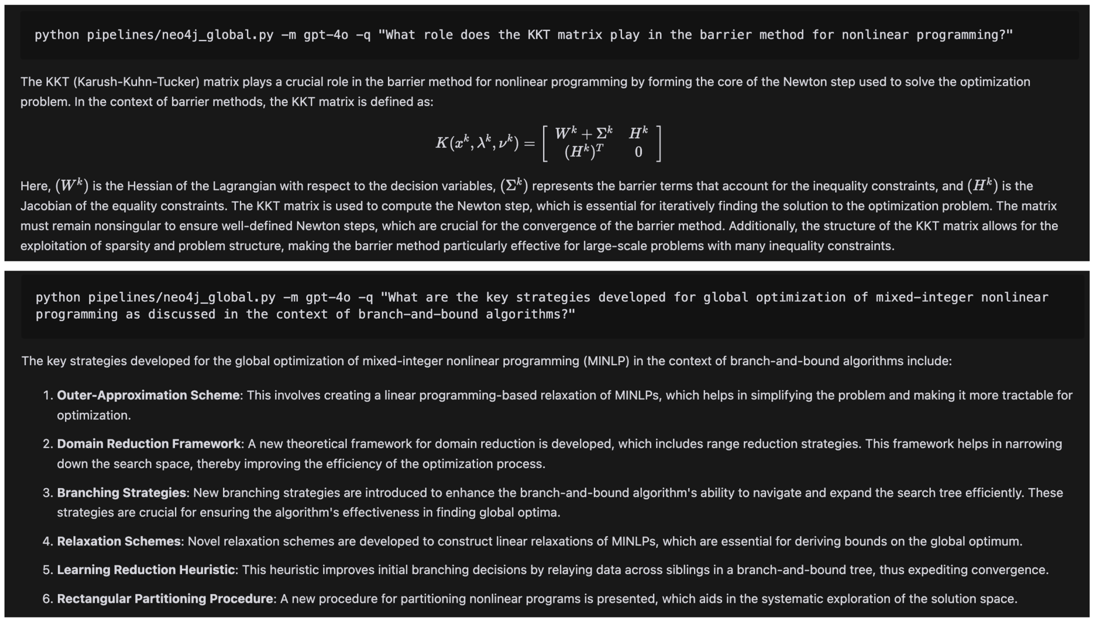


Figure 4. Metric comparison for different LLM pipelines performing in a PSE benchmark. Note a score of 1 is the best in all metrics.

Our proposed pipeline consistently achieves the highest scores across all evaluated metrics. The Factual Correctness and Faithfulness metrics are not applicable to the simple LLM call, as it lacks a retrieval mechanism. The Semantic Similarity metric shows minimal variation across pipelines, as it primarily reflects closeness in the embedded space rather than natural language similarity, which explains the uniformly high similarity between responses and queries. The most critical metric, however, is Context Recall, which evaluates the effectiveness of the retrieval mechanism. This metric shows the most significant improvement in our pipeline, demonstrating that the retrieved documents effectively support the generation of accurate answers.

Q&A

To demonstrate the practical application of our framework, we posed two illustrative questions to our pipeline. The LLM was instructed to provide two distinct types of responses, showcasing the system's flexibility, while maintaining quality and rigor. The questions and corresponding answers are presented in Figure 5.



```
python pipelines/neo4j_global.py -m gpt-4o -q "What role does the KKT matrix play in the barrier method for nonlinear programming?"
```

The KKT (Karush-Kuhn-Tucker) matrix plays a crucial role in the barrier method for nonlinear programming by forming the core of the Newton step used to solve the optimization problem. In the context of barrier methods, the KKT matrix is defined as:

$$K(x^k, \lambda^k, \nu^k) = \begin{bmatrix} W^k + \Sigma^k & H^k \\ (H^k)^T & 0 \end{bmatrix}$$

Here, (W^k) is the Hessian of the Lagrangian with respect to the decision variables, (Σ^k) represents the barrier terms that account for the inequality constraints, and (H^k) is the Jacobian of the equality constraints. The KKT matrix is used to compute the Newton step, which is essential for iteratively finding the solution to the optimization problem. The matrix must remain nonsingular to ensure well-defined Newton steps, which are crucial for the convergence of the barrier method. Additionally, the structure of the KKT matrix allows for the exploitation of sparsity and problem structure, making the barrier method particularly effective for large-scale problems with many inequality constraints.

```
python pipelines/neo4j_global.py -m gpt-4o -q "What are the key strategies developed for global optimization of mixed-integer nonlinear
programming as discussed in the context of branch-and-bound algorithms?"
```

The key strategies developed for the global optimization of mixed-integer nonlinear programming (MINLP) in the context of branch-and-bound algorithms include:

1. **Outer-Approximation Scheme**: This involves creating a linear programming-based relaxation of MINLPs, which helps in simplifying the problem and making it more tractable for optimization.
2. **Domain Reduction Framework**: A new theoretical framework for domain reduction is developed, which includes range reduction strategies. This framework helps in narrowing down the search space, thereby improving the efficiency of the optimization process.
3. **Branching Strategies**: New branching strategies are introduced to enhance the branch-and-bound algorithm's ability to navigate and expand the search tree efficiently. These strategies are crucial for ensuring the algorithm's effectiveness in finding global optima.
4. **Relaxation Schemes**: Novel relaxation schemes are developed to construct linear relaxations of MINLPs, which are essential for deriving bounds on the global optimum.
5. **Learning Reduction Heuristic**: This heuristic improves initial branching decisions by relaying data across siblings in a branch-and-bound tree, thus expediting convergence.
6. **Rectangular Partitioning Procedure**: A new procedure for partitioning nonlinear programs is presented, which aids in the systematic exploration of the solution space.

Figure 5: Responses generated by our proposed pipeline for two illustrative questions. For the first question the LLM was instructed to provide a concise and mathematically focused answer For the second question, it was prompted to provide a broader and more general explanation.

CONCLUSIONS

We proposed a graph-based retrieval-augmented generation pipeline specifically designed for PSE papers. The pipeline is fully replicable and customizable to accommodate any papers the user requires. Additionally, we integrated a benchmark generation tool to facilitate the evaluation of the resulting RAGs, potentially paving the way for the creation of PSE question sets. Our pipeline outperforms both non-RAG and vanilla RAG implementations demonstrating its effectiveness. In this paper, we focused on zero-shot prompting to generate and evaluate answers. However, future work will involve integrating this pipeline with reasoning models to further enhance answer quality.

DIGITAL SUPPLEMENTARY MATERIAL

All the code used in this work can be found in an openly availably GitHub repository: https://github.com/arpanseth/GRAPSE

ACKNOWLEDGEMENTS

We thank the Center for Advanced Process Decision-making (CAPD) and the Department of Chemical Engineering from Carnegie Mellon University for kindly supporting this work.

REFERENCES

1. Schweidtmann AM. Generative artificial intelligence in chemical engineering. Nature Chemical Engineering 1.3:193-193 (2024)
2. Hao C, Constante-Flores GE, Li C. Diagnosing infeasible optimization problems using large language models. INFOR: Information Systems and Operational Research 62.4:573-587 (2024).
3. Petroni F. Language models as knowledge bases?. arXiv preprint arXiv:1909.01066 (2019).
4. Pistikopoulos EN, Barbosa-Povoa A, Lee JH, Misener R, Mitsos A, Reklaitis GV, Venkatasubramanian V, You F, Gani R. Process systems engineering-the generation next?. Computers & Chemical Engineering 147: 107252 (2021).
5. Lewis P, Perez E, Piktus A, Petroni F, Karpukhin V, Goyal N, Küttler H, Lewis M, Yih WT, Rocktäschel T, Riedel S. Retrieval-augmented generation for knowledge-intensive nlp tasks. Advances in Neural Information Processing Systems 33:9459-9474 (2020).
6. Hu Y, Lei Z, Zhang Z, Pan B, Ling C, Zhao L. GRAG: Graph Retrieval-Augmented Generation. arXiv preprint arXiv:2405.16506 (2024).
7. Edge D, Trinh H, Cheng N, Bradley J, Chao A, Mody A, Truitt S, Larson J. From local to global: A graph rag approach to query-focused summarization. arXiv preprint arXiv:2404.16130

(2024).

8. Wen Y, Wang Z, Sun J. Mindmap: knowledge graph prompting sparks graph of thoughts in large language models. arXiv. arXiv preprint arXiv:2308.09729 (2023).
9. Leskovec J, Rajaraman A, Ullman JD. Mining of massive data sets. Cambridge University Press (2020).
10. Miller JJ. Graph database applications and concepts with Neo4j. In Proceedings of the southern association for information systems conference. 2324-36:141-147 (2013).
11. Sarthi P, Abdullah S, Tuli A, Khanna S, Goldie A, Manning CD. Raptor: Recursive abstractive processing for tree-organized retrieval. arXiv preprint arXiv:2401.18059 (2024).
12. Es S, James J, Espinosa-Anke L, Schockaert S. Ragas: Automated evaluation of retrieval augmented generation. arXiv preprint arXiv:2309.15217 (2023).
13. Grossmann IE. Advanced Optimization for Process Systems Engineering. Cambridge University Press (2021).