ContextCraft - Whitepaper for Expansion and Experimentation

1. Project Overview

ContextCraft is a framework for building a robust document processing and query retrieval system. It combines multiple components such as PDF parsing, vector generation, and query engine for content retrieval and response generation. The project is designed to be modular and scalable, with the potential for easy extension to fit various use cases in document processing and AI-driven query systems.

---

2. Experimentation Guidelines

A. Setting Up Your Own Experiments

To extend ContextCraft with new experiments or features, follow these guidelines:

1. **Choosing a New Dataset**:
    - Place your dataset in the `pdfs/` folder and update the pipeline to handle new data formats if necessary. You can create a custom notebook to preprocess your dataset and integrate it into the pipeline.
2. **Model Experimentation**:
    - Experiment with different language models (LLMs) or sentence transformers (SLMs) for vector generation and retrieval. You can use pre-trained models from Hugging Face or other repositories.
    - For instance, test with other transformer models like `distilbert`, `roberta`, or even specialized models such as `deepset/bert-base-cased-squad2` for better performance on document-based queries.
3. **Customization**:
    - Modify the **query engine** or **retrieval function** by adding custom algorithms or experiment with different similarity measures (e.g., cosine, dot product, Euclidean).
    - Experiment with **query structuring** to make the input queries more robust for your specific use case.

---

3. Example Use Cases

Here are some example experiments you can run within the framework:

- **New Document Types**: Add support for new file formats (e.g., DOCX, HTML). Modify the `PDF Parser` to extract text from these file types and adjust the pipeline accordingly.
- **New Vectorization Models**: Test different vectorization models (e.g., using `sentence-transformers` library) to improve the quality of embeddings.
- **Performance Tuning**: Use techniques like **dimensionality reduction** (e.g., PCA, LDA) to reduce the vector size for faster retrieval while maintaining performance.

---

4. Modularizing the Pipelines

To make the project more scalable, consider modularizing each step of the pipeline. The modular pipelines can help in reusing components and adjusting the flow of the process.

Suggested Pipeline Modules:

- **PDF Parser**: Create a dedicated module for parsing and chunking documents.
- **Vector Generator**: A separate module to handle vector generation from the chunks.
- **Query Engine**: Another module to manage query processing and retrieval.
- **Retriever Function**: A modular function that can be swapped or modified to support different retrieval techniques.

---

5. Logging Changes and Tracking

Implement **logging** to track changes and experiments within the system. This will allow you to:

- Log changes in the system (e.g., vector model switches, new document formats, etc.).
- Track performance metrics for different experiments.
- Debug and monitor the flow of data through the system.

Use Python's `logging` module for structured logging.

```
import logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# Example logging for change tracking
logger.info("Started experiment with new vectorizer model.")
```

---

6. File Management with YAML

To manage configurations, file paths, and directory structures, use **YAML files** for better flexibility and readability.

Example `config.yaml`:

```
paths:
  pdf_directory: "pdfs/"
  parsed_directory: "parsed_content/"
  embeddings_directory: "embeddings/"
  vector_store_directory: "vector_store/"

model:
  vector_model: "all-mpnet-base-v2"
  retriever_function: "cosine_similarity"
```

- Update the pipeline to read from this configuration file using a YAML parser.

---

7. Environment Management with Poetry

Instead of manually managing environments with `requirements.txt`, use **Poetry** for automatic dependency management and virtual environment setup.

- To install dependencies:

  ```
  poetry install
  ```

- To create a virtual environment:

  ```
  poetry shell
  ```

This simplifies the setup process for new users and ensures compatibility between different environments.

---

8. Advanced Experimentation Ideas

A. Distillation for Better Latency

Experiment with **distillation learning** for faster model inference. Distillation can reduce the size of a large model (like a transformer) without losing much performance, which helps reduce latency during query generation.

- Explore libraries like **Hugging Face's** `distilBERT` for distillation.

B. Dimensionality Reduction for Faster Retrieval

Use **PCA (Principal Component Analysis)** or **LDA (Linear Dicriminant Analysis)** to reduce the dimensionality of sentence embeddings, which will speed up the retrieval process.

- Apply these techniques to the vector store to improve query response time.

C. Prompt Injection

Test **prompt injection** techniques where specific patterns are introduced into the query to improve the model's understanding and retrieval accuracy.

- Use more structured input to help the model retrieve more relevant answers.

---

9. Advanced Techniques for Experimentation

A. Routing and Query Structuring

Implement advanced query processing techniques like **routing** and **query structuring**.

- **Routing**: Route different types of queries to specialized retrieval models or pipelines based on query type (e.g., FAQ queries, specific document type queries).
- **Query Structuring**: Break down the query into components or entities and match them against structured data (e.g., extract keywords and match them to specific sections in documents).

---

10. Contributing to the Project

We welcome contributions from the community! Here's how you can contribute:

- Fork the repository and create a new branch for your experiment.
- Add your changes, whether it's a new notebook, module, or improvement to the pipeline.
- Submit a pull request and describe the experiment or feature you implemented.

Please ensure that your code is well-documented and passes all tests before contributing.

---

11. Future Experiment Ideas

Here are a few more areas for future experimentation:

- Integrating a **search engine** like Elasticsearch or Solr for more complex querying and faster retrieval.
- Exploring the use of **multi-modal** data (e.g., images, audio) in the retrieval process.
- Fine-tuning or experimenting with more **advanced transformer models** like GPT-3 or BERT for specialized tasks.

Conclusion

ContextCraft offers a robust foundation for document parsing and query retrieval tasks, and with the guidelines above, users can expand the project in various ways to experiment with new techniques, models, and optimizations. We encourage you to dive into the code, try out these experiments, and contribute back to make ContextCraft even better!
