Capstone Project – Phase 1 Submission

Abstract

In this project, we propose an automated pipeline that generates a complete, structured LaTeX report directly from two user inputs: a question PDF and its corresponding solution Jupyter notebook. Students generally write all their work—including code, explanations, markdown, and outputs—inside Jupyter notebooks, but final submissions are required in LaTeX format. Converting notebook content into properly formatted reports is repetitive, time-intensive, and error-prone. Our system aims to remove this manual effort entirely.

Unlike large end-to-end LLM solutions, which require substantial compute resources, our workflow is intentionally designed to run efficiently on standard student hardware by using smaller models and classical processing. The pipeline includes PDF block detection, block parsing using Nougat, classification and linking of question blocks, tree-structured question reconstruction, notebook code summarization, semantic matching of notebook content to questions, and automated generation of formatted LaTeX answer blocks.

The expected outcome is a complete end-to-end workflow that significantly reduces the time and labour involved in academic report preparation. The system reads the question PDF and notebook, reconstructs question–answer mappings, and produces a polished LaTeX document ready for submission with minimal human intervention.

Introduction

In many programming and data-centric courses, students solve assignments within Jupyter notebooks, where code, markdown explanations, and output figures are all naturally integrated. However, final submissions usually require a separate, cleanly structured LaTeX report. This leads to repetitive manual work such as copying outputs, rewriting explanations, formatting figures, and combining everything into a consistent structure.

Although large LLMs could theoretically automate this entire workflow, their computational demands make them impractical for typical student hardware. Our project addresses this challenge by developing a modular, efficient system that intelligently reads the assignment PDF, reconstructs the hierarchical question structure, interprets notebook content, and generates a complete LaTeX report using lighter-weight models.

The significance of our work lies in its practicality. It reduces time spent on formatting rather than learning, ensures consistency across submissions, and lowers the barrier to automated report generation for students who lack access to high-end GPUs.

Methodology

Our methodology follows a structured, multi-stage pipeline combining deterministic algorithms, efficient multimodal models, and controlled LLM-driven reasoning.

1. **PDF Processing & Question Reconstruction**
   - Detect all semantic blocks in the assignment PDF using a block segmentation model.
   - Parse each block into clean text using Nougat.
   - Classify blocks into categories such as *question*, *metadata*, *instruction*, and *note* using a one-shot Qwen-7B classifier.
   - Use a two-step Qwen-7B linking pipeline to determine whether a block is a **new question** or a **continuation**.
   - Construct a final hierarchical JSON structure containing question descriptions, sub-questions, and sub-sub-parts.
2. **Notebook Processing**
   - Parse the `.ipynb` file into code cells, markdown cells, and output cells.
   - Generate short technical summaries for each code block using a small LLM, preserving function names and logic.
   - Extract global context such as function and class definitions.
   - Build a searchable database combining markdown, outputs, and code summaries.
   - Use embeddings to find the most relevant notebook blocks corresponding to each question and sub-question.
   - Apply temporal heuristics (e.g., consecutive and sequential boosting) to refine the matches.
3. **Answer Generation**
   - Compile a context bundle for each question: its text, matched notebook blocks, global definitions, and summaries.
   - Use an LLM to generate a structured LaTeX answer block including explanations, code, equations, tables, and figures when relevant.
   - Ensure that answers remain self-contained and appropriately formatted.
4. **Final Report Synthesis**
   - Merge all generated answer snippets into a master LaTeX document.
   - Insert sections corresponding to each question and sub-question.
   - Produce a complete, compile-ready submission PDF.
5. **Additional Planned Extensions**

- Support for alternative answer formats such as `.py` scripts.
- Optional Google search verification for vague explanations.
- Image parsing for PDF diagrams and notebook output figures.
- Deployment of the entire workflow as a web application.

Work Done So Far: Successes and Failures

We document here the major attempts, challenges, failures, and breakthroughs achieved up to the current stage of the project.

Initial Failures

Early prototypes relied on small general-purpose models such as DeBERTa-v3-base and Gemma-2B for PDF block classification and question linking. These models failed in two critical areas:

- **Block Classification Failure:** The models frequently misidentified metadata and instructions as questions, and could not consistently distinguish continuation blocks.
- **Linking Failure:** Zero-shot NLI models treated each block independently and lacked the contextual reasoning to detect question continuation across pages.

Thus, standard lightweight models were insufficient for understanding academic PDF structures.

Breakthrough Using Qwen-7B

A unified pipeline using Qwen-7B solved both core issues.

1. One-Shot Block Classifier By providing a carefully curated one-shot example with simplified labels (question, metadata, instruction, note), Qwen-7B achieved highly accurate classification, solving all previously encountered mislabeling issues.

2. Two-Step Linking System A robust linking pipeline was constructed:

- **Step 1:** Qwen-7B generates a structured explanation for “NEW” vs. “CONT” decisions using a 4-block sliding window.
- **Step 2:** A second Qwen-7B call extracts the final label from the explanation.

Additional prompt engineering ensured stability, reduced hallucination, and prevented contradictory responses. This resulted in consistent and accurate question-continuation linking.

Current Status

- PDF block detection: **Completed**
- Nougat parsing: **Completed**
- Block classification: **Accurate and stable**
- Block linking: **Fully functional**
- JSON question tree: **Generated**
- Notebook parsing and code summarization: **Completed**
- Notebook-to-question mapping: **In progress**
- LaTeX synthesis engine: **Designed and partially implemented**

Individual Member Contributions

All team members contributed to the project, with responsibilities distributed to maintain balanced workload while allowing each member to lead specific components.

- **Debanjan Saha (23607): Pipeline Architecture & PDF Intelligence**
  - Heavy: Planning of pipeline and implemented the PDF block detection and segmentation framework.
  - Medium: Integrated Nougat parsing and established the data flow conventions used by later stages.
  - Light: Built internal utilities for block normalization, structured storage formats, and debugging tools.
- **Adithya K Anil (23619): Question Semantics & Structuring**
  - Heavy: Developed the rule set for JSON and juputer block linking and implemented the structural constraints for multi-page question reconstruction.
  - Medium: Contributed to tuning the block linking prompt templates.
  - Light: Assisted in verifying the JSON question hierarchy produced during early tests.
- **Vinay (23653): Notebook Interpretation & Code Context**
  - Heavy: Implemented notebook blocks' dependency and division rules along with utilities and the schema for representing code, markdown, and outputs uniformly.
  - Medium: Wrote helper logic for handling external `.py` files and additional formats.

- Light: Performed initial experiments on extraction of comments and inline explanations from code blocks.

- **Subhadeep Singh (23682): Answer Mapping & LaTeX Assembly**
  - Heavy: Set up the LaTeX snippet generator used for rendering cleaned answer blocks.
  - Medium: Helped refine the heuristics for selecting relevant explanation segments.
  - Light: Contributed utilities for merging final LaTeX snippets into a single document.
  -

- **Saksham Maitri (23787): Integration & Application Layer**
  - Heavy: Implemented the integration skeleton for connecting PDF, notebook, and answer-generation modules.
  - Medium: Added optional components like image parsing for figures and diagrams.
  - Light: Drafted the initial plan for the web deployment interface.