Mini Project- 2: Code Files Description

Rakesh | Sai Krishna | Sai | Bhargav | Krishnendu Ghosh

To start and run the code files, follow these steps:

1. Installation

a. **Download and extract the project**

Download the MiniProject(Coding-part).zip

b. **Install required dependencies**
pip install -r requirements.txt

2 Running the Application

a. **Navigate to the project directory**
cd MiniProject

b. **Run the Streamlit app**
streamlit run app.py

3 Using the Interface

a. Start the application and click "Get Started"
b. Select question types you want to generate
c. Choose a context source:
d. Sample contexts: Select from pre-defined mathematical topics
e. Custom input: Enter your own mathematical text
f. Dataset: Upload or select a JSONL file
g. Specify the number of questions for each selected type
h. Configure options:
i. Enable/disable deduplication
j. Set difficulty level (if applicable)
k. Click "Generate Questions" to create the questions
l. View and export the generated questions

The project file is structured & code flow as follows:

1.Project Structure

The project follows a modular structure with separate components for question generation, utility functions, and data handling.

Directory Structure

MiniProject/

├── app.py → **Main application file with Streamlit UI**
├── data/ → **Data directory**
│ ├── example_model_solutions.jsonl
│ └── train_socratic.jsonl
├── question_generators/ → **Question generation modules**
│ ├── _init_.py
│ ├── mcq.py → **Multiple-choice question generator**
│ ├── short_answer.py → **Short answer question generator**
│ └── fill_blank.py → **Fill-in-the-blank question generator**
├── utils/ → **Utility modules**
│ ├── _init_.py
│ ├── deduplication.py → **Question deduplication logic**
│ └── evaluation.py → **Question evaluation metrics**

2. Components

a. Core Components

- app.py
    - The main application file that contains the Streamlit UI and core logic for the question generator.

Key Functions:

- main(): Entry point that sets up the Streamlit interface
- generate_questions(): Handles question generation based on user selections
- display_questions(): Formats and displays generated questions

Key Functions:

- calculate_bleu_scores(): Computes **BLEU** scores for generated questions

b. Question Generators

- mcq.py

  Implements the MCQGenerator class for creating multiple-choice questions with randomized answer options.

Key Methods:

- generate_from_problem(): Creates MCQs from problem-solution pairs
- generate_from_context(): Creates MCQs from mathematical text

  Various specialized methods for different mathematical topics
- short_answer.py

  Implements the ShortAnswerGenerator class for creating short answer questions.

Key Methods:

- generate_from_problem(): Creates short answer questions from problem-solution pairs
- generate_from_context(): Creates short answer questions from mathematical text
- fill_blank.py

  Implements the FillBlankGenerator class for creating fill-in-the-blank questions with consistent formatting.

Key Methods:

- generate_from_problem(): Creates fill-in-the-blank questions from problem-solution pairs
- generate_from_context(): Creates fill-in-the-blank questions from mathematical text

c. Utility Modules

- deduplication.py

  Contains the QuestionDeduplicator class that prevents similar questions from being generated.

Key Methods:

- is_duplicate(): Checks if a question is similar to previously generated ones
- add_question(): Adds a question to the tracking system
- evaluation.py

Provides functions for evaluating the quality of generated questions using metrics like **BLEU** scores.

Key Functions:

- calculate_bleu(): Computes **BLEU** scores between generated and reference questions
- evaluate_questions(): Performs comprehensive evaluation of question quality

Human Evaluation Results Sheet Link:

https://docs.google.com/spreadsheets/d/1sl-TPm9bjBISHXOh76JV50i2Fk-dTq7JymU3u8_e-pk/edit?usp=sharing