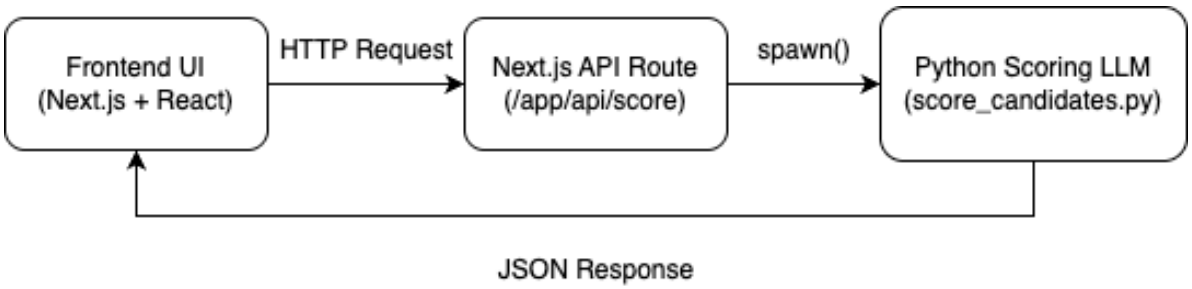
Technical Report - Code Challenge: LLM-Powered Candidate Screening & Scoring System

1. Architecture Diagram



2. Key Prompt Design Decisions

- **System Prompt:** Provides the primary instruction defining the AI as a technical recruiter with expertise in GoLang and Ruby on Rails. This sets the overall context and tone for evaluation.
- **User Prompt:** Incorporates the job description provided by the recruiter (≤200 characters) to tailor the evaluation to the specific role. It ensures the AI considers the unique requirements of each job.
- **Few-Shot Examples:** Includes a small set of sample candidate evaluations in proper JSON format. This guides the response format and scoring scale, ensuring consistency, clarity, and expected output structure.

3. Challenges & Solutions

- **JSON Parsing:** Challenge: Debug logs or extra formatting (e.g., markdown code block delimiters) risk corrupting the JSON output, leading to parse errors. Solution: Route all debugging output to stderr (using file=sys.stderr) and apply regexp cleaning (e.g., removing “```json” wrappers) to ensure only valid JSON reaches the parser.
- **Rate Limiting & API Quota:** Challenge: Hitting the OpenAI API rate limits (e.g., 429 errors) during production and tests. Solution: Implement exponential backoff retry logic in the Python script and creating a mock OpenAI response in the Jest test file.
- **Process Timing in Mocks:** Challenge: In test environments, the spawn process may trigger the “close” event too soon, resulting in an empty response string. Solution: Enhance the mocked spawn function with a slight delay (via setTimeout) to ensure that all data (stdout) is emitted before triggering the “close” event. Very helpful.