Use Cases for AI-Driven Fıkra Generator

Use Case 1: Generate a Turkish Fıkra

Use Case ID: UC-01
Description: A user inputs a prompt through the web-based interface to request the generation of a humorous Turkish fıkra, and the system generates a culturally relevant joke.
Actor: General User (e.g., anyone accessing the web interface)
Preconditions:

- The FastAPI server is running, and the fine-tuned Mistral7BInstructv0.2 model is loaded on the Apple M4 Pro MacBook Pro.
- The user has access to the web interface at http://localhost:8000.
- The system has a stable internet connection (optional, as it runs on-device).
  Postconditions:
- A humorous Turkish fıkra is displayed on the web interface, relevant to the user's prompt.
- The generated fıkra aligns with Turkish cultural humor and avoids offensive content.
  Main Flow:

1. The user navigates to the web interface (http://localhost:8000).
2. The user sees the initial screen with a prompt input field ("What can I help with?").
3. The user enters a prompt (e.g., "Generate a Nasreddin Hoca joke" or "Tell me a funny Turkish anecdote").
4. The system processes the prompt using the fine-tuned Mistral model, leveraging 4-bit QLoRA and LoRA adapters for efficient computation.
5. The system streams the generated fıkra in real-time via the FastAPI server, with a first-token latency of approximately 3.1 seconds and throughput of ~1.8 tokens/s.
6. The generated fıkra is displayed progressively in the web interface's output area.
7. The user reads the completed fıkra, which has a 78% likelihood of being humorously relevant based on human evaluations.
   Alternative Flows:

- **A1: Invalid Prompt Input**
  - If the user submits an empty or invalid prompt, the UI displays an error message (e.g., "Please enter a valid prompt").
  - The user is prompted to re-enter a valid prompt, returning to step 3.
- **A2: System Overload or Memory Issue**
  - If the system encounters an out-of-memory error, the UI displays a loading error with a suggestion to refresh the page.
  - The system logs the error for debugging, and the user retries the request.
    Exceptions:
- If the server is not running, the UI displays a "Server Unavailable" error.

- If the model generates an incomplete fıkra due to reaching the max_new_tokens limit (no explicit <END_OF_JOKE> token), the output may appear truncated but is still displayed.
  Frequency: Frequent, as this is the primary function of the application.
  Assumptions:
- The user understands basic Turkish to interpret the fıkra.
- The system is deployed on the specified hardware (24 GB Apple M4 Pro MacBook Pro).
  Notes:
- The fıkra generation leverages a dataset of ~450 curated Turkish jokes, ensuring cultural relevance.
- The system achieves a 34% improvement in humor relevance over the base model, as validated by human annotators.

Use Case 2: Interact with the Streaming API for Fıkra Generation

Use Case ID: UC-02
Description: A developer or advanced user interacts with the FastAPI-based inference server to programmatically request and receive streamed Turkish fıkra outputs.
Actor: Developer or Advanced User (e.g., a programmer integrating the fıkra generator into another application)
Preconditions:

- The FastAPI server is running (uvicorn fikra_api_server:app --port 8000).
- The fine-tuned Mistral model is loaded with INT8 GGUF quantization (7.4 GB).
- The developer has access to the API documentation and necessary tools (e.g., cURL, Postman, or a programming environment).
  Postconditions:
- The developer receives a streamed response containing a generated Turkish fıkra, suitable for integration into external applications.
  Main Flow:

1. The developer sends an HTTP POST request to the API endpoint (e.g., http://localhost:8000/generate) with parameters such as the prompt, temperature, and max_new_tokens.
2. The FastAPI server validates the request parameters and forwards the prompt to the fine-tuned Mistral model.
3. The model processes the prompt using optimized inference techniques (e.g., sliding-window KV caching, INT8 quantization).
4. The server streams the generated tokens back to the developer in real-time, with a first-token latency of ~3.1 seconds and throughput of ~1.8 tokens/s.
5. The developer's application receives and processes the streamed fıkra for display or further use (e.g., in a chatbot or mobile app).
   Alternative Flows:

- **A1: Invalid API Parameters**
  - If the request contains invalid parameters (e.g., negative max_new_tokens), the API returns a 400 Bad Request error with details.
  - The developer corrects the parameters and resubmits the request.

- **A2: Server Overload**
    - If the server is handling too many concurrent requests, it returns a 429 Too Many Requests error.
    - The developer implements retry logic with exponential backoff and resubmits the request.
      Exceptions:
- If the server is down, the developer receives a connection error and must wait for the server to restart.
- If the model over-generates due to the lack of an <END_OF_JOKE> token, the response may be longer than expected but is still usable.
  Frequency: Occasional, depending on developer integration needs.
  Assumptions:
- The developer is familiar with API interaction and JSON-based responses.
- The API is running locally or on an accessible server.
  Notes:
- The API supports streaming for a natural, progressive output experience, improving perceived responsiveness.
- Parameters like temperature and top-p sampling allow developers to control the creativity and coherence of the generated fıkra.

Use Case 3: View and Test the Web Interface

Use Case ID: UC-03
Description: A user interacts with the minimal web-based interface to explore the fıkra generation functionality, view generated outputs, and test the system's responsiveness.
Actor: General User or Tester (e.g., project evaluators, team members, or curious users)
Preconditions:

- The FastAPI server is running, and the web interface is accessible at http://localhost:8000.
- The fine-tuned Mistral model is loaded and operational.
- The user has a compatible web browser (e.g., Chrome, Firefox).
  Postconditions:
- The user successfully navigates the interface, inputs prompts, and views generated fıkra outputs.
  Main Flow:

1. The user opens the web interface in a browser, landing on the initial screen with a prompt input field.
2. The interface displays a clear input area and a "Generate" button, as shown in the provided screenshots.
3. The user enters a prompt (e.g., "Tell me a regional Turkish joke").
4. The user clicks "Generate," and the interface shows a loading indicator while the fıkra is generated.
5. The generated fıkra appears progressively in the output area, streamed from the FastAPI server.
6. The user views the completed fıkra, which is formatted for readability and cultural relevance.
7. The user can input additional prompts to test different joke styles or categories (e.g., Nasreddin Hoca, contemporary humor).
   Alternative Flows:

- **A1: Interface Error**
    - If the interface encounters an error (e.g., server timeout), it displays an error message with a suggestion to refresh or retry.
    - The user refreshes the page or retries the prompt.
- **A2: Unresponsive Interface**
    - If the interface freezes due to heavy model computation, the loading indicator persists, and the user waits up to 15 seconds for the response.
    - If no response is received, the user refreshes the page.
      Exceptions:
- If the server is not running, the interface displays a "Cannot connect to server" error.
- If the browser is incompatible, some UI elements may not render correctly, but core functionality remains accessible.
  Frequency: Frequent during testing and demonstration phases.
  Assumptions:
- The user has basic familiarity with web interfaces.
- The system is deployed on the specified hardware with adequate thermal management (e.g., laptop lid open, fans on).
  Notes:
- The UI is intentionally minimal to focus on joke generation, with clear labeling and responsive design for accessibility.
- Screenshots provided in the project report illustrate the interface's simplicity and functionality (initial screen, in-progress generation, and final output).

Use Case 4: Deploy and Run the Fıkra Generator Locally

Use Case ID: UC-04
Description: A technical user deploys and runs the fıkra generator on a 24 GB Apple M4 Pro MacBook Pro to enable local joke generation.
Actor: Technical User (e.g., developer, system administrator, or project team member)
Preconditions:

- The user has a 24 GB Apple M4 Pro MacBook Pro with Python 3.9+, PyTorch 2.3, Transformers 4.41, peft, fastapi, uvicorn, and BitsAndBytesMetal installed.
- The source code is cloned from the GitHub repository: https://github.com/Maltepe-University-SWEng/term-project-team-3.
- The fikralar.json dataset file is placed in the project directory.
  Postconditions:
- The fıkra generator is fully deployed, with the fine-tuned model trained and the FastAPI server running, ready for user interaction.
  Main Flow:

1. The user sets up a virtual environment and installs dependencies (Python 3.9+, PyTorch 2.3, etc.) using the BitsAndBytesMetal branch for Apple Silicon compatibility.
2. The user places the fikralar.json dataset file in the project directory, ensuring it follows the instruction-response JSONL format.
3. The user runs the training script (python fikra_train.py) to fine-tune the Mistral model, which takes approximately 2 hours and 40 minutes and uses ~19 GB of memory.
4. The trained model and LoRA adapters are saved to ./mistral_fikra_output.
5. The user starts the FastAPI server (uvicorn fikra_api_server:app --reload --port 8000), which loads the fine-tuned model with INT8 GGUF quantization.
6. The user verifies that the server is running by accessing http://localhost:8000 in a browser.
7. The system is now ready for users to generate fıkra via the web interface or API.
Alternative Flows:

- **A1: Dependency Installation Failure**
    - If a dependency fails to install due to version conflicts, the user consults the project documentation for specific version requirements and retries installation in a clean virtual environment.
- **A2: Training Failure Due to Thermal Throttling**
    - If the MacBook Pro overheats during training, the script pauses between epochs for thermal recovery.
    - The user ensures proper cooling (lid open, fans on high) and resumes training.
- **A3: Dataset Formatting Error**
    - If the fikralar.json file is incorrectly formatted, the training script fails with an error.
    - The user runs the provided validation script to check the dataset and corrects any issues.
Exceptions:
- If the hardware lacks sufficient memory (less than 24 GB), the training process fails with an out-of-memory error.
- If the user skips the BitsAndBytesMetal installation, the system cannot leverage Apple Silicon optimizations, resulting in poor performance.
Frequency: One-time setup, with occasional retraining if the dataset is updated.
Assumptions:
- The user has administrative access to the MacBook Pro and familiarity with command-line operations.
- The hardware meets the specified requirements (24 GB unified memory, M4 Pro chip).
Notes:
- The deployment process is optimized for consumer hardware, ensuring privacy and offline availability.
- Thermal management is critical during training to avoid performance degradation.

Use Case 5: Evaluate Generated Fıkra for Quality

Use Case ID: UC-05
Description: A tester or evaluator assesses the quality of generated Turkish fıkra to ensure they meet humor relevance,

cultural appropriateness, and low toxicity standards.
Actor: Tester or Evaluator (e.g., project team member, human annotator, or quality assurance personnel)
Preconditions:

- The fıkra generator is deployed and accessible via the web interface or API.
- The tester has access to evaluation tools (e.g., Perspective API for toxicity checks) and guidelines for humor relevance assessment.
- A set of test prompts is prepared to generate a sample of fıkra outputs.
  Postconditions:
- The tester compiles a report on the fıkra quality, including metrics for humor relevance (target: 78%), perplexity (target: 17.8), and toxicity (target: ≤2.5%).
  Main Flow:

1. The tester selects a set of test prompts covering various fıkra categories (e.g., Nasreddin Hoca, regional jokes, contemporary humor).
2. The tester inputs each prompt into the web interface or API to generate fıkra outputs.
3. The tester collects the generated fıkra and evaluates them for:
   - **Humor Relevance**: Using human annotation, scoring whether the fıkra is funny and culturally relevant (based on guidelines provided to annotators).
   - **Toxicity**: Running the outputs through the Perspective API to ensure toxicity remains at or below 2.5%.
   - **Perplexity**: If applicable, computing perplexity on a held-out test set using provided scripts to verify a value of ~17.8.
4. The tester aggregates results, noting that humor relevance averages 78% (up from 44% in the base model) and toxicity is stable at 2.5%.
5. The tester records inter-annotator agreement (kappa score of 0.69) to confirm reliable humor assessments.
6. The tester compiles a report summarizing the quality metrics and any observed issues (e.g., over-generation due to missing <END_OF_JOKE> token).
   Alternative Flows:

- **A1: Inconsistent Humor Evaluation**
  - If annotators disagree significantly (kappa score < 0.6), the tester revises the evaluation guidelines and recalibrates annotators before re-evaluating a subset of outputs.
- **A2: Toxicity Detection**
  - If a generated fıkra exceeds the 2.5% toxicity threshold, the tester flags it for manual review and potential dataset cleaning.
  - The team investigates whether the issue stems from the dataset or model bias and updates the training process if needed.
    Exceptions:
- If the Perspective API is unavailable, the tester relies on manual toxicity checks, which may slow the evaluation process.

- If the generated fıkra are consistently incomplete, the tester notes the lack of an <END_OF_JOKE> token as a limitation and recommends future implementation.
  Frequency: Periodic, during testing phases or after model updates.
  Assumptions:
- The tester is trained in humor evaluation and familiar with Turkish cultural context.
- The evaluation process follows the methodology outlined in the project report (e.g., blind assessments, multiple annotators).
  Notes:
- The significant improvement in humor relevance (+34%) and perplexity reduction (-28%) validates the fine-tuning approach.
- The evaluation process is critical for ensuring the system meets its proof-of-concept objectives.

Conclusion

These use cases cover the primary interactions with the AI-Driven Fıkra Generator, aligning with the project's scope as a proof-of-concept application. They emphasize:

- **Core Functionality**: Generating culturally relevant Turkish fıkra with high humor relevance (UC-01, UC-02).
- **User Accessibility**: Providing a simple, intuitive web interface for general users and testers (UC-03).
- **Technical Deployment**: Enabling local deployment on consumer hardware with optimized performance (UC-04).
- **Quality Assurance**: Validating the system's outputs for humor, cultural appropriateness, and safety (UC-05).