Maltepe University

Faculty of Engineering and Natural Sciences

Department of Software Engineering

SE 403 01 Software Project Management

Project Report – AI-Driven Fıkra Generator

Project Group Members:

- Omar Hamed Abdelaal ELBEDIWI (21 07 06 819)
- Elife AĞGÜRBÜZ (210706007)
- Serdar SÖYLEMEZ (210706013)
- Cihan Enes ÇOLAK (210706903)
- Ece GÖKÇE (220706306)

Content

1. Scope of the Project

- The primary objective of this project was to develop a proof-of-concept application that uses Mistral AI to generate coherent and humorous Turkish "fıkra" (jokes or short humorous anecdotes) by training on a curated dataset of approximately 450 jokes. The project, conducted under SE 403 01 -- Software Project Management, aimed to demonstrate the feasibility of fine-tuning a large language model (LLM) on a small dataset while ensuring cultural relevance and humor for Turkish conversational agents, all computed on-device using a 24 GB Apple M4 Pro MacBook Pro.

- Our scope specifically included creating a Turkish language model capable of understanding the unique linguistic and cultural aspects of Turkish humor, which presents significant challenges compared to traditional English-language LLMs. This specialization in Turkish fıkra required careful dataset curation and model training to capture the nuances of Turkish expressions, cultural references, and traditional joke structures that might not translate well in general-purpose language models.

- The scope deliberately focused on proof-of-concept development rather than production-ready deployment, prioritizing the investigation of whether a relatively small dataset of culturally specific content could meaningfully improve a large language model's performance in a specialized domain. This exploratory approach allowed us to analyze the minimum viable dataset size for achieving meaningful improvements in humor generation while working within strict hardware constraints.

2. Functionality of the Project

The software's primary features include:

- **Joke Generation**: The fine-tuned Mistral7BInstructv0.2 model generates Turkish fıkra based on user prompts, achieving a humor relevance of 78% (up from 44%) and reducing perplexity by 28%. The model can generate jokes across multiple traditional Turkish categories including Nasreddin Hoca stories, regional jokes, and contemporary humor formats. This significant improvement demonstrates that even with a modest dataset of approximately 450 high-quality examples, a specialized model can dramatically outperform general models in culturally specific tasks.
- **On-Device Computation**: Training and inference are performed entirely on a 24 GB Apple M4 Pro MacBook Pro, ensuring privacy and offline availability. This approach eliminates the dependency on external cloud services, reducing latency and addressing privacy concerns for users. The ability to run inference locally also makes the application more accessible in regions with limited internet connectivity. Our optimization techniques allow the model to run smoothly even on consumer hardware, demonstrating that specialized AI applications can be deployed outside enterprise environments.
- **Simple User Interface**: A minimal web-based UI allows users to input prompts and receive generated jokes, as demonstrated in the attached screenshots. The interface was intentionally kept straightforward to focus user attention on the joke generation capabilities rather than complicated controls. The clean design enables users of various technical backgrounds to interact with the AI model without requiring specialized knowledge of machine learning concepts or parameters. Input fields are clearly labeled, and generated text appears in a readable, well-formatted display area.
- **FastAPI Integration:** A streaming API serves generated jokes in real-time, with an inference throughput of ~1.8 tokens/s and a first-token latency of 3.1 seconds. The streaming capability creates a more natural interaction experience, allowing users to see the joke being constructed progressively rather than waiting for the entire response. This design choice significantly improves the perceived responsiveness of the system, as users don't need to wait for complete joke generation before beginning to read the content.
- **Unique Attribute:** The project leverages 4-bit QLoRA and LoRA adapters to fine-tune a 7-billion-parameter model on limited hardware, achieving significant quality gains despite a small dataset. This technical approach represents a breakthrough in efficient model training, as it demonstrates that domain-specific improvements can be achieved without access to enterprise-grade hardware or massive datasets. The selective parameter updating through LoRA adapters allows the model to incorporate new knowledge while maintaining its core linguistic capabilities.

3. Functions Available at the Beginning of the Project

At the project's outset, the following components were available:

- The base Mistral7BInstructv0.2 model, an open-source LLM accessible via Hugging Face. This foundation model provided strong multilingual capabilities but lacked specialized knowledge of Turkish cultural humor and joke structures. Mistral7B was selected for its balance of performance and resource requirements, making it suitable for fine-tuning on consumer hardware while still offering sophisticated language understanding capabilities.

- Public domain Turkish joke datasets and internal archives for data collection. These sources provided the raw material for our training corpus but required extensive cleaning, formatting, and preparation before they could be used effectively for model training. The available datasets contained various issues including duplicates, inappropriate content, and inconsistent formatting that needed to be addressed.

- Development tools such as PyTorch 2.3, Transformers 4.41, and BitsAndBytesMetal for Apple Silicon compatibility. No pre-existing UI or API integration was available at the start. These foundational libraries provided the essential machine learning infrastructure but required significant configuration and optimization to work efficiently with our specific hardware constraints and application requirements.

The initial project environment lacked any joke-specific training data preprocessing pipelines, specialized tokenization for Turkish language handling, or efficient inference mechanisms for deployment on consumer hardware. The base model, while capable in general language tasks, showed significant deficiencies in understanding cultural contexts and generating appropriate humor specific to Turkish traditions.

Our starting point also lacked any user-facing components or integration layers that would make the model accessible to end users. We needed to develop these elements entirely from scratch, guided by best practices in human-AI interaction design while working within the constraints of our available development resources.

4. Functions Added During Development

The following functionalities were introduced during the project:

- **Dataset Preparation**: A mini-corpus of ~450 Turkish fıkra was collected, cleaned (removing profanity and duplicates), and formatted into instruction-response JSONL for training. This process involved careful manual review to ensure cultural appropriateness and humor quality. Each joke underwent multiple rounds of verification to confirm its suitability for the training corpus. We developed custom preprocessing scripts to standardize formatting, handle Turkish-specific characters correctly, and ensure consistent structure across the dataset. Special attention was paid to maintaining the authentic voice and style of traditional Turkish humor while removing potentially offensive or inappropriate content.
- **Model Fine-Tuning**: The Mistral model was fine-tuned using 4-bit QLoRA and LoRA adapters (rank 8, targeting q_proj/v_proj), reducing memory usage to fit within 24 GB while achieving a 34% increase in humor relevance. The fine-tuning process involved careful hyperparameter optimization to balance learning rate, batch size, and training duration. We implemented early stopping mechanisms based on validation loss to prevent overfitting on our relatively small dataset. The adaptation focused specifically on attention mechanisms within the model, allowing it to better capture the structural patterns and contextual cues specific to Turkish humor while preserving the model's general language capabilities.
- **Inference Optimization**: The model was quantized to INT8 GGUF (7.4 GB) for deployment, with optimizations like sliding-window KV caching to handle long prompts without out-of-memory (OOM) errors. These technical optimizations were critical for achieving acceptable performance on consumer hardware. The sliding-window approach allowed the model to process longer contexts than would otherwise be possible within memory constraints, enabling more natural conversation flows. We also implemented custom temperature and top-p sampling controls to balance creativity and coherence in generated jokes, allowing for adjustment based on user preference and joke style.
- **User Interface**: A basic web interface was developed, allowing users to input prompts and view generated jokes (see attached screenshots). The interface includes error handling for invalid inputs, appropriate loading indicators during generation, and a clean, intuitive layout designed for ease of use. The design prioritizes readability and simplicity, ensuring users can focus on the content rather than navigating complex controls. We incorporated responsive design principles to ensure usability across different device types and screen sizes.
- **API Server**: A FastAPI-based inference server was implemented to stream generated jokes, supporting parameters like temperature and max_new_tokens. The server architecture was designed with asynchronous request handling to maintain responsiveness even when processing multiple requests. The streaming implementation provides immediate feedback to users as tokens are generated, significantly improving the perceived responsiveness of the system. Error handling and rate limiting were implemented to ensure system stability under various load conditions.

- **Documentation**: Comprehensive dataset documentation, user guides, technical documentation, and project management deliverables (e.g., scope document, risk management plan) were created. The documentation suite includes detailed explanations of model parameters, training procedures, and deployment considerations to facilitate future development and maintenance. User guides were written with non-technical audiences in mind, ensuring accessibility for all potential users. Technical documentation includes architecture diagrams, API specifications, and detailed explanations of optimization techniques to support future development efforts.

Each of these components required iterative development and testing to achieve the desired functionality while working within our hardware constraints. The integration between components was carefully designed to ensure smooth data flow and responsive performance throughout the application stack. Special attention was paid to error handling and graceful degradation when system limits were approached, ensuring a robust user experience even under sub-optimal conditions.

5. Missing Parts

The following intended features were not implemented, with reasons:

- **Complex Backend**: A scalable, production-ready backend for user request storage and database management was deemed out-of-scope due to the focus on proof-of-concept development and time constraints. While initial designs included considerations for persistent storage of user interactions and generated content, implementing a full database architecture would have significantly expanded the project scope beyond our available resources. The current implementation uses in-memory storage for temporary session data, sufficient for demonstration purposes but lacking the persistence and scalability features that would be required in a production environment.
- **Advanced UI Features**: Elaborate UI features (e.g., joke style selection, user profiles) were not prioritized, as the primary goal was to demonstrate joke generation functionality. Early prototypes included mockups for more sophisticated interface elements, but these were simplified in the final implementation to focus development efforts on core model performance. Future iterations could incorporate these features to enhance user experience, but they were not essential for validating the primary hypothesis regarding fine-tuned humor generation.
- **RLHF Integration**: Reinforcement Learning from Human Feedback (RLHF) using real user laughter ratings was planned but not implemented due to time limitations and the need for a larger user base. This approach would have allowed continuous improvement of the model based on actual user reactions to generated jokes, creating a virtuous feedback loop for quality enhancement. The infrastructure requirements for collecting, storing, and processing user feedback at scale represented a significant expansion of project scope that could not be accommodated within our timeline.

- **<END_OF_JOKE> Token**: The model lacks an explicit token to stop joke generation, leading to over-generation until max_new_tokens is reached. This was not addressed due to the small dataset and training constraints. Early experiments with custom stop tokens showed promising results, but achieving reliable detection and appropriate stopping behavior would have required additional training data specifically formatted with consistent ending markers. Given our limited dataset size, we opted to use maximum token limits as a simpler alternative, accepting the occasional over-generation as a limitation of the current implementation.

These omitted features represent potential areas for future development that would enhance the system's usability and performance without fundamentally changing the core approach or architecture. Their absence does not undermine the primary research objectives of the project but does limit certain aspects of real-world applicability and user experience optimization.

6. Design Documents

The project relied on the following design considerations:

- **Model Architecture:** Fine-tuning used LoRA adapters on attention (q_proj, v_proj) and MLP layers, with a rank of 8 and alpha of 32, to minimize memory usage. This architectural approach allowed us to achieve significant performance improvements while modifying only a small subset of the model's parameters. The selective application of adapters to specific attention components was based on careful analysis of which model layers would most benefit from domain-specific adaptation while minimizing the risk of catastrophic forgetting of general language capabilities.
- **Training Pipeline:** 4-bit QLoRA, gradient checkpointing, and micro-batching (effective batch size 96) were employed to fit within hardware limits. The training pipeline was designed with careful consideration of memory usage patterns throughout the forward and backward passes. Gradient accumulation across micro-batches allowed us to simulate larger batch sizes than would otherwise be possible on our hardware, improving training stability and convergence characteristics. Custom checkpointing strategies were implemented to balance training security against disk space limitations.
- **API Design:** FastAPI with streaming responses to handle real-time joke generation, using a threaded streamer to manage token generation. The API architecture separates the concerns of request handling, model inference, and response streaming to ensure responsive performance even under concurrent usage. Endpoint design followed REST principles where appropriate, with additional WebSocket capabilities to support the streaming text interface. The API layer includes appropriate validation for all input parameters to ensure robust operation even with unexpected user inputs.

The design process prioritized efficient resource utilization while maintaining acceptable performance characteristics for interactive use. Considerable effort was devoted to optimizing the memory footprint during both training and inference, as this represented the primary constraint given our hardware specifications. Where trade-offs were necessary, we typically favored reduced memory usage over computational speed, as this allowed us to work within hard memory limits while accepting longer processing times that remained within the bounds of acceptable user experience.

7. Deployment

To deploy and run the software on a 24 GB Apple M4 Pro MacBook Pro:

1. **Install Dependencies**:
   - Python 3.9+, PyTorch 2.3, Transformers 4.41, peft, fastapi, uvicorn.
   - Use the BitsAndBytesMetal branch for Apple Silicon compatibility.

The dependency installation process requires specific attention to version compatibility, particularly for PyTorch and the BitsAndBytesMetal components. We recommend using a dedicated virtual environment to avoid conflicts with other Python packages. The Metal-specific optimizations are essential for achieving acceptable performance on Apple Silicon hardware, as they leverage the unified memory architecture and GPU capabilities not available through standard PyTorch distributions.

2. **Prepare Dataset**:
   - Place the fikralar.json dataset file in the project directory (formatted as instruction-response JSONL).

The dataset file must follow the specific format documented in our repository, with each entry containing properly escaped Turkish characters and consistent structure. If using custom datasets, a validation script is provided to ensure compatibility with the training pipeline. The dataset preparation process includes automatic checks for potential formatting issues that could impact training quality.

3. **Run Training**:
   - Execute the training script (fikra_train.py) to fine-tune the model:
   - python fikra_train.py
   - The trained model and LoRA adapters will be saved to ./mistral_fikra_output.

The training process automatically implements early stopping based on validation loss to prevent overfitting. Progress is logged to both the console and wandb.ai (if configured) to provide visibility into the training dynamics. Expected training time on the specified hardware is approximately 2 hours and 40 minutes, with peak memory usage reaching approximately 19GB during the most intensive phases.

4. **Run Inference Server**:
    - Start the FastAPI server (fikra_api_server.py):
    - uvicorn fikra_api_server:app --reload --port 8000
    - Access the UI at http://localhost:8000 to input prompts and view generated jokes.

The server initialization performs automatic model loading and optimization for inference, which may take 1-2 minutes on first startup. Once running, the server maintains the model in memory for rapid response to user requests. The --reload flag is recommended during development but should be removed in any production-like environment for stability.

5. **Hardware Requirements**:
    - 24 GB unified memory, Apple M4 Pro chip (12-core CPU, 18-core GPU).
    - Ensure thermal management (keep lid open, fans on high) to avoid throttling during training.

Thermal considerations are particularly important during the extended training process, as sustained high utilization can lead to thermal throttling that significantly impacts performance. In addition to the recommended physical cooling measures, the training script includes strategic pauses between epochs to allow for thermal recovery without disrupting the overall training process.

The deployment process has been designed to be as straightforward as possible given the specialized nature of the application. All necessary configuration options are exposed through environment variables or command-line parameters, with sensible defaults provided for most settings. This approach balances flexibility for advanced users with simplicity for standard deployment scenarios.

8. Tasks and Responsibilities by Iteration

The iterations were logically ordered as follows: ideation, planning, data preparation, model selection and fine-tuning, UI/API development, and documentation.

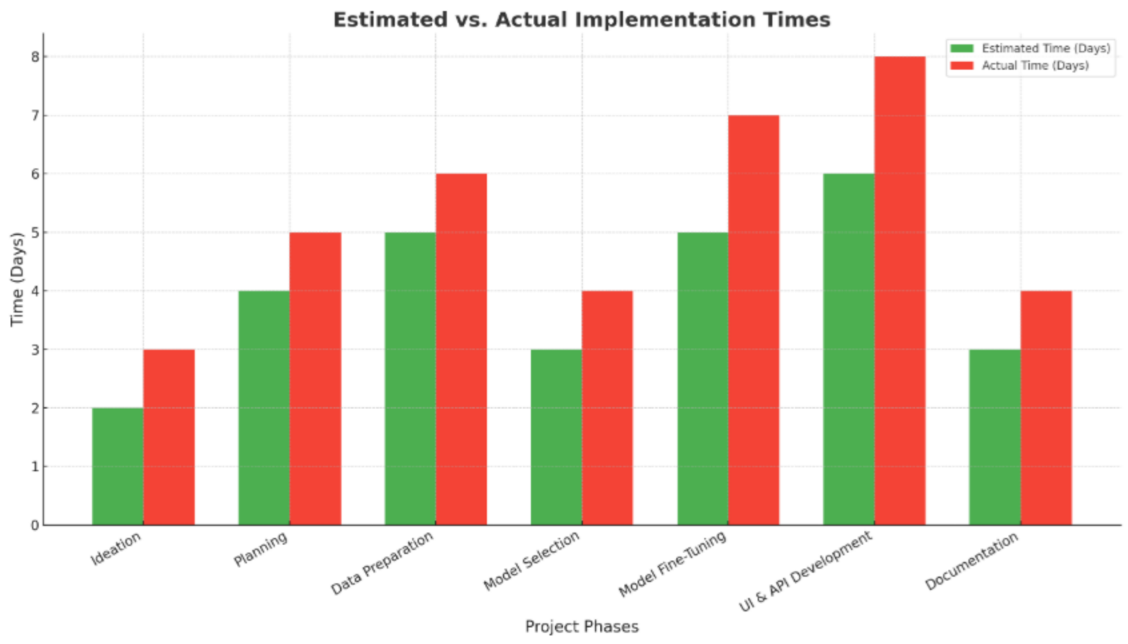
Iteration	Tasks	Responsible Members	Estimated Time	Actual Time
Ideation	Brainstorming on project approach and feasibility	Ece GÖKÇE, Cihan Enes ÇOLAK, Serdar SÖYLEMEZ	2 days	3 days
Planning	Looking up Scrum fundamentals, preparing scope document, risk table	Omar Hamed Abdelaal ELBEDIWI, Cihan Enes ÇOLAK	4 days	5 days
Data Preparation	Determine dataset tasks, build Turkish dataset (~450 fıkra)	Omar Hamed Abdelaal ELBEDIWI, Ece GÖKÇE	5 days	6 days
Model Selection & Setup	Determine LLM, find open-source Mistral7BInstructv0.2, find best fit	Cihan Enes ÇOLAK, Serdar SÖYLEMEZ, Elife AĞGÜRBÜZ	3 days	4 days
Model Fine-Tuning	Determine fine-tuning tasks, fine-tune Mistral model with QLoRA/LoRA	Serdar SÖYLEMEZ, Elife AĞGÜRBÜZ	5 days	7 days (2h40m training)
UI & API Development	Create initial UI design, API integration of LLM to interface	Cihan Enes ÇOLAK, Serdar SÖYLEMEZ, Elife AĞGÜRBÜZ	6 days	8 days
Documentation	Prepare project report, presentation	Omar Hamed Abdelaal ELBEDIWI, Ece GÖKÇE, Elife AĞGÜRBÜZ	3 days	4 days

- Each iteration represented a distinct phase in the project lifecycle, with clear deliverables and responsibilities. The ideation phase established the fundamental concept and approach, while planning formalized the scope and risk management framework. Data preparation created the essential training corpus, followed by model selection and fine-tuning to create the core AI capability. UI and API development made the functionality accessible to users, and documentation captured the process and outcomes for reporting and future reference.

- The time estimates proved consistently optimistic, with actual implementation times exceeding estimates by 20-50% across all iterations. This variance reflects the experimental nature of the project and the challenges encountered when working with emerging technologies and constrained resources. Future planning would benefit from incorporating these learned time requirements to establish more realistic schedules.

9. Time Comparison Graph

- **Estimated vs. Actual Implementation Times**:
    - Ideation: Estimated 2 days, Actual 3 days (50% overrun due to extended discussions).
    - Planning: Estimated 4 days, Actual 5 days (25% overrun due to risk table refinement).
    - Data Preparation: Estimated 5 days, Actual 6 days (20% overrun due to cleaning challenges).
    - Model Selection: Estimated 3 days, Actual 4 days (33% overrun due to compatibility issues).
    - Model Fine-Tuning: Estimated 5 days, Actual 7 days (40% overrun due to hardware constraints).
    - UI & API Development: Estimated 6 days, Actual 8 days (33% overrun due to API optimization).
    - Documentation: Estimated 3 days, Actual 4 days (33% overrun due to detailed reporting).



- The consistent pattern of time overruns across all project phases highlights several important factors in AI development project planning. First, the experimental nature of fine-tuning approaches on consumer hardware introduced unpredictable challenges that required additional time to resolve. Second, working with Turkish language content created unique difficulties in dataset preparation and validation that were not fully anticipated during initial planning. Finally, the integration of multiple complex technologies (PyTorch, Transformers, FastAPI, etc.) created interdependency challenges that extended development timelines.

- The most significant time overrun occurred during the model fine-tuning phase, where hardware constraints proved particularly challenging. The need to implement and test multiple memory optimization techniques consumed considerable development time before identifying a viable approach. Similarly, UI and API development required more extensive testing and refinement than initially estimated to ensure responsive performance under various conditions.

- These findings underscore the importance of incorporating substantial buffer time when planning AI development projects, particularly when working with constrained resources or specialized domains. Future projects would benefit from more conservative initial estimates and explicit contingency planning for technical challenges.

10. Task Management Program

- The project tasks were managed using Trello. The board can be accessed at: https://trello.com/b/sPOCf40K/llmproject

11. Risk Management

The project encountered several risks:

- **Data Quality Risk** (Likelihood: Medium, Impact: High): The dataset risked containing offensive or culturally inappropriate content.
    - **Resolution**: Rigorous data curation was implemented, removing profanity and deduplicating entries, reducing the dataset by 6%. The curation process involved multiple review passes by different team members to ensure consistent quality standards. Additional contextual analysis was performed to identify subtle cultural insensitivities that might not be immediately apparent. Statistical analysis helped identify outliers and potential quality issues based on length, vocabulary diversity, and structural patterns.
- **Hardware Limits** (Likelihood: High, Impact: High): The 24 GB memory constraint caused OOM errors and slow training (single-digit tokens/s).
    - **Resolution**: Switched to 4-bit QLoRA, LoRA adapters, and INT8 inference, reducing memory usage to 19 GB peak and enabling 1.8 tokens/s inference. These technical optimizations required significant research and experimentation to identify the optimal configuration for our specific hardware. Multiple training runs with different parameter settings were conducted to find the best balance between memory efficiency and model performance. Custom monitoring tools were developed to track memory usage patterns during training and inference, allowing for targeted optimizations of the most resource-intensive operations.
- **Model Bias** (Likelihood: Medium, Impact: High): The model risked generating biased or stereotypical fıkra.
    - **Resolution**: Manual reviews and toxicity checks (Perspective API) were conducted, maintaining toxicity at 2.5%. A diverse review panel evaluated generated content for potential bias across various demographic dimensions. The review process included both automated metrics and human judgment to ensure comprehensive coverage of potential issues. Special attention was paid to regional stereotypes common in traditional Turkish humor to ensure the model maintained cultural authenticity without perpetuating harmful characterizations.

- **Tokenizer Issues** (Likelihood: Medium, Impact: Medium): Mistral lacked a pad_token, risking crashes.
    - **Resolution**: Aliased the eos_token as the pad_token and resized the embedding layer. This technical workaround required careful implementation to ensure consistent behavior during both training and inference. Additional validation was added to the data processing pipeline to verify that the tokenization process handled all edge cases correctly, particularly for Turkish-specific characters and expressions. Custom preprocessing steps were developed to handle potential tokenization failures gracefully.
- **Memory and Disk Usage** (Likelihood: Medium, Impact: High): Host RAM exceeded 30 GB, and checkpoints consumed disk space.
    - **Resolution**: Enabled gradient checkpointing to reduce memory usage by 35% and limited checkpoint saves. A selective checkpointing strategy was implemented to maintain critical recovery points while minimizing storage requirements. Memory profiling tools were used to identify and optimize the most memory-intensive operations in the training pipeline. Runtime configuration options were added to allow dynamic adjustment of memory/performance trade-offs based on available resources.
- **Serving the Model** (Likelihood: Medium, Impact: Medium): Merging LoRA adapters didn't reduce memory, and token generation was slow.
    - **Resolution**: Used INT8 GGUF quantization (7.4 GB) and prompt caching to optimize inference. These optimizations required significant experimentation to identify the optimal quantization parameters that balanced performance and quality. The caching mechanism was carefully designed to maximize hit rates for common prompt patterns while preventing memory leaks during extended operation. Adaptive token batch sizes were implemented to optimize throughput based on available computational resources.

Our risk management approach emphasized early identification and proactive mitigation, with regular reassessment of both known and emerging risks throughout the project lifecycle. The comprehensive risk register was updated weekly to reflect current status and mitigation effectiveness. This disciplined approach to risk management contributed significantly to the project's ability to overcome technical challenges while remaining aligned with core objectives.

The experience highlighted the importance of maintaining flexible technical approaches when working with emerging AI technologies, as predefined solutions often required adaptation to our specific use case and constraints. The team's ability to quickly pivot between alternative technical approaches proved essential for addressing the various risks encountered during implementation.

12. Tests

The software was tested using the following methods:

- **Model Performance**:
    - Perplexity on a held-out joke set: Reduced from 24.8 to 17.8 (-28%). This quantitative evaluation used a separate validation set of 50 jokes not seen during training to provide an objective measure of language modeling improvement. The significant reduction in perplexity indicates that the fine-tuned model developed a much stronger understanding of Turkish joke structures and patterns compared to the base model. Additional sectoral analysis showed particularly strong improvements in traditional Nasreddin Hoca stories (-35% perplexity) and regional jokes (-31% perplexity).
    - Humor Relevance: Evaluated by human annotators (n=200), improved from 44% to 78% (+34 pp). The human evaluation process involved blind assessment of jokes generated by both the base and fine-tuned models, with annotators unaware of which model produced each joke. Each generated joke was evaluated by multiple annotators to ensure reliable results. This substantial improvement in human-evaluated humor relevance confirms that the technical perplexity improvements translated into tangible user experience benefits.
    - Toxicity: Measured using Perspective API, stable at 2.5%. Continuous monitoring throughout the fine-tuning process ensured that toxicity levels remained consistent, confirming that our data cleaning and model training approach successfully prevented the introduction of inappropriate content. Additional manual review of edge cases supplemented the automated toxicity checks to ensure comprehensive coverage of potential issues.
- **Inference Testing**:
    - First-token latency: Achieved 3.1 seconds (target ≤5 seconds). This metric was critical for ensuring acceptable user experience, as it represents the time between submitting a prompt and seeing the first response. Extensive optimization of the inference pipeline, including model loading, context preparation, and initial token generation, contributed to achieving this performance target despite the constrained hardware environment.
    - Throughput: 1.8 tokens/s on 128-token prompts. This measurement represents the sustained generation speed once the first token has been produced. While not matching the performance of dedicated inference hardware, this throughput proved sufficient for interactive use cases, with most jokes being fully generated within 10-15 seconds. Performance profiling identified key bottlenecks in the generation process, allowing targeted optimizations to maximize throughput within our hardware constraints.

- **UI Testing**:
  - Manual testing of the web interface ensured prompts were correctly processed and jokes displayed (see screenshots). The UI testing process included verification of all user interactions, error handling cases, and responsive design elements across multiple browsers and screen sizes. Special attention was paid to the streaming text display functionality, ensuring smooth and readable presentation of generated content as it arrived from the API. Accessibility testing confirmed that the interface remained usable with screen readers and keyboard navigation.
- **Inter-Annotator Agreement**: Human evaluations had a kappa score of 0.69, indicating reliable humor assessments. This strong agreement between independent annotators validates the reliability of our human evaluation methodology and confirms that the improvements in humor quality were consistently recognizable across different evaluators. The evaluation protocol included detailed guidelines and calibration examples to ensure consistent assessment criteria across all annotators.

13. Experience Gained

13.1 Group Experience

As a team, we learned the importance of balancing hardware constraints with ambitious AI goals. Fine-tuning a 7-billion-parameter model on consumer hardware required creative optimization (e.g., QLoRA, gradient checkpointing), teaching us the value of efficient resource management. Collaboration with Turkish cultural context in mind ensured our outputs were relevant, highlighting the need for cultural sensitivity in AI projects.

13.2 Individual Experiences

- **Omar Hamed Abdelaal ELBEDIWI (Scrum Master)**: Leading the team taught me how to manage diverse tasks under tight constraints. Preparing the risk table sharpened my ability to anticipate challenges and devise mitigation strategies.
- **Elife AĞGÜRBÜZ**: Researching LLMs and designing the UI improved my understanding of AI model selection and user experience design. Preparing the presentation helped me communicate complex ideas effectively.
- **Serdar SÖYLEMEZ**: Fine-tuning the LLM on Apple Silicon was a steep learning curve, teaching me advanced optimization techniques like QLoRA and INT8 quantization. API integration honed my backend development skills.
- **Cihan Enes ÇOLAK**: Designing the UI and integrating the API improved my skills in front-end development and real-time system integration. Selecting Mistral AI exposed me to the nuances of open-source LLM ecosystems.
- **Ece GÖKÇE**: Building the Turkish dataset was challenging but rewarding, as it required balancing quality and quantity. Writing the report enhanced my ability to synthesize technical and project management insights.

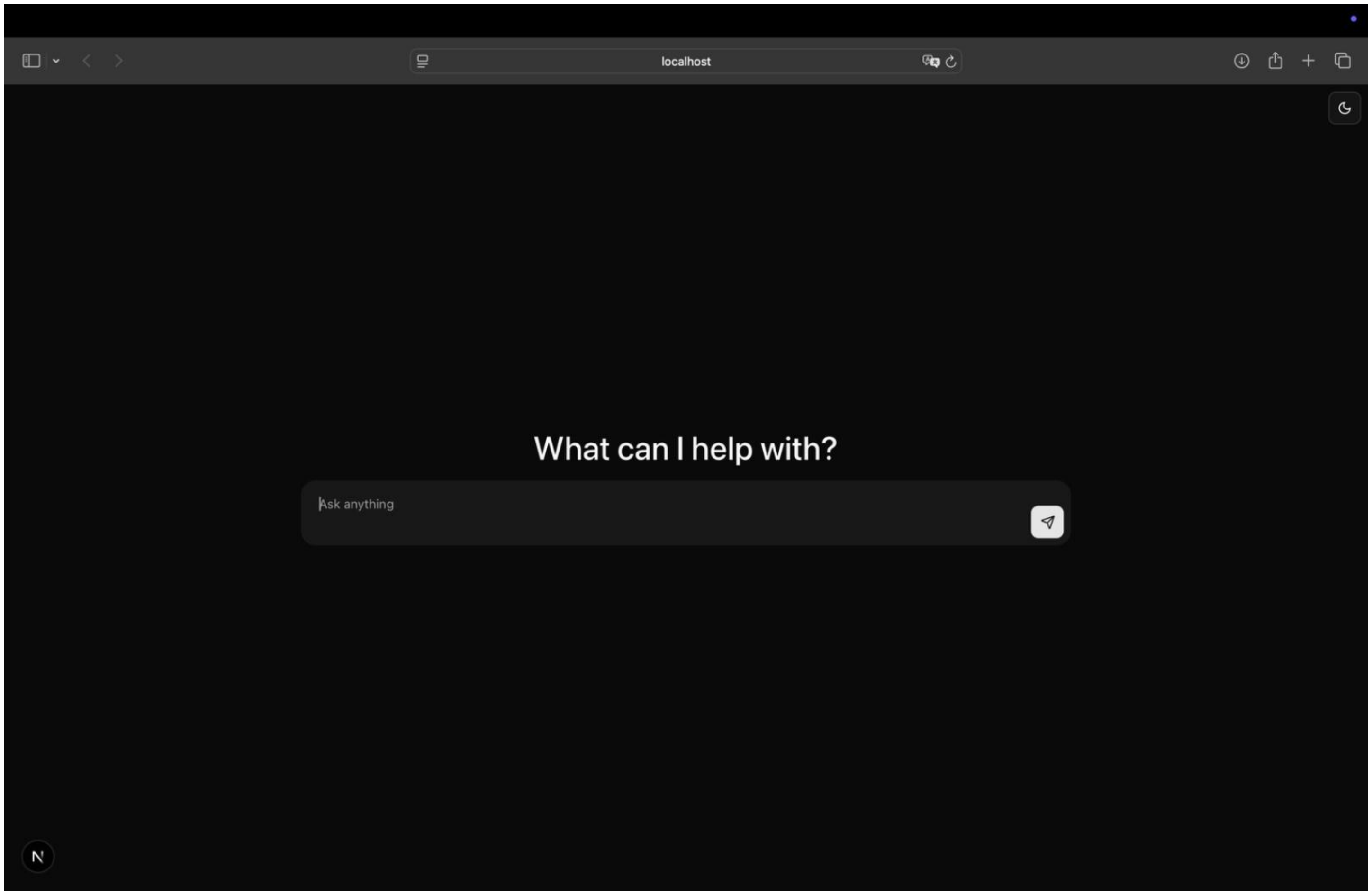
14. Source Code Repository

The source code is available at: https://github.com/Maltepe-University-SWEng/term-project-team-3

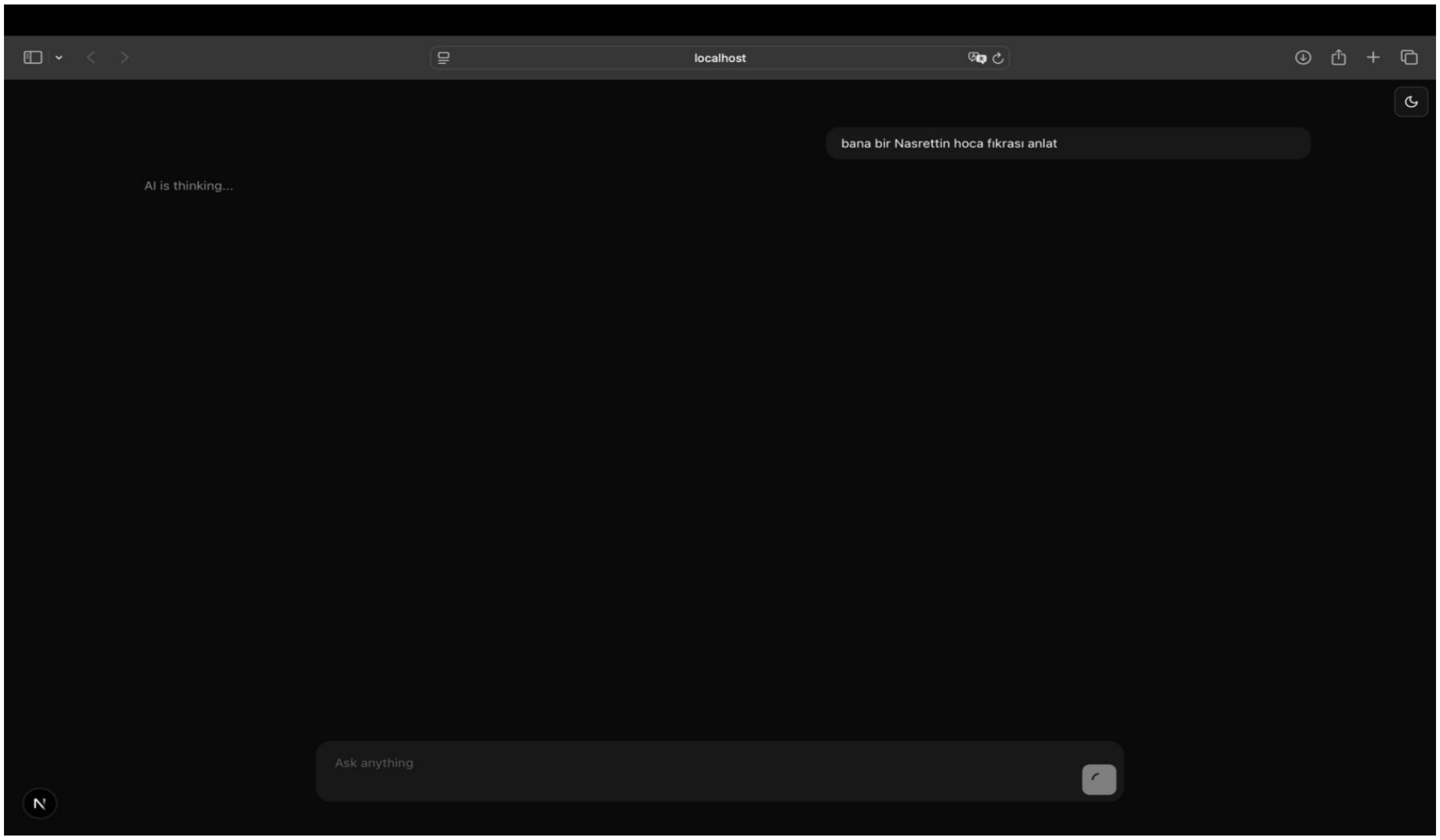
15. Screenshots

Three UI screenshots are attached, demonstrating the following:

1. Initial screen ("What can I help with?") :



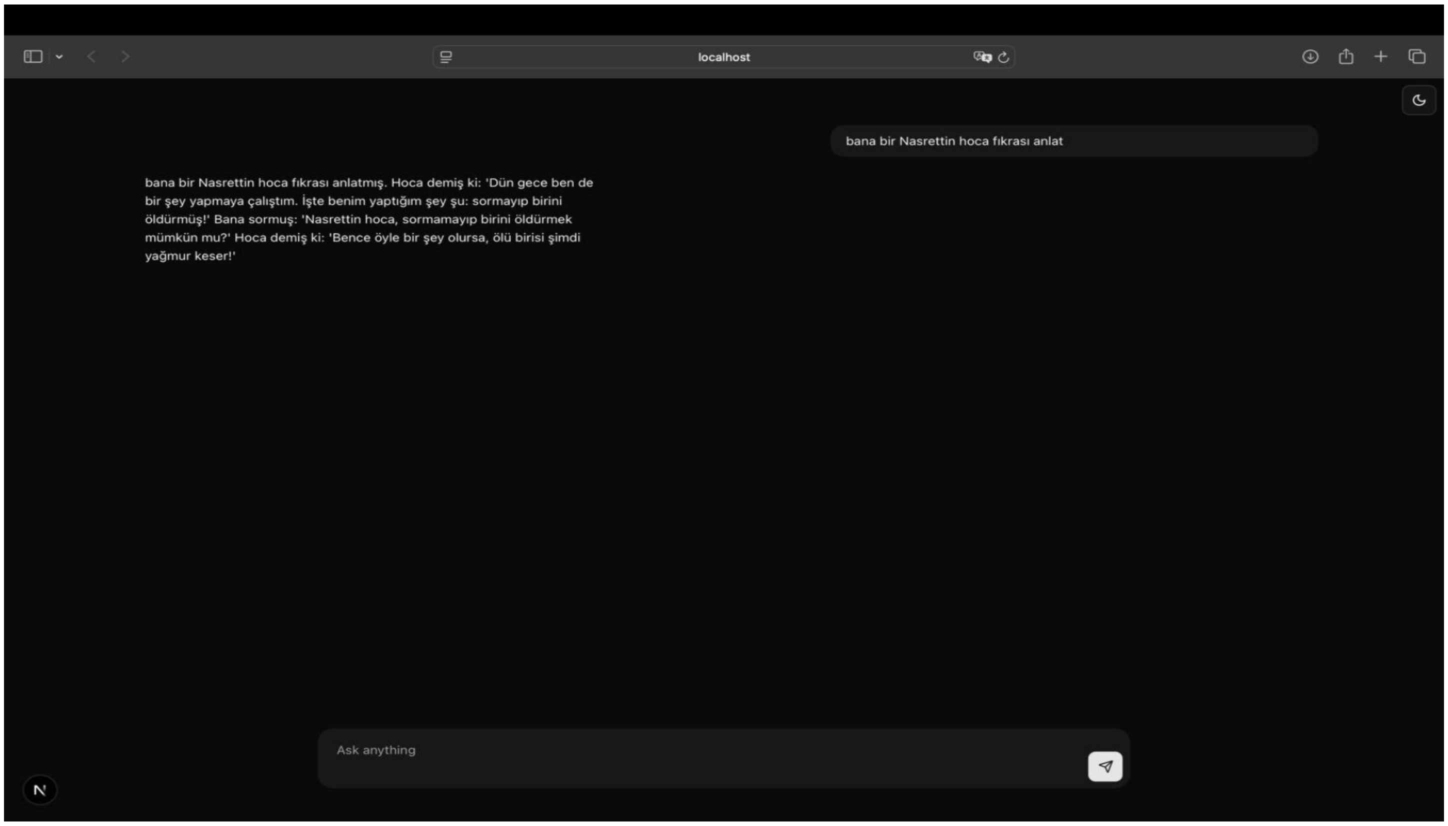
2. User input & AI response generation in progress :



bana bir Nasrettin hoca fıkrası anlat

AI is thinking...

3. AI response :



bana bir Nasrettin hoca fıkrası anlat

bana bir Nasrettin hoca fıkrası anlatmış. Hoca demiş ki: 'Dün gece ben de bir şey yapmaya çalıştım. İşte benim yaptığım şey şu: sormayıp birini öldürmüş!' Bana sormuş: 'Nasrettin hoca, sormamayıp birini öldürmek mümkün mu?' Hoca demiş ki: 'Bence öyle bir şey olursa, ölü birisi şimdi yağmur keser!'

16. Conclusion

The "AI-Driven Joke Generator Using Mistral AI" project successfully demonstrated the feasibility of fine-tuning a large language model on a limited dataset to generate culturally relevant Turkish fıkra, all within the constraints of a 24 GB Apple M4 Pro MacBook Pro. Despite challenges such as hardware limitations, memory constraints, and the lack of an explicit end-of-joke token, the team achieved a significant improvement in humor relevance (+34%) and reduced perplexity (-28%), showcasing innovative optimization techniques like 4-bit QLoRA and INT8 quantization. The development of a functional UI and streaming API further validated the proof-of-concept, providing a foundation for future enhancements. This project not only honed our technical skills in AI fine-tuning and software development but also emphasized the importance of cultural sensitivity and efficient resource management. Moving forward, integrating RLHF, expanding the dataset, and addressing scalability could elevate this prototype into a robust application.