AI Project

QuestCrafter: Build an AI "Dungeon Master" for RPG Quests

MSc1 — AI

Project Overview

QuestCrafter is a small text-generation system that creates **short fantasy quests** from a user prompt, e.g.:

- `"Create a level-3 quest in a desert city with a betrayal twist."`

Main idea: start from an existing small language model, build a baseline, then fine-tune it with a curated dataset, and deliver a simple demo.

You will practice:

- Data preprocessing for instruction-style generation (prompt → completion)
- Baseline vs fine-tuned model comparison
- Lightweight training with **PyTorch** (via **Hugging Face** `transformers`)
- Evaluation (automatic + human rubric)
- Demo app (Streamlit or Gradio)

Learning Objectives

By the end of week 5, you should be able to:

1. Prepare a text dataset in a clean supervised format (JSONL/CSV).
2. Run inference with a pretrained model and analyze typical failures.
3. Fine-tune a small model with **PyTorch** (optionally with LoRA/PEFT).
4. Evaluate generations using a fixed test set + a human scoring rubric.
5. Deliver a reproducible mini-product: repo + report + demo.

Dataset (Choose One)

Pick **one** dataset for training:

- **WritingPrompts (Kaggle)**: prompt/story pairs (excellent for controlled prompting).
- **TinyStories (Hugging Face)**: short, clean stories (easy and fast to train/evaluate).
- **Reddit Jokes (Kaggle)** (optional alternative): style control for humor.

Recommended for beginners: WritingPrompts or TinyStories.

Target training format:

- `instruction/prompt` $\rightarrow$ `response` (quest text)
- Optional metadata: `level`, `setting`, `tone`, `length`

Modeling Approach (with PyTorch)

Framework: PyTorch + Hugging Face `transformers` (PyTorch backend)

Suggested small models (CPU/GPU friendly):

- Decoder-only: `distilgpt2` or `gpt2`
- Encoder-decoder: `t5-small` (instruction-friendly)

Training options:

- **Full fine-tuning** (simple, but heavier)
- **LoRA / PEFT** (recommended: faster, cheaper, easy to compare)

Implementation tip: start with `distilgpt2` for quick results.

End-to-End Pipeline

1. **Data** download $\rightarrow$ cleaning $\rightarrow$ formatting (JSONL).
2. **Baseline inference:** generate quests with pretrained model (no training).
3. **Fine-tuning:** train on your prompt $\rightarrow$ quest pairs.
4. **Control:** add fields like `LEVEL`, `SETTING`, `TONE` in prompts.
5. **Evaluation:** automatic metrics + human rubric + example comparisons.
6. **Demo:** small app for interactive generation.

Timetable (5 Weeks)

Week	**Main activities and outputs**
W1	Dataset selection + exploration; cleaning and formatting; define quest template fields; build train/val/test splits.
W2	Baseline generation (pretrained model); create fixed test prompts (50+); define evaluation rubric; first failure analysis.
W3	Fine-tune one small model in **PyTorch** (optionally LoRA); track training curves; save checkpoints; sample outputs after training.
W4	Add controllability (level/setting/tone/length); add safety filters; structured evaluation: baseline vs tuned; robustness tests.
W5	Build demo (Streamlit/Gradio); finalize report (4–8 pages); create final repo (scripts + README); final presentation.

Timetable Matrix (What Happens When)

Task	**W1**	**W2**	**W3**	**W4**	**W5**
Dataset pipeline (clean + format)	✓				
Baseline inference + analysis		✓			
Fine-tuning (PyTorch)			✓		
Control + safety + robustness				✓	
Evaluation + reporting		✓	✓	✓	✓
Demo app + packaging					✓

Note: ✓indicates the main focus of the week (some tasks continue across weeks).

# Mandatory Tasks (Required)

1. **Dataset pipeline**
   - Download dataset, clean, format to JSONL/CSV: `prompt` → `response`
   - Split into train/val/test (e.g., 80/10/10)
2. **Baseline generation (no training)**
   - Generate outputs using a pretrained model
   - Document typical issues (repetition, off-topic, too long, incoherence)
3. **Fine-tuning with PyTorch**
   - Train at least one model (`distilgpt2`/`gpt2`/`t5-small`)
   - Provide config (epochs, LR, batch size, max length)
4. **Evaluation**
   - Fixed test set of **50+ prompts**
   - Automatic metric: validation loss / perplexity + one diversity metric (e.g., Distinct-n)
   - Human rubric: coherence, creativity, prompt-faithfulness (1–5)
5. **Final deliverable**
   - Repo + README + report (4–8 pages) + demo app

# Optional Tasks (Choose Any)

- **LoRA/PEFT** fine-tuning and compare vs full fine-tuning
- Add **control tokens** like `<LEVEL=3><TONE=dark><SETTING=forest>`
- **RAG add-on:** retrieve from a small “lore book” before generation
- Multi-output: quest + NPC dialogue + item description + reward summary
- Automatic quality filters (repetition detection, length control, regeneration)
- Class leaderboard: compare models with the same test prompts + rubric
- Write a short **model card** (limitations, bias, safety, intended use)

Evaluation (Baseline vs Fine-Tuned)

Fixed test prompts: 50+ prompts covering:

- different settings (forest, desert, cyberpunk, medieval city)
- different levels (1–10) and lengths (short/medium)
- different tones (epic, humorous, dark)

Human rubric (1–5 each):

- Coherence (logical story flow)
- Prompt-faithfulness (respects constraints)
- Creativity (interesting elements, not generic)

Automatic: validation loss/perplexity + Distinct-n (diversity).

Final Deliverables

1) Git repository

- `data/` (or script to download + preprocess)
- `preprocess.py` (or notebook)
- `train.py` (PyTorch training via **Hugging Face** `Trainer` or custom loop)
- `evaluate.py` (metrics + saving generations)
- demo$_a pp.py(Streamlit/Gradio)$

- `README.md (how to run + results summary)`

2) Report (4–8 pages)

- Dataset + formatting, baseline, fine-tuning setup, evaluation protocol, results

3) Presentation

- demo + baseline vs tuned examples + what you learned

Suggested Grading Breakdown (Example)

Criterion	**Weight**
Dataset pipeline quality + reproducibility	20%
Baseline + analysis (failure cases, examples)	15%
Fine-tuning (correctness, experiment tracking)	25%
Evaluation protocol (test set + rubric + metrics)	20%
Demo + final packaging (repo + README + usability)	20%

Practical Notes (Compute & Tips)

- Start small: use a subset of the dataset for quick iteration.
- Keep generations short at first (e.g., max 120–200 tokens).
- Track everything: seed, hyperparameters, checkpoints, prompt set.
- Prefer **LoRA/PEFT** if GPU memory is limited.
- Always compare against baseline with the **same** test prompts.

Common failure modes to monitor:

- repetition loops, off-topic drift, ignored constraints, over-long outputs

Conclusion

QuestCrafter is a beginner-friendly, end-to-end generative AI project:

- real dataset + real training in **PyTorch**
- clear mandatory milestones
- optional extensions (LoRA, RAG, control tokens)
- a demo you can show in interviews

Next step: pick the dataset (WritingPrompts or TinyStories) and build W1 pipeline.