MAE301 Course Project Guideline (v1.0)

---

0. Overview

In this team project, you will self-learn Python, GitHub, and basic deep learning knowledge to follow a guided "build GPT from scratch" exercise to implement, train, and analyze a nanoGPT-style transformer language model. You will also develop an AI startup concept (not limited to engineering topics) and build a Minimum Viable Product (MVP) that uses the nanoGPT model or other models you learn through the class.

You will deliver work in three phases:

- **Mar 16** – Idea Pitch (1-page + 1 short video)
- **Apr 13** – nanoGPT Implementation & Report
- **Apr 27** – MVP Implementation & Report
- Top projects (based on peer and instructor review) present on **Apr 29 & May 1**

All deliverables must be in markdown format and tracked in your team GitHub repository.

---

1. Project Timeline & Milestones

Phase 1 – Idea Pitch (Due: Mar 16)

- Submit 1-page idea markdown and a 5-minute video in your GitHub repo.
- In-class peer review on Mar 16.
- Top-scoring ideas will present on **Mar 18**.

Phase 2 – nanoGPT Build (Due: Apr 13)

- Complete a basic GPT-style language model, using Karpathy's material as reference.
- Demonstrate training runs, validation losses, sample generations.
- Submit code + a brief nanoGPT report in GitHub.

Phase 3 – MVP Build (Due: Apr 27)

- Develop an **MVP** attempt for your AI startup idea.
- Submit updated report in GitHub and code + demo instructions (demo video optional).
- In-class peer review on Apr 27.
- Top-scoring MVPs present on **Apr 29 & May 1**.

---

2. Team Formation & Tools

- **Team size**: ~6 students per team, randomly assigned after enrollment finalized.
- **Required**: Python, Git + GitHub, IDE (e.g., VS code or Cursor), access to AI assistant (e.g., GPT thinking/pro, Gemini pro, Claude)
- **Optional**: Access to GPU clusters (e.g., Google Colab, ASU Research Computing cluster)
- **GitHub repository naming**: mae301-2026spring-<team-name>. Example: mae301-2026spring-attentionisallyouneed
- Recommended directory structure:
  - `/proposal/ # idea pitch markdown + video link`
  - `/nanogpt/ # nanoGPT code, configs, logs, report`
  - `/mvp/ # MVP code, data notes, demo, report`
  - `/docs/ # any extra documentation`

---

3. Self-Learning Resources (Python, GitHub, nanoGPT)

3.1 Python for Beginners

Most of you are new to Python. The goal is not to master everything, but to be comfortable with:

- Variables, control flow, functions
- Lists / dicts
- Basic file I/O
- Using pip, virtual environments, and running scripts
- Numpy-style tensor operations at a basic level

Suggested resources:

- Official Python "getting started" docs
- Real Python "Python Basics" tutorial collection
- A clean "absolute beginner" text tutorial, e.g. GeeksforGeeks intro

Minimum expectation: By the end of Week 2 of the project, you should be able to:

- Write and run a .py script from the command line or an IDE.
- Use pip install to install packages.
- Read and modify simple Python code.

3.2 Git & GitHub Basics

You will hand in all work via GitHub. Every team member should know how to:

- Clone a repo

- Create a branch
- Commit and push changes
- Open a pull request

Recommended starter: GitHub's official **"Hello World"** tutorial

Minimum expectation: by the end of Week 2, each student has:

- A GitHub account.
- Completed the "Hello World" GitHub exercise.
- Successfully cloned, edited, and pushed to the team repo.

3.3 GPT-from-scratch Resources

We will anchor on Andrej Karpathy's materials:

- Karpathy "Let's build GPT: from scratch, in code, spelled out"
- Companion repo for building nanoGPT step-by-step (build-nanogpt)
- Neural Networks: Zero to Hero course (repo)

You are not required to reproduce GPT-2 scale. We will focus on small character-level or word-level language models (e.g., training on tiny Shakespeare) that can run on a laptop.

---

4. Phase 1 – Idea Pitch (Due Mar 16)

4.1 Deliverable: 1-Page Markdown Proposal

Create /proposal/idea.md in your repo with:

1. **Project Title**
2. **Team Members** (names, emails)
3. **Problem Statement**
    - Who is the user?
    - What problem or pain point do they experience today?
4. **Why Now?**
    - Why does this problem matter in the next 3–5 years?
    - What changed (technology, regulations, culture) that makes this possible now?
5. **Proposed AI-Powered Solution**
    - What does your product do for the user?
    - Where does AI/ML add unique value vs simple rules / heuristics?
6. **Initial Technical Concept**
    - What data would you need (or already have)?
    - What model(s) might you use (e.g., GPT-style text model, classifier, recommender, vision model)?

    - How could your nanoGPT work feed into this (e.g., custom generative component, fine-tuning, evaluation)?
7. **Scope for MVP**
    - What can you realistically build in ~6 weeks?
    - Define a very concrete v1 feature: "A user can ___ and our system returns ___."
8. **Risks and Open Questions**
    - Top 3 unknowns (e.g., data availability, evaluation, user adoption).
9. **Planned Data Sources**
    - E.g., Kaggle, Hugging Face Datasets, public APIs, synthetic data.

4.2 Deliverable: 5-Minute Video Pitch

Include in your proposal a YouTube video pitch. Suggested structure:

1. **Story**: a short, concrete user scenario ("Imagine you are a ___ trying to ___").
2. **Solution**: show how your AI product fits into the scenario.
3. **Why AI?**: what makes this solution uniquely enabled by AI?
4. **Feasibility for this course**: what you'll actually build as MVP.

Screen-sharing slides is acceptable. Graded mainly based on idea quality and clarity.

4.3 Idea Generation & Self-Validation

I strongly recommend you spend time on idea quality before writing code:

- Work on problems you or your friends actually have; this ensures the problem is real.
- Avoid "cool tech in search of a problem." Start from users and pain, not solutions.
- Talk to real or proxy users early and often; qualitative interviews beat speculation.

Practical YC techniques:

1. **Problem Inventory**: Each teammate lists 10 recurring annoyances in their own life (school, hobbies, family, part-time jobs). Compare lists for overlap.
2. **"Hair-on-fire" test**: Ask: "Is this a 'nice to have' or 'I would pay today to fix this' problem?"
3. **"Why now?" test**: What changed (new open models, cheaper GPUs, regulatory change, remote work, etc.) that makes your idea newly possible or significantly better now?
4. **User Conversations**: Talk to at least **3 potential users**. Ask open-ended questions:
    - "Walk me through how you do X today."
    - "What's the most annoying part?"
    - "What happens if you just don't solve this?"

4.4 Peer-Review Rubric (Tentative)

You will review peers' ideas on a 1–5 scale (1=poor, 5=excellent) along these dimensions:

1. Clarity of Problem & User
2. Importance / Impact of Problem
3. Fit Between Problem and AI Solution
4. Technical Feasibility for a Student MVP
5. Originality / Insight

The average peer score plus instructor review will determine which teams present on **Mar 18**.

---

5. Phase 2 – nanoGPT Build (Due Apr 13)

5.1 Minimum Technical Requirements

By Apr 13, your team must:

1. Implement (or cleanly adapt) a **GPT-style language model** in PyTorch or similar, following Karpathy's "build GPT from scratch" lecture and repos.
2. Train on at least one text dataset (e.g., Tiny Shakespeare or another small corpus).
3. Run at least three experimental configurations varying architecture, such as:
    - Number of layers
    - Number of attention heads
    - Embedding dimension
    - Context length
    - Dropout
4. Log and report:
    - Training & validation loss curves
    - Final validation loss for each config
    - Example generated text samples for each config

You may start from Karpathy's teaching repo and modify it; you are graded on understanding and experimentation, not on writing every line from memory.

5.2 Deliverables

In /nanogpt/:

1. **Code**
    - Training script(s) (e.g., train.py, model.py).
    - Config files (YAML/JSON or simple Python dicts).
    - requirements.txt.
    - Clear README.md explaining: how to install dependencies; how to run a small training job; expected runtime (e.g., on CPU vs GPU).
2. **Data**

    - Either include small text files in the repo (if allowed by license) or document clearly how to download (e.g., Shakespeare from public domain).
    - Document tokenization strategy.
3. **Logs / Plots**
    - Loss curves as .png or a Jupyter notebook.
    - A simple results.md or section in the report summarizing experiments.
4. **nanoGPT Report**
    - Place in /nanogpt/report.md.
    - Suggested structure:

        - **Objective**: What did you attempt (e.g., character-level GPT on Shakespeare)?
        - **Model Architecture**: Describe embedding layer, positional encoding, self-attention block, feedforward, softmax, etc.
        - **Training Setup**: Dataset description, train/val split, batch size, learning rate, optimizer, number of steps.
        - **Experiments**: Table of configs and results (layers, heads, embed dim, context length, dropout, params, training time, best val loss)
        - **Qualitative Samples**: Generated text for at least two configs; evaluate quality.
        - **Analysis & Reflection**: What did you learn about: over/under-fitting; capacity vs data size; stability of training. How will this inform your MVP model design?

5. **Tag a Release**
    - Create a GitHub release (e.g., v0.1-nanogpt) for the nanoGPT milestone.

---

6. Phase 3 – MVP Development (Due Apr 27)

6.1 Potential Data Sources

You should leverage open datasets if available, e.g.:

- **Kaggle** – thousands of datasets for text, tabular, images, etc.
- **Hugging Face Datasets** – many NLP and multimodal datasets, integrated with the HF ecosystem.
- Other standard sources (UCI ML repo, government open data portals, Wikipedia/Kaggle collaboration datasets, etc.).

When using external data, you must respect licensing/terms of use and document data origin and any preprocessing.

6.2 Model Hosting & Sharing

If you wish to share your model publicly or build a demo around it: **Hugging Face Model Hub** is the standard place to host model checkpoints and create a "model card" describing your model, data, and limitations.

At a minimum, within the course: have a way for the instructor and peers to **run your model locally** or in Colab; optionally, create a public HF repo and include the link in your MVP report.

6.3 MVP Deliverables

In /mvp/:

1. **MVP Code**
    - Scripts/notebooks for training and inference.
    - If you build a web UI (e.g., Streamlit/Gradio), include entrypoint code and instructions.
    - Clearly separate: data/ (or instructions for downloading data), models/ (checkpoints or instructions), src/ (code)
2. **Demo Instructions**
    - README.md with: How to set up environment; how to run a minimal demo, e.g., "Run python demo.py then open http://localhost:8501"; any required API keys (if using external APIs) and where to put them.
3. **MVP Report**
    - Place in /mvp/report.md.
    - Suggested structure:

        1. **Executive Summary**: Problem, solution, and what your MVP actually does.
        2. **User & Use Case**: Clear persona and usage narrative.
        3. **System Design**: High-level architecture diagram (can be ASCII or image in repo); where the model sits, how data flows through.
        4. **Data**: Source(s), size, cleaning, splits.
        5. **Models**: What model(s) you used: Your own nanoGPT variant vs. pre-trained model from elsewhere; any fine-tuning or prompt-engineering strategies.
        6. **Evaluation**: Quantitative metrics where possible; qualitative assessment (e.g., user examples, error analysis).
        7. **Limitations & Risks**: Failure modes, biases, data issues, privacy concerns.
        8. **Next Steps**: What you'd do with 2–3 more months (technical and product).

4. **Optional: MVP Demo Video (3–5 min)**
    - Walk through: The problem; product interface; live or recorded demo of the system solving one realistic example.

6.4 Peer-Review Rubric (Tentative)

Each student will rate other teams' MVPs on 1–5 scale for:

1. **Problem–Solution Fit**
2. **Usefulness / Potential Impact for Real Users**
3. **Technical Depth & Use of AI**
4. **Quality of Evaluation & Reflection**
5. **Demo Quality & Reproducibility**

Top-scoring projects present on **Apr 29 & May 1**.

---

7. Using ASU Research Computing Resources

If your team needs more compute (e.g., training larger models or more experiments), you can use **ASU Research Computing** resources. Key points from ASU Research Computing:

- ASU's supercomputers are available to **ASU faculty, staff, students, and affiliates** for research and learning; some users (e.g., students) must be **sponsored by a faculty member**.
- To get started, you:
    1. **Create an account** via the Research Computing "Getting Started" page.
    2. Complete an **account request form** (through the RTO/iLab / Request Help portal), agreeing to Research Computing policies.
    3. Wait for approval (typically within ~2 business days per documentation; do not rely on last-minute access).

Course-specific guidance:

- **Plan ahead**: Submit access requests **early in the semester**, not near conference deadlines (May) when queues are longer. For this project, aim to have accounts ready by the time you start intensive training (around the nanoGPT phase).
- **Start small**: Prototype on small datasets and models locally or in Google Colab. Move only heavier experiments to ASU clusters.
- In your reports, clearly note:
    - Whether you used ASU Research Computing.
    - Which system and GPU type you used.
    - Any constraints (queue times, job limits) that influenced your design.

---

8. Evaluation & Grading

- **Idea Pitch (Mar 16)** – 30%
  - Written proposal: 15%
  - Video pitch: 15%
- **nanoGPT Build (Apr 13)** – 40%
  - Correctness and completeness of implementation: 20%
  - Experimental design & analysis (loss curves, configs): 15%
  - Code quality & documentation: 5%
- **MVP (Apr 27)** – 20%
  - Technical depth & model use: 10%
  - Evaluation & reproducible demo: 10%
- **Professionalism / Collaboration** – 10%
  - Responsiveness in peer reviews.

---

9. Academic Integrity & Collaboration

- Teams are expected to collaborate internally and discuss general concepts with other teams.
- You may use open-source codebases (e.g., Karpathy's repositories) provided you clearly attribute the source and explain what you changed/contributed.
- Do not present external code or pre-built demos as entirely your own work without attribution.
- All analysis, reports, and write-ups must be written by your team; tools like ChatGPT may be used as assistants, but you remain responsible for correctness and must not copy large unedited auto-generated text.