Course Project

Code of Honor. All external resources used in the project, including research papers, open-source repositories, datasets, and any content or code generated using AI tools, e.g., ChatGPT, GitHub Copilot, Claude, Gemini, must be *clearly cited* in the final submission. The final report must also include *a clear breakdown of individual group member contributions.* Any lack of transparency in the use of external resources or in reporting group contributions will be considered academic dishonesty and will significantly impact the final evaluation.

Topic	Educational Code Generation using LLMs with Self-Refinement
Category	Applications of Generative Models
Supervisor	Mohammadreza Safavi

Objective Design a simple intelligent agent that takes algorithmic problem descriptions, e.g., from introductory programming course or Leetcode-style tasks, and generates *not only* the corresponding code but also *an educational breakdown of the solution*. The agent should aim to provide human-readable explanations alongside correct and runnable code, and include a *self-refinement* mechanism to debug and correct incorrect generations.

Motivation. Generative models are increasingly used as coding assistants, yet current systems often lack pedagogical depth. This project explores how pretrained LLMs can serve as educational agents, generating explanations that enhance user understanding. It also introduces a layer of reasoning by incorporating feedback loops where generations are tested and refined through a feedback loop.

Requirements The final submission should address the following core components:

1. Choose a small set of algorithmic problems (10–15) from public sources such as Leetcode, HackerRank, or open educational CS repositories with permissions for public usage.

2. Use a pretrained code-generating LLM, e.g., CodeT5 [4], StarCoder [2], GPT-J [3] or Mistral [1], to generate solutions based on the problem statements.

3. Implement a test suite for each problem to evaluate correctness of generated code.

4. Design a prompt structure that encourages the LLM to provide step-by-step explanations, e.g., via docstrings, comments, or separate plan generation.

5. Implement a simple *self-refinement loop* where the agent detects incorrect code via test failures and attempts prompt reformulation or self-debugging.

6. Evaluate both code correctness, e.g., test pass rate, and pedagogical quality, e.g., readability, coherence of explanations.

7. [**Optional**] Implement a simple GAN or VAE to generate stylistic variants of your code snippets. Evaluate the diversity and correctness of generated variants.

Milestones

1. *Dataset preparation and task formulation.* Select algorithmic problems, write ground-truth test cases, and define the format of prompts and expected outputs.

2. *Initial pipeline construction.* Build the core pipeline that sends problem statements to the LLM, retrieves code, and runs it against the test suite.

3. *Pedagogical enhancement.* Extend the pipeline to request and extract explanations or inline comments. Define a rubric for evaluating their quality.

4. *Self-refinement loop.* Build a mechanism for detecting test failures and modifying the prompt or re-querying the model with contextual feedback.

5. [**Optional**] Build a generative model, e.g., VAE or GAN, that learns to transform correct code snippets into stylistically different but functionally equivalent versions. You may use a *small style reference bank* for this part.

6. *Evaluation and final report.* Analyze performance across problem types and LLM behaviors. Document findings, ablations, and limitations in the final report.

Submission Guidelines The main body of work is submitted through Git. In addition, each group submits a final paper and gives a presentation. In this respect, please follow these steps.

- Each group must maintain a Git repository, e.g., GitHub or GitLab, for the project. By the time of final submission, the repository should have
  - Well-documented codebase
  - Clear `README.md` with setup and usage instructions
  - A `requirements.txt` file listing all required packages or an `environment.yaml` file with a reproducible environment setup
  - Demo script or notebook showing sample input-output
  - *If applicable,* a `/doc` folder with extended documentation

- A final report (maximum *5 pages*) must be submitted in a PDF format. The report should be written in the provided formal style, including an abstract, introduction, method, experiments, results, and conclusion.
  Important: Submissions that do not use template are considered *incomplete.*

- A 5-minute presentation (maximum *5 slides including the title slide*) is given on the internal seminar on Week 14, i.e., *Aug 4 to Aug 8,* by the group. For presentation, any template can be used.

Final Notes While planning for the milestones please consider the following points.

1. You are encouraged to explore innovative approaches to conditioning or generation as long as the core objectives are met.

2. While computational resources are limited, carefully chosen datasets and training setups can make even diffusion models feasible. Trade-offs, e.g., resolution, training steps, are expected and should be justified.

3. Teams are expected to manage their computing needs and are advised to perform early tests to estimate runtime and training feasibility. As graduate students, team members can use facilities provided by the university, e.g., ECE Facility. Teams are expected to inform themselves about the limitations of the available computing resources and design the model accordingly.

References

[1] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

[2] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*, 2023.

[3] Ben Wang. Mesh-Transformer-JAX: Model-Parallel Implementation of Transformer Language Model with JAX. `https://github.com/kingoflolz/mesh-transformer-jax`, May 2021.

[4] Yue Wang, Weishi Wang, Shafiq Joty, and Steven CH Hoi. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. *arXiv preprint arXiv:2109.00859*, 2021.