Project 2: Exploring the Potential of Large Language Models for Assembly Code

1. Project Summary

In this project, students are expected to evaluate large language models (LLMs) for understanding and generating assembly language, including but not limited to x86 assembly.

2. Ways to access code LLMs

There are three ways to access large language models:

1) via web or app-based conversation, such as ChatGPT, Kimi, and doubao, etc.;
2) through API calls, like GPT-4, DeepSeek Coder, Qwen-turbo, and Zhipu AI;
3) by using open-source models, such as the Llama series, Qwen series, and ChatGLM series, which can also be fine-tuned or further developed.

3. Topic Recommendation

Here are some topic recommendations that you can explore, but these are not exhaustive lists. Feel free to explore any topics related to evaluating how LLMs perform in assembly language and low-level programming.

- **Exploring LLMs' Understanding of Assembly Instruction Sets**
  1) Construct test sequences of assembly instructions
  2) Evaluate and analyze the LLM’s understanding of assembly instructions, including its comprehension of addressing modes, calculation results, and status flag updates.
  3) Attempt to correct LLMs' understanding errors.
  4) Identify scenarios where the LLM’s understanding is incorrect and investigate patterns in these errors.

- **Exploring LLMs' Understanding of Assembly Directives**
  1) Construct test sequences of assembly instructions to evaluate and analyze LLMs' understanding of assembly directives.
  2) Investigate the accuracy of LLMs' comprehension of concepts such as instructions, directives, and instruction prefixes.
  3) Attempt to correct LLMs' understanding errors.
  4) Identify scenarios where the LLM’s understanding is incorrect and investigate patterns in these errors.

- **Exploring the Ability of LLMs to Identify Different Versions of the Same Code**
  1) Identification of different versions of the same code.
  2) Analysis of the differences and respective advantages of each optimized version.
  3) If identification is inaccurate, attempt to correct the errors.
  4) Identify scenarios where the LLM’s identification is incorrect and investigate patterns in

these errors.

- **Exploring the Ability of LLMs to Generate Assembly Code**
  1) Designing tasks for assembly code generation
  2) Evaluating and analyzing the code generation capabilities of LLMs
  3) Identify scenarios where the LLM's generated code is incorrect, inefficient, or fails to compile, and investigate patterns in these errors.

- **Exploring the Error-Correction Capabilities of LLMs for Assembly Code**
  1) Designing tasks for assembly code generation
  2) Evaluating and analyzing the error-correction capabilities of LLMs for incorrect code
  3) Identify scenarios where the LLM cannot correct the code or where the corrected code is still incorrect, and investigate patterns in these errors.

- **Exploring the Ability of LLMs to Translate Assembly Code**
  1) Prompt the LLM to generate functionally equivalent assembly routines across multiple architectures. For example, translate a function call from x86 to RISC-V, ARM, or MIPS.
  2) Analyze code output for syntax accuracy, register usage, and any notable differences that reflect each architecture's unique requirements.
  3) Identify scenarios where the LLM's generated code is incorrect, inefficient, or fails to compile, and investigate patterns in these errors.

- **Exploring the Capability of LLMs in Generating Optimized Assembly Code**
  1) Design code optimization objectives (e.g., minimizing the number of instructions, minimizing the space occupied by instructions)
  2) Evaluate the ability of LLMs to generate assembly code that meets specified optimization objectives.
  3) Identify scenarios where the LLM's generated code is incorrect, inefficient, or fails to compile, and investigate patterns in these errors.

4. Submission Requirements

- **Deadline: December 31, 2024 by 23:59.**
- **Late submission policy**
  1) Within 24 hours after the submission is due: 50% point deduction;
  2) More than 24 hours late: 100% point deduction.
- **Submission Checklist**
  1) Technical report, code, and the screenshot of the interaction with LLMs..
  2) Demonstration video (optional)
- **Code Submission Requirements**
  1) Source code files, release version executable files.
  2) Please remove unnecessary intermediate files from the project.
  3) Please include program instructions, development environment, usage conditions, and run command instructions.

5. References

- Unifying the Perspectives of NLP and Software Engineering: A Survey on Language Models for Code, https://arxiv.org/abs/2311.07989
- Awesome-Code-LLM : https://github.com/codefuse-ai/Awesome-Code-LLM
- Mankowitz, Daniel J., et al. "Faster sorting algorithms discovered using deep reinforcement learning." Nature 618.7964 (2023): 257-263.
- DeepSeek Coder: https://www.deepseek.com