Milestone 2 – One-Pager

Project Information

Title: CTokenAnalyzer

Team Members: TJ Hall

Executive Summary

This will be a C/C++ program to simulate the execution of AI and calculate it's cost at granular levels (total compute cost, per token cost, etc.) for developer testing their local AI models.

Problem Statement

Currently, AI cost projection is usually a black box given to a user by random per token cost input/output. This program aims to breakdown the cost's of AI in electricity and compute rather than giving a simple cost per token.

Goals

1. Enabled user to define their local electricity cost, hardware power draw, and max context length of a model to calculate basic total estimated cost given a string of tokens
2. Increase variations of tokenizers to show how different tokenizers affect overall outcome.

Proposed Approach

There will be a basic a CostCalculator, Tokenizer, and Backend. The backend is not needed for now but for future extension of integrating real AI models or a specific computation of input tokens to generate output tokens. The other's do as they say calculates cost given a number of token and break's a string into tokens. There will also be a basic UI class that allow's the user to interact with the given classes.

Inheritance

1. Cost Calculator will always have an cost calculator strategy inherited from it's specific implementation (simpleCalc, complexCalc, etc.)
2. Tokenizer will inherit the tokenization algorithm from it's respective tokenizer

Interfaces

1. ICostCalculator
    - Common calculator interface to take token count and determine electrcity and token cost
2. ITokenizer
    - Common tokenizer to return parsed tokens from a given string, there can be many different variations but they all must have the same interface
3. IBackend
    - Determines the type of AI model on the backend which defines max-token length, compute-time-per-token, etc.

Design Patterns

1. Strategies, A tokenizer will implement algorithm's for splitting a string into an array of smaller string's or character's because of this different algorithms will be used to split strings in a different matter resulting in the need for different tokenization strategies.
2. Adapter Pattern, for more common currencies like the Euro and Yen add a basic adapter to calculate cost for different countries as all calculation initially will be in USD.
3. Singleton CTokenAnalyzer for the UI which integrates all classes and interfaces defined for operation.

Key Requirements

Functional:

- Take user input for given string, electricity cost, system power draw, and max context length.
- Use max content length and user string to calculate maximum number of tokens the AI operation could cost.
- Use maximum number of tokens to determine the cost in electricity Non-functional:
- Testing real-time hardware to map to actual power values
- Integrating with Llama.cpp to get real token outputs rather than max estimations

Risks or Challenges

1. C/C++, I want to use C/C++ with C-style API's to be interoperable with as many modern hardware machines as possible. However, this increases basic type complexity and accurate memory management.
2. Accuracy in math, a lot of the values I need can only be simulated or derived real time making accurate power and cost calculations hard. Which is why the MVP will be a basic simulation with user defined values and if those simulations result to close enough real life with my algorithms then I will worry about implementing for real models and/or hardware.