**Problem Statement Discussion Summary**

**1) Problem Statement (3–5 sentences)**

The problem this project aims to solve is estimating the **local electricity cost** of running AI workloads on a user’s own machine, expressed both as **total run cost** and **per-token electricity cost**. Developers and students working with local LLMs often lack a quick, console-based way to translate **power draw**, **token throughput**, and **token counts** into understandable cost figures without relying on web pricing or hardware auto-detection. This project will provide a deterministic, input-driven calculator that tokenizes text, derives run time from tokens/sec, converts power to energy (kWh), and reports electricity cost and cost per token using user-supplied parameters. The result is valuable as a repeatable, offline baseline for budgeting and teaching, and it is feasible for a solo developer in 5 weeks with a console UI.

**2) Conversation Summary**

**Major discussion points & how the idea evolved**

* **Initial idea**: “Basic power/cost analyzer for AI tokens” that could approximate cost per token (electricity or dollars) from user parameters; intentionally **no hardware auto-detect** in MVP.
* **Clarifications**: Chose **C++** console; committed to **local electricity model**, not API token pricing (web) for MVP; **user provides** tokens/sec.
* **Scope discipline**: Tokenizers will start simple (word/subword) with a strategy-friendly design to add more later; output is **total electricity cost** and **per-token electricity cost**; file persistence is deferred.
* **Acceptance posture**: Inputs are free-form (no strict validation ranges); on bad input, the program exits (MVP simplicity).
* **Feasibility**: Week-by-week plan settled; demo will compare console output to TJ’s measured wall-power samples at home.

**Final boundaries of scope**

* **In scope (MVP)**
  + Console-only app in C++
  + User inputs: total system power (W), kWh price (USD/kWh), tokens/sec (tokens/s), and text to tokenize (developer/education both use manual inputs)
  + Tokenization: basic word/subword (strategy-ready)
  + Compute: total tokens → run seconds → kWh → electricity cost (USD) → cost per token (USD/token)
  + Output: total tokens, run seconds, kWh, electricity cost, per-token electricity cost
  + Error handling: any bad input → exit
* **Out of scope (MVP)**
  + Hardware auto-detection or power telemetry
  + Web/API price estimation (OpenAI per-token pricing)
  + File persistence, reports, or GUI

**Users & Use Cases (as stated by the team)**

* **User Types (team-authored)**
  + **Developer** — wants to see real costs of different inputs in compute and cost for local AI.
  + **Education** — demonstrate that context length and AI generation have real electricity costs under strict parameters; same flow, illustrative values.
* **Key Use Cases & Happy Paths (team-authored language)**
  + **Developer — Critical use case**
    1. **Enter total system power draw**
    2. **Choose tokenizer**
    3. **Enter kWh price**
    4. **Enter string**
    5. **Calculate Cost**
    6. **Per-token analysis**
  + **Education — Critical use case**
    1. *Same as Developer*, using illustrative (not necessarily exact) values provided by the user at the console.

**Clarified terms / assumptions & acceptance criteria**

* **“Local-only”** = no web access and no API token-pricing in MVP.
* **Timing model** = user provides **tokensPerSecond**; total tokens are the sum of input tokens and (if provided) output tokens.
* **Determinism** = fixed equations: tokens → runSeconds → kWh → cost → per-token cost.
* **Acceptance criteria (team-aligned, testable at console)**
  + Program starts in console and prompts for: totalSystemPowerWatts [W], pricePerKWhUSD [USD/kWh], tokensPerSecond [tokens/s], and an **input string** (tokenized); output tokens may be provided or derived per tokenizer limits.
  + After **Calculate Cost**, the console displays: totalTokens, runSeconds, energyKWh, electricityCostUSD, and perTokenElectricityCostUSD (units shown with each value).
  + **Bad input** (non-numeric for numeric prompts) causes an immediate, clear **exit** message (no retries required in MVP).

**3) Course-Fit Verification (checked by the mentor)**

* ☑ **Written in C#, Java, or C++** → **C++**
* ☑ **Problem is clear and specific to a user group** → Local LLM users (Developer), and Education for demonstration
* ☑ **Scope is manageable in 5 weeks for 1 person** → Console-only, simple tokenizers, fixed equations, no auto-detect
* ☑ **Console UI is feasible** → Prompts + printed results; file output deferred
* ☑ **OOP principles feasible now; patterns later**
  + **Encapsulation**: CostCalculator owns power/price parameters and math; Tokenizer owns token counting logic.
  + **Abstraction / Interfaces**: pure-virtual base classes like ITokenizer and ICostCalculator.
  + **Inheritance**: WordTokenizer, SubwordTokenizer derive from ITokenizer; future calculators derive from ICostCalculator.
  + **Polymorphism**: main UI holds std::unique\_ptr<ITokenizer> / ICostCalculator and invokes via base interfaces.
  + **I/O**: console read/write of parameters and results (units included).
  + **Design patterns (for later milestone)**: strategy for tokenizers/calculators; others can be added from a second GoF category when required.