CSCE 631-D: TAMU LLM API Setup Guide — Fall 2026

Alan Kuhnle, Texas A&M University

1 What is the TAMU API?

Texas A&M provides access to several large language models (Claude, GPT, Gemini) through `https://chat.tamu.ai`, an OpenAI-compatible API gateway. For your programming assignments, you will use this API to build and evaluate systems with real LLMs.

Access requires two credentials:

1. A **shared API key** (provided below — same for all students in the course). This key identifies the *course application* to the gateway, not any individual student.
2. A **Cloudflare Access cookie** from your browser session (personal to your NetID, expires every ~24 hours). This authenticates *you* and enforces your per-student usage quota.

Your usage is capped at **$5/day**, enforced per-student via the cookie. This is more than enough for your programming assignments if you follow the budgeting guidance below.

2 Step 1: Log into chat.tamu.ai

1. Open your browser and navigate to **`https://chat.tamu.ai`**
2. Click **"Sign in with TAMU SSO"**
3. Authenticate with your NetID and Duo MFA
4. You should see the chat interface — this confirms your session is active

You need an active browser session before proceeding. The session sets a Cloudflare Access cookie that your Python code will use.

3 Step 2: Get the Cloudflare Cookie

After logging in, you need to extract the `CF_Authorization` cookie from your browser. This cookie is marked **HttpOnly**, which means JavaScript in the console cannot access it via `document.cookie`. You must use one of the two DevTools methods below.

3.1 Method A: Application Tab

1. Open **DevTools** (press `F12`, or right-click anywhere and select "Inspect")
2. Go to the **Application** tab (in Chrome/Edge) or **Storage** tab (in Firefox)
3. In the left sidebar, expand **Cookies** and click on `https://chat.tamu.ai`
4. Find the cookie named **`CF_Authorization`**
5. Click on it and **copy the Value** — it is a long JWT string starting with `eyJ...`

3.2 Method B: Network Tab

1. Open **DevTools** (press `F12`) and go to the **Network** tab
2. **Refresh the page** (`Ctrl+R` or `F5`)
3. Click the **first request** to `chat.tamu.ai` in the list (usually the document request at the top)
4. In the panel that opens on the right, scroll down to **Request Headers**
5. Find the line starting with `Cookie:` — it contains multiple `name=value` pairs separated by semicolons
6. Copy everything from `CF_Authorization=eyJ...` through the end of that cookie value (it ends before the next `;`)

This method has the advantage that the raw cookie header shows the exact `CF_Authorization=eyJ...` format you need to paste into your Python code.

Important:

- This cookie expires after approximately **24 hours** — you will need to repeat Steps 1–2 each coding session
- The cookie is personal to your NetID — do not share it

4 Step 3: Install the OpenAI Python Library

The TAMU API is OpenAI-compatible, so we use the standard OpenAI Python client:

```
pip install openai
```

If you are using a virtual environment or conda, activate it first.

5 Step 4: Make Your First API Call

Create a Python file or Jupyter cell with the following code:

```python
import openai

# Paste your CF_Authorization cookie value here (the eyJ... string)
CF_COOKIE = "CF_Authorization=eyJ..."  # <-- replace with your actual cookie

client = openai.OpenAI(
    base_url="https://chat.tamu.ai/api",
    api_key="sk-0183ed7c1c8e47c0a8f4d1e75161aa6d",
    default_headers={"Cookie": CF_COOKIE}
)

response = client.chat.completions.create(
    model="protected.Claude Sonnet 4.5",
    messages=[{"role": "user", "content": "What is a Nash equilibrium? Answer in one sentence
    ."}],
    temperature=1,       # Required for Claude thinking models on TAMU
    max_tokens=16384,    # Must exceed the thinking budget (see note below)
    stream=True
)

text = ""
for chunk in response:
    if chunk.choices[0].delta.content:
        text += chunk.choices[0].delta.content
print(text)
```

Note: The `stream=True` parameter is required for the TAMU API gateway. The gateway returns string-wrapped responses in non-streaming mode, which prevents the OpenAI client from parsing the response object correctly. Streaming mode avoids this issue entirely.

If you see a coherent answer about Nash equilibrium, your setup is working.

Important — Thinking Mode on Claude Models: The TAMU gateway enables *extended thinking* on Claude Sonnet and Opus models. This has three consequences:

1. `temperature` must be exactly `1` — other values will produce an error.
2. `max_tokens` must be large (at least 16384) — the model uses some of these tokens for internal reasoning before producing the answer. Setting `max_tokens=256` will fail because it is smaller than the thinking budget.
3. The response includes reasoning content — in addition to `response.choices[0].message.content` (the actual answer), the response also contains `response.choices[0].message.reasoning_content` (the model's chain of thought). You can ignore the reasoning content for your programming assignments — just read `.content`.

Non-Claude models (GPT, Gemini) do **not** have this constraint and work with any `temperature` and small `max_tokens` values.

6 Available Models

The gateway provides access to 30+ models. The table below lists the most useful ones for your programming assignments; use `client.models.list()` to see the full list.

Model identifier	Provider	Notes
`protected.Claude Sonnet 4.5`	Anthropic	**Recommended default.** Requires `temperature=1`, `max_tokens`≥`16384` (thinking mode).
`protected.Claude-Haiku-4.5`	Anthropic	Fastest Claude model. No thinking constraints — works with any `temperature` and small `max_tokens`. Good for high-volume experiments.
`protected.Claude Opus 4.5`	Anthropic	Strongest reasoning, slower and more expensive. Thinking mode.
`protected.gpt-5-mini`	OpenAI	Fast, cheap. Works with standard parameters (`temperature=0.7`, `max_tokens=256`).
`protected.gpt-5.2`	OpenAI	Strong reasoning model.
`protected.o3`	OpenAI	Reasoning specialist (chain-of-thought).
`protected.gemini-2.5-pro`	Google	Alternative for comparison experiments.
`protected.gemini-2.5-flash`	Google	Fast and cheap Google model.

Thinking-mode models (Claude Sonnet 4/4.5/4.6, Claude Opus) require `temperature=1` and `max_tokens`≥`16384`. All other models accept any `temperature` and small `max_tokens`.

Recommendation: Use `protected.Claude Sonnet 4.5` as your default — it is the best balance of quality and cost. For high-volume experiments, such as PA1 empirical payoff estimation or PA3 Kuhn Poker self-play, `protected.Claude-Haiku-4.5` or `protected.gpt-5-mini` are cheaper alternatives. Use a different provider (GPT or Gemini) when your assignment calls for a model comparison.

7 Budget Awareness

Your daily cap is **$5/day**. Here is a rough guide to costs:

- **Claude Sonnet 4.5:** approximately $3/M input tokens, $15/M output tokens
- A typical single-turn API call: ~200 input tokens and ~100 output tokens
- Your exact cost depends on the model, prompt length, and generated response length
- You can run many such calls per day on Sonnet without exceeding $5; start small and monitor usage before scaling an experiment

Track your usage by checking the `usage` field in each API response:

```
print(f"Prompt tokens: {response.usage.prompt_tokens}")
print(f"Completion tokens: {response.usage.completion_tokens}")
print(f"Total tokens: {response.usage.total_tokens}")
```

Cost-saving tips:

- For Claude thinking models, `max_tokens` must be large (16384), but the model will stop generating when the answer is complete — you are only billed for tokens actually produced, not the max_tokens cap
- For non-thinking models (Haiku, GPT, Gemini), keep `max_tokens` low (256 or less)
- Use `protected.Claude-Haiku-4.5` or `protected.gpt-5-mini` for high-volume experiments — they are significantly cheaper per token
- Cache results: save API responses to a JSON file so you don't repeat identical calls
- Start with small experiments (50 games) before scaling up

8 Troubleshooting

Error	Cause	Fix
`401 Unauthorized` or `403 Forbidden`	Cookie expired	Log into `chat.tamu.ai` again and copy a fresh `CF_Authorization` cookie
`"model not found"` or `"invalid model"`	Wrong model name	Check that the model name exactly matches the table above, including the `protected.` prefix
Empty or nonsensical response	Content filtering triggered	Rephrase your prompt; avoid adversarial or sensitive content
`429 Too Many Requests`	Rate limiting	Add `time.sleep(1)` between API calls
`openai.APIConnectionError`	Network issue	Check your internet connection; retry after a moment
Response object has no `usage` field	Some models omit usage stats	Wrap in `getattr(response.usage, 'total_tokens', 0)`

9 Important Notes

- **Do NOT share the API key** outside this course
- The `CF_Authorization` cookie is **personal to your NetID** — do not share it
- Always include **both** the `Cookie` header and the `Authorization` header (the `api_key` parameter handles the latter)
- **Add error handling** in your programming-assignment code — API calls can fail intermittently
- If you exhaust your $5/day budget, you will need to wait until the next day to continue
- All API calls are logged — usage that violates TAMU's acceptable use policy may result in access revocation