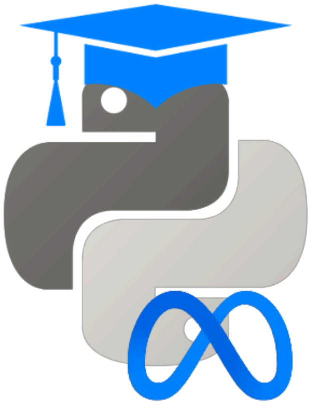
Anatole Dupuis
Final Project
COSC 461-001
Qingqing Li

.PYLLAMATUTOR

# AI Tutor for Introductory Python Programming Design Report



## Goals

- Deliver a conversational tutor for beginners (0–1 semester of Python) that can explain fundamentals, debug small snippets, provide examples, generate exercises, and give feedback.
- Enforce a consistent four-section response structure (Concept Explanation, Code Example, Practice Exercise, Feedback) to keep guidance predictable.
- Use an open-source LLM (Llama 3.2 1B Instruct) locally so the app can run offline after initial setup; prefer GPU/Metal but fall back to CPU automatically if needed.
- Track token usage and estimated cost to demonstrate awareness of LLM metering (token counts and cost are shown in the footer).
- Provide a simple, reliable PyQt GUI that remains responsive during model inference and adapts to user hardware via presets; include curated quick prompts (no auto-evaluation per latest scope).

## Design

- **Architecture**: gui.py (PyQt chat UI) → agent.py (enforces structure, session token/cost tracking) → model_client.py (LLM wrapper with native Llama loader, transformers fallback, and CPU retry). setup.py handles dependency install/model download. app.py provides the launcher with a splash for dependency + system scanning + model warmup before the main window opens.

- **Response format**: Agent builds a system prompt reminding the model to always produce the four sections. GUI post-processes headings to bold labels for clarity, renders code blocks with lightweight syntax tinting, and separates turns with padded horizontal rules.
- **Local model**: Uses Llama 3.2 1B Instruct checkpoint downloaded via the signed Meta URL into ~/.llama/checkpoints/Llama3.2-1B-Instruct. model_client first tries transformers; if unavailable, it loads the native checkpoint with GPU/Metal preference but will reload on CPU if native generation fails (e.g., MPS placeholder issue). Initialization is warmed up during splash and retried lazily if needed.
- **UI/UX**: QTextEdit history with colored speaker labels (green for user, red for tutor); busy indicator while the model runs in a background thread; footer shows model, device, tokens, cost, and last response time. Splash handles dependency checks/installs, system scan, and model warmup. Quick prompts menu pre-fills common questions. Code blocks are syntax-tinted.
- **Presets**: On launch, a system scan (CPU cores, RAM, CUDA/Metal detection) derives light/balanced/overkill presets (device, max tokens, CPU threads) with fallback defaults if scanning fails. Config tab shows scan status, allows preset selection, manual tweaks, and background rescan; manual mode bypasses presets.
- **Token/cost**: Token counting via tokenizer when available (heuristic fallback) and session totals surfaced in the GUI footer; illustrative pricing for cost estimation. These satisfy the “demonstrate tokenization and cost analysis” requirement even for local models.

## Tools & Libraries

- **PyQt5**: Desktop GUI, threading (QThread) to avoid blocking, rich-text display for messages and formatting; splash dialog for dependency and system scan.
- **llama-models**: Meta tooling to download and run native Llama checkpoints; provides chat formatting utilities. Additional deps: fairscale, torchvision for the native loader.
- **Transformers/Torch**: Optional pipeline path if a Hugging Face–style model is available; native path used here for the Meta checkpoint. Lazy init prevents UI freeze.
- **tiktoken**: Optional token counting (falls back to heuristic if missing).
- **psutil** (optional): Used for hardware detection when present.

## Implementation Details

- **Setup/quickstart**: python3 app.py runs a splash that checks dependencies (installs missing ones), scans hardware, warms up the model, and then launches the GUI. Defaults: model path ~/.llama/checkpoints/Llama3.2-1B-Instruct; override with LLAMA_LOCAL_PATH if needed. Device preference via Config (auto/cpu/gpu with CPU fallback if GPU/MPS fails); CPU threads configurable.

- **Model loading**: Environment defaults (RANK=0, WORLD_SIZE=1, MASTER_ADDR=127.0.0.1, MASTER_PORT=29500) ensure torch.distributed initializes on CPU/GPU as needed. Native Llama3 loader consumes the Meta checkpoint shards and tokenizer. Outputs are stripped of special end tokens before display. Lazy init keeps startup fast; splash warmup plus CPU retry improves reliability.
- **Threading**: AskWorker runs in a separate QThread; UI disables input/send (and Apply) while a response is in flight, shows a busy bar, and re-enables once the response arrives. Dependency/system scan and model warmup run in splash threads; Config rescan runs in a background thread. A response timer is shown in the footer. Evaluation rows are persisted to disk and reloaded on startup (current session rows are tinted using the system palette).
- **Formatting**: GUI converts any ### Section headers to bold labels; message labels are colored; history uses HTML with padded separators; code blocks are syntax-tinted. Placeholders only appear if model loading fails after retry.

## Evaluation

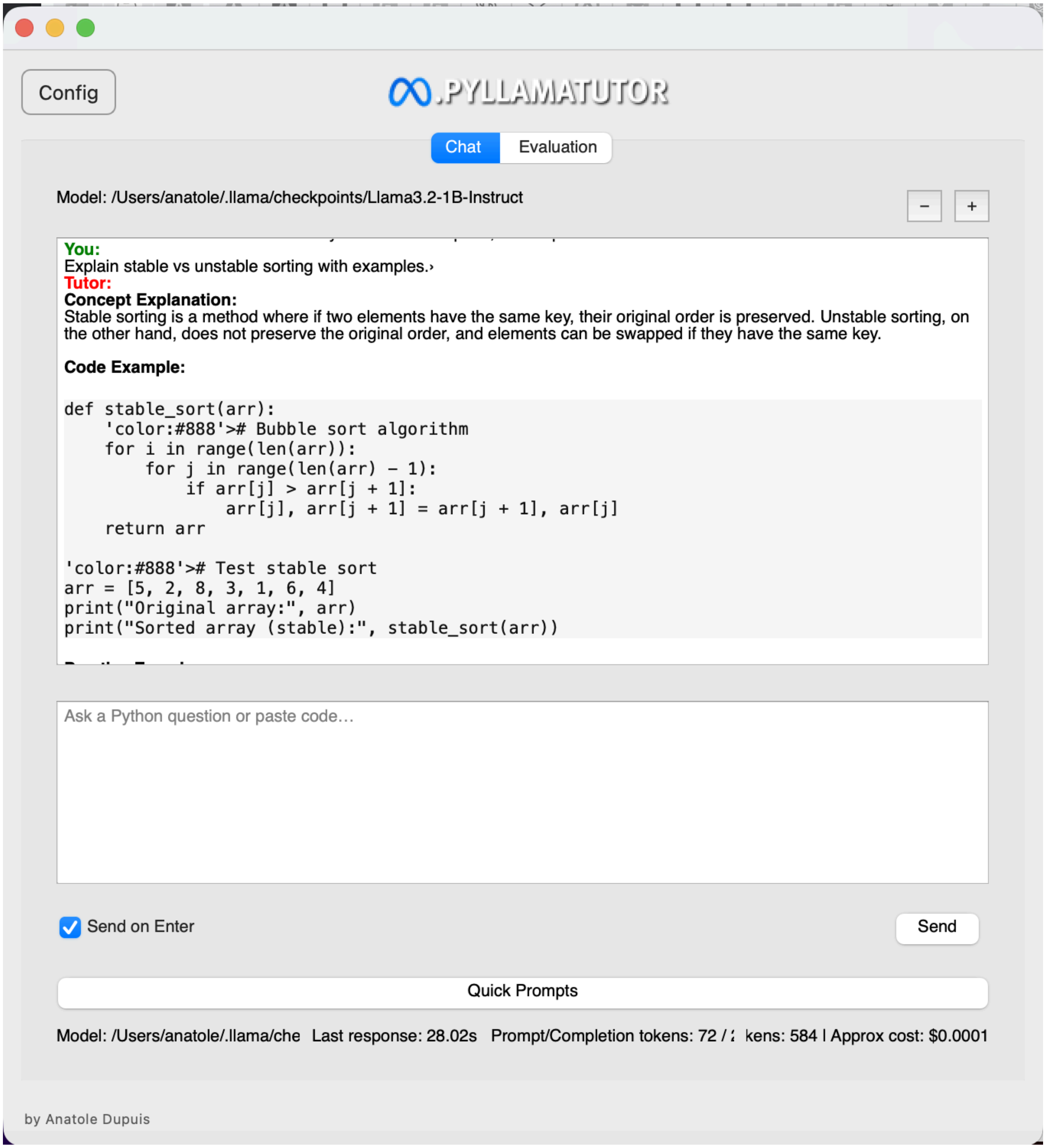
- **Correctness**: Manual checks on prompt/response structure; quick prompts validated the four-section format. Native loader tested with the downloaded checkpoint; if GPU/Metal errors occur, CPU retry restores generation. Placeholder path verified when no model is available.
- **Performance**: Splash warmup plus lazy init prevent UI freeze; first load incurs checkpoint load time. Presets allow quick tuning (device/tokens/threads). CPU inference on Llama 3.2 1B is slower than GPU but acceptable for short answers; quantization options are available in Config (int4/fp8 when supported).
- **UX responsiveness**: Background threads keep the window responsive during installs, scans, and inference. Colored labels, bold headers, and padded separators improve scanability; footer shows timing, token counts, and cost to demonstrate metering.
- **Tokenization & metering**: Token usage comes from the tokenizer when available (heuristic fallback otherwise). Totals/averages in the Evaluation tab are computed across all saved evaluations (persisted to disk), per-row response time is shown, and a wipe control clears the stored metrics.
- **Persistence**: Evaluation rows (timestamp, tokens, cost, model/device/quant, elapsed) are auto-saved to ~/.pyllama/chat_history.json after each turn, reloaded on startup, and aggregated into totals; a wipe control clears the file and in-memory stats. Chat text itself is not persisted or replayed.
- **Risks/Limitations**:
  - CPU-only inference is slower; consider GPU if available or quantized variants for speed.

  - Signed URLs expire; users must supply a valid LLAMA_CUSTOM_URL if the bundled one is invalid.
  - Native loader depends on fairscale/torchvision; if missing or incompatible, the client falls back to placeholder text. Missing optional deps (e.g., torchao) will disable native quantized loads.
  - Model license terms (Meta Llama) apply to checkpoints; source code is MIT.

## Future Work

- Add memory management for more refined help.
- Use the LLM model to generate challenges like leet code.
  - Model can generate questions and store answers
  - Users can attempt challenge and confirm answer with model
  - Users could even get hints from model
- Use a newer model for quicker answer generation and more accuracy.
- Create mobile app

# Main view



Model: /Users/anatole/.llama/checkpoints/Llama3.2-1B-Instruct

**You:**
Explain stable vs unstable sorting with examples.›
**Tutor:**
**Concept Explanation:**
Stable sorting is a method where if two elements have the same key, their original order is preserved. Unstable sorting, on the other hand, does not preserve the original order, and elements can be swapped if they have the same key.

**Code Example:**

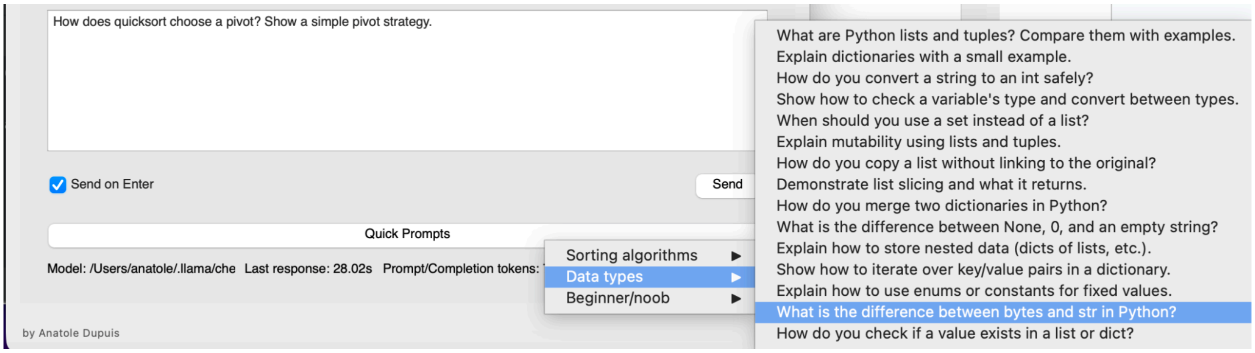
```
def stable_sort(arr):
    'color:#888'># Bubble sort algorithm
    for i in range(len(arr)):
        for j in range(len(arr) - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    return arr

'color:#888'># Test stable sort
arr = [5, 2, 8, 3, 1, 6, 4]
print("Original array:", arr)
print("Sorted array (stable):", stable_sort(arr))
```

Model: /Users/anatole/.llama/che Last response: 28.02s Prompt/Completion tokens: 72 / kens: 584 | Approx cost: $0.0001

by Anatole Dupuis

# Quick Prompts



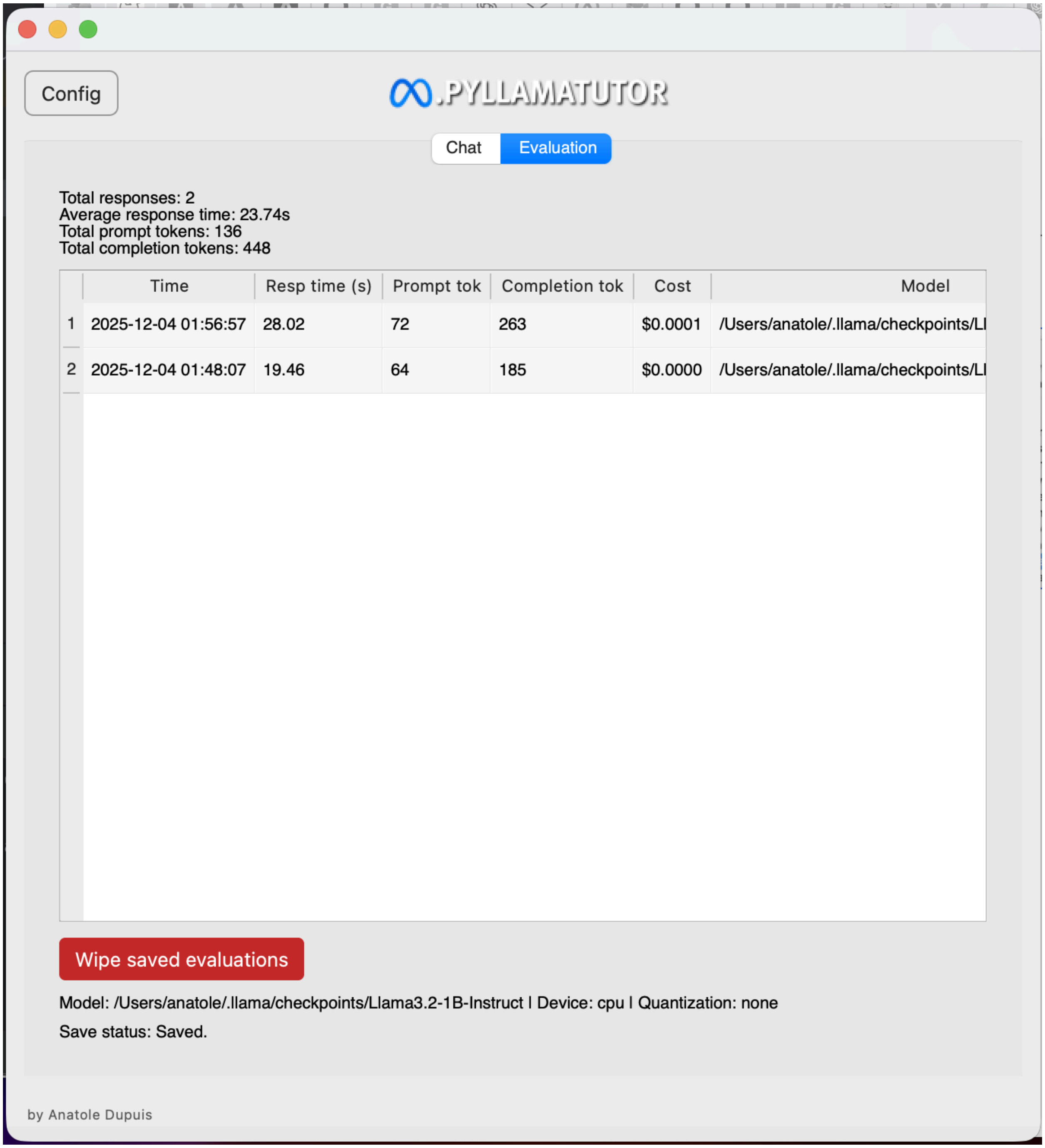
How does quicksort choose a pivot? Show a simple pivot strategy.

☑ Send on Enter

Model: /Users/anatole/.llama/che  Last response: 28.02s  Prompt/Completion tokens:

Send
Quick Prompts
Sorting algorithms ▶
Data types ▶
Beginner/noob ▶
What are Python lists and tuples? Compare them with examples.
Explain dictionaries with a small example.
How do you convert a string to an int safely?
Show how to check a variable's type and convert between types.
When should you use a set instead of a list?
Explain mutability using lists and tuples.
How do you copy a list without linking to the original?
Demonstrate list slicing and what it returns.
How do you merge two dictionaries in Python?
What is the difference between None, 0, and an empty string?
Explain how to store nested data (dicts of lists, etc.).
Show how to iterate over key/value pairs in a dictionary.
Explain how to use enums or constants for fixed values.
What is the difference between bytes and str in Python?
How do you check if a value exists in a list or dict?
by Anatole Dupuis

# Persistent Evaluations



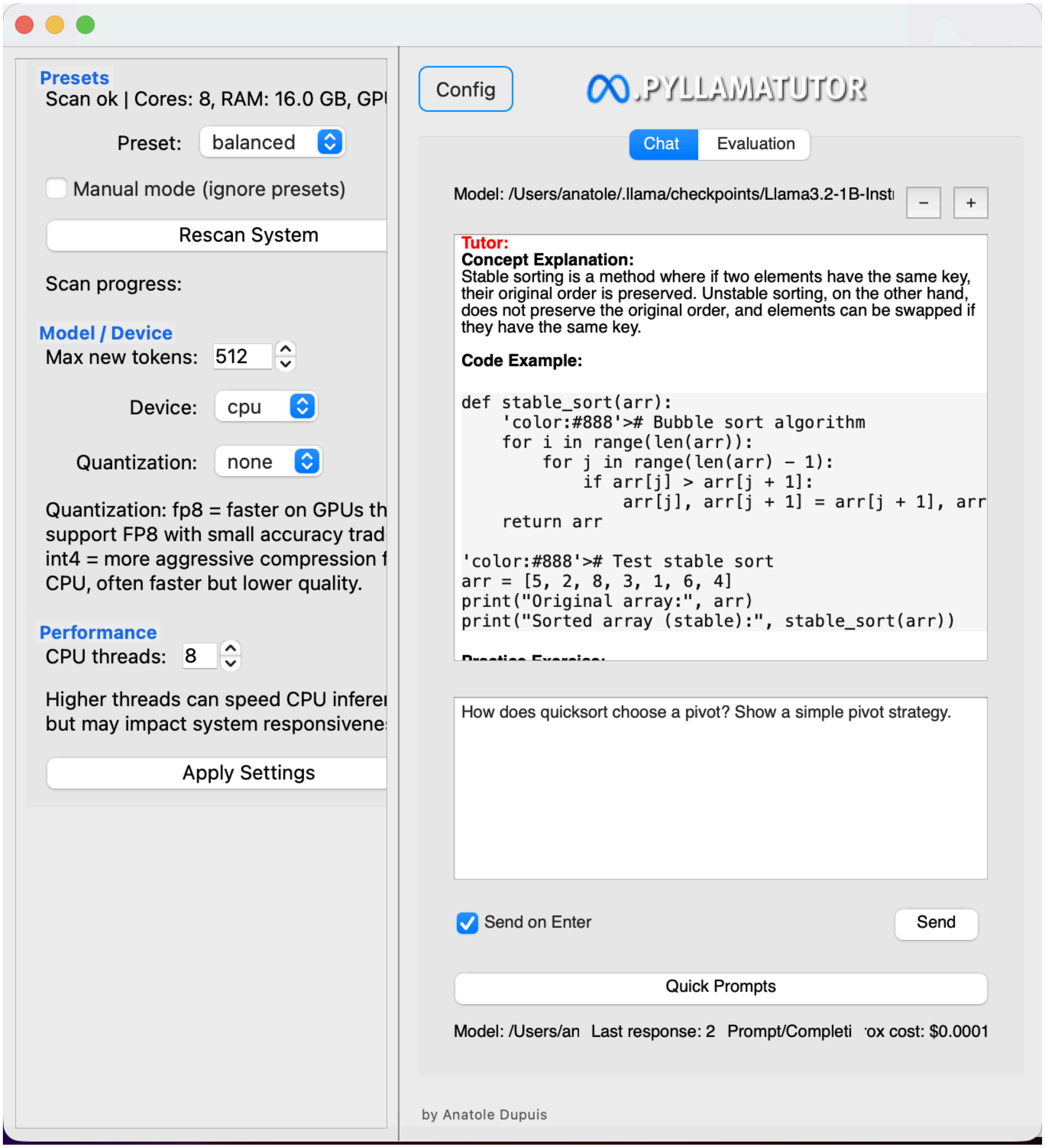
Total responses: 2
Average response time: 23.74s
Total prompt tokens: 136
Total completion tokens: 448

| | Time | Resp time (s) | Prompt tok | Completion tok | Cost | Model |
|---|---|---|---|---|---|---|
| 1 | 2025-12-04 01:56:57 | 28.02 | 72 | 263 | $0.0001 | /Users/anatole/.llama/checkpoints/Ll |
| 2 | 2025-12-04 01:48:07 | 19.46 | 64 | 185 | $0.0000 | /Users/anatole/.llama/checkpoints/Ll |

Model: /Users/anatole/.llama/checkpoints/Llama3.2-1B-Instruct | Device: cpu | Quantization: none

Save status: Saved.

# Configs



**Presets**

Scan ok | Cores: 8, RAM: 16.0 GB, GPI

Preset: balanced

☐ Manual mode (ignore presets)

Scan progress:

**Model / Device**

Max new tokens: 512

Device: cpu

Quantization: none

Quantization: fp8 = faster on GPUs th
support FP8 with small accuracy trad
int4 = more aggressive compression f
CPU, often faster but lower quality.

**Performance**

CPU threads: 8

Higher threads can speed CPU infere
but may impact system responsivene

Model: /Users/anatole/.llama/checkpoints/Llama3.2-1B-Insti

**Tutor:**

**Concept Explanation:**

Stable sorting is a method where if two elements have the same key, their original order is preserved. Unstable sorting, on the other hand, does not preserve the original order, and elements can be swapped if they have the same key.

**Code Example:**

```
def stable_sort(arr):
    'color:#888'># Bubble sort algorithm
    for i in range(len(arr)):
        for j in range(len(arr) - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr
    return arr

'color:#888'># Test stable sort
arr = [5, 2, 8, 3, 1, 6, 4]
print("Original array:", arr)
print("Sorted array (stable):", stable_sort(arr))
```

**Practice Exercise:**

How does quicksort choose a pivot? Show a simple pivot strategy.

☑ Send on Enter

Model: /Users/an Last response: 2 Prompt/Completi ·ox cost: $0.0001

by Anatole Dupuis