How To Call TT Chat API

2025/02/19

This document explains how to call the chat service API in TT Cloud.

To chat with our models in a web browser or your phone, go to chat.tenstorrent.com. There you will need to sign in with your TT email and your TT Microsoft account.

To make API calls to the same service, the url is https://litellm-proxy--tenstorrent.workload.tenstorrent.com. Please read on.

Credentials

First you have to have valid credentials for calling the API.

Please contact the Cloud Team through slack in `#ttcloud_support` if you are interested in getting credentials for using the API.

We recommend using python-dotenv with a file to hold your secret credentials to be used in a python script making API calls.

Privacy

For the models we provide chat UI and API access to, they should be safe to share Tenstorrent data with because either the models are hosted by us in TT Cloud, or they are paid-tier models in 3rd party clouds connected to our proxy that should be privacy safe because they are “paid tier”.

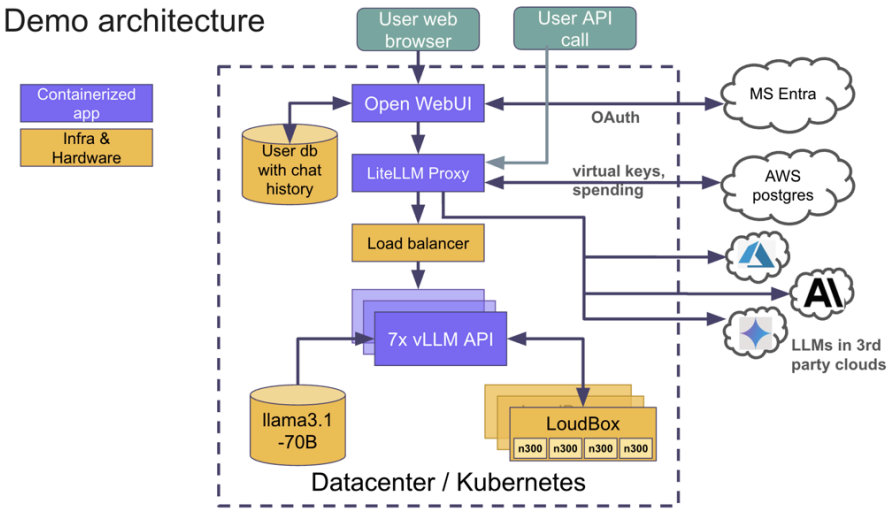
Do not discuss Tenstorrent data with your own personal chat accounts, or even enterprise accounts if they are not “paid-tier” and provided by Tenstorrent.

See the AI Tool Use Policy @ Tenstorrent. Please confirm any doubts with the Legal team.

Chat demo architecture

Overview:

- Leverages multiple open-source components together with our software and hardware
- Default model: Meta-Llama-3.3-70B-Instruct running on top of TT Metalium
- Running on multiple TT LoudBox scaled in Kubernetes
- Served with LLM inference engine vLLM, supporting Continuous Batching and PagedAttention
- Uses Automatic Prefix Caching and sticky sessions for efficient KV cache re-use over multiple requests
- Provides an OpenAI-compatible API
- LiteLLM proxy used to manage API keys
- Open WebUI front-end, to which we are making open-source contributions to enable sticky sessions
- SSO authentication with MS Entra



In the drop-down menu of the Open WebUI, you can select from different models we provide.

- `tenstorrent/Meta-Llama-3.3-70B-Instruct` - served in TT Cloud on LoudBoxes (default model)
- `tenstorrent/DeepSeek-R1-Distill-Llama-70B` - served in TT Cloud on LoudBoxes
- `azure/gpt-4o` - routed to Azure Cloud payed by TT
- ... other models are often being added

Making API calls

You can make API calls to the proxy and use any of these models. You should use the following URL:

```
BASE_URL="https://litellm-proxy--tenstorrent.workload.tenstorrent.com"
```

You will also need an API key for the proxy. Ask the Cloud team if you don't have a key.

Example with curl

As a simple example using curl, this should work in your terminal. This will return the available models:

```
API_KEY="XXXXXXXXXXXXXXXXX"
BASE_URL="https://litellm-proxy--tenstorrent.workload.tenstorrent.com"

curl --location ${BASE_URL}/models \
  --header "Content-Type: application/json" \
  --header "Authorization: Bearer ${API_KEY}"
```

This is an example of chatting:

```
API_KEY="XXXXXXXXXXXXXXXXX"
BASE_URL="https://litellm-proxy--tenstorrent.workload.tenstorrent.com"

curl --location ${BASE_URL}/chat/completions \
    --header 'Content-Type: application/json' \
    --header "Authorization: Bearer ${API_KEY}" \
    --data '{
    "model": "azure/gpt-4o",
    "messages": [
        {
        "role": "user",
        "content": "What is the purpose of life?"
        }
     ]
 }'
```

Example with Python

If you create a `.env` file with your credentials defined,

```
API_KEY="XXXXXXXXXXXXXXXXX"
BASE_URL="https://litellm-proxy--tenstorrent.workload.tenstorrent.com"
MODEL="tenstorrent/Meta-Llama-3.3-70B-Instruct"
```

you should be able to create a python script like the following toy example.

```python
from dotenv import load_dotenv
import os
from openai import OpenAI

load_dotenv()

API_KEY = os.getenv("API_KEY")
BASE_URL = os.getenv("BASE_URL")
MODEL = os.getenv("MODEL")

def main():
    client = OpenAI(api_key=API_KEY, base_url=BASE_URL)

    response = client.chat.completions.create(
        messages=[
            {
                "role": "user",
                "content": "How's it going?",
            }
        ],
        model=MODEL,
        stream=True,
    )

    full_content = []
    for chunk in response:
        chunk_content = chunk.choices[0].delta.content
        if chunk_content:
            print(chunk_content, end="", flush=True)
            full_content.append(chunk_content)

    content = "".join(full_content)
    return content


if __name__ == "__main__":
    main()
```

A more complete but simple example of a chat client for the terminal is here: openai-chat-client.py

Using with Claude Code

Claude Code docs can be found here: Claude Code overview - Anthropic

On most distributions, you can find Claude Code at

```
/tools_vendor/FOSS/claude-code/stable/bin/claude
```

If not preset, you will need to install it following the steps below.

Installing Claude Code:

You will need to ensure that you have `node` version 18 or higher.

If you are on the AUS login servers, you can add the following to your `PATH`:

```
/tools_vendor/FOSS/node/22.14.0/bin
```

Or use some variation of this method:

```
## install node.js locally
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.3/install.sh | bash
\. "$HOME/.nvm/nvm.sh"
nvm install 24
```

Then you can install:

```
# if you are on the AUS login servers, you will not be able to install trully globally
# so you should set the global prefix to somewhere in your home dir as follows
mkdir -p ~/.npm-global/lib
npm config set prefix '~/.npm-global'
# then add ~/.npm-global/bin to your PATH

# install claude code itself
npm install -g @anthropic-ai/claude-code
```

Configuring Claude Code

Because we are using Claude through our LiteLLM proxy instead of directly from Anthropic or AWS Bedrock, we will need to setup a custom endpoint. This can be done via exported environment variables or via the `~/.claude/settings.json`.

The `settings.json` route is probably the right way to go, but I will provide both methods here for reference.

> In both methods, the `ANTHROPIC_AUTH_TOKEN` variable will be the same as your tt-chat API key, contact Ryan Reece on Slack if you need one.

There are a ton of variables and settings to play around with. See the docs here: Claude Code settings - Anthropic

To setup using CI variables:

```
# set the api URL to tenstorrent's LiteLLM proxy
export ANTHROPIC_BASE_URL="https://litellm-proxy--tenstorrent.workload.tenstorrent.com"
# set the model name, since our proxy adds the "anthropic/" prefix to the model names
export ANTHROPIC_MODEL="anthropic/claude-sonnet-4-20250514" # this can also be set to opus if you're feeling luxurious
# set the small+fast+cheap model for background tasks (this saves some money on inference when a powerful model is not need)
export ANTHROPIC_SMALL_FAST_MODEL="anthropic/claude-3-5-haiku-20241022"
# enable interleaved thinking via custom headers, allows for models to think after toolcalls
# this setting is also used by cursor
export ANTHROPIC_CUSTOM_HEADERS="anthropic-beta: interleaved-thinking-2025-05-14"
# set the auth key
export ANTHROPIC_AUTH_TOKEN=sk-<token>
# or as follows if you have your token saved to a file
export ANTHROPIC_AUTH_TOKEN=$(cat ~/.tt_ai_key)
# run claude
cd /path/to/your/project
claude
```

You can also set these in your .bashrc/.zshrc somewhere.

To setup using `~/.claude/settings.json`:

```
# you need to make sure these files and directories exist if you haven't invoked claude code before
 mkdir -p ~/.claude
 touch ~/.claude/settings.json
```

Then edit `settings.json` with your favorite editor and paste in the following config:

```
{
  "apiKeyHelper": "~/.tt-get-claude-key.sh",
  "env": {
    "ANTHROPIC_BASE_URL": "https://litellm-proxy--tenstorrent.workload.tenstorrent.com",
    "ANTHROPIC_MODEL": "anthropic/claude-sonnet-4-20250514",
    "ANTHROPIC_SMALL_FAST_MODEL": "anthropic/claude-3-5-haiku-20241022",
    "ANTHROPIC_CUSTOM_HEADERS": "anthropic-beta: interleaved-thinking-2025-05-14",

    "DISABLE_TELEMETRY": "true"
  },
  "includeCoAuthoredBy": false,
  "model": "anthropic/claude-sonnet-4-20250514"
}
```

The `apiKeyHelper` field allows you to provide a script that provides the API key to Claude dynamically. Optionally, you can add the following KV pair to the `env` block to simply hardcode the API key.

```
"ANTHROPIC_AUTH_TOKEN" : "<token>"
```

You can run the following shell commands to generate your api key helper script.

```
echo "$ANTHROPIC_AUTH_TOKEN" > ~/.tt_ai_key
echo "cat \$HOME/.tt_ai_key" > ~/.tt-get-claude-key.sh
chmod 700 ~/.tt-get-claude-key.sh
```

You can then proceed to launch claude-code via

```
cd /path/to/your/project
claude
```

Claude Code Integrations

VSCode (and its forks – Cursor, Windsurf)

ClaudeCode integrates with VSCode and VSCode forks via the the Claude Code plugin.

See the docs here: Add Claude Code to your IDE - Anthropic

To get going with claude code in vscode/cursor:

1. Open the integrated terminal at the bottom of the window
    a. Make sure your PATH is up to date with the changes made above
2. Invoke `claude` within the integrated terminal
    a. In theory, this should automatically prompt you to install the ClaudeCode plugin
3. If the official ClaudeCode plugin isn't install automatically, install it yourself from here https://marketplace.visualstudio.com/items?itemName=anthropic.claude-code

This integration will allow claude code to show you the changes its making to source files via a diff view, and more.

Neovim

claude-code.nvim

Emacs

claude-code.el

IntelliJ

Claude Code IntelliJ

IDE integrations

Many text editors and IDEs have plugins for chat assistants.

- Notes by Todd Yamakawa on Using Copilot with Neovim
- Note that this actually doesn't using anything from TT-Chat; It uses copilot.

Ryan's summary: How to setup copilot for vim (Copilot is not served in our cloud):

```
# On MacOS
brew install vim
brew install node

# check that your vim version is >= 9.0.0185
vim --version

git clone https://github.com/github/copilot.vim.git \
  ~/.vim/pack/github/start/copilot.vim

vim
# Then run :Copilot setup
```