Jetson MLC-LLM

These steps are a distillation of my journey to a proven, working build and deployment. It took me a while and several reinstalls to find the right flow for a smooth experience. This guide will provide you with everything to help you achieve:

- Building TVM from source on your Jetson
- Building MLC-LLM from source on your Jetson
- Converting a model from Hugging Face to MLC format and running it from command line and/or using a Python script

Target Audience

- **Jetson Orin Nano users** who want to run LLMs locally.
- **Developers** comfortable with Linux/command line.
- **Hobbyists** experimenting with edge AI.
- **Not for**: Cloud-only users or those without NVMe storage.

Assumptions:

- Jetpack 6.2 is installed
- An NVME is installed, formatted to EXT4 and partitioned. Make sure it's compatible with the Jetson and preferrably low power.

If you need an NVME drive, WD SN530 is a good choice as it's low power, cheap and has plenty of diskspace!

0. Setup

0.1. Mount NVME drive

- Note: *In this step you decide what your NVME drive is mounted as. I use `/mnt/nvme`, some people prefer `/mnt/ssd`. Throughout this guide, `/mnt/nvme` is used. It's perfectly safe to replace this with your own name, but do this consistently for every instance*
- Note2: *Replace [user] in the chown command with your username*
- Note3: Verify your NVME is `/dev/nvme0n1p1` with this command: `lsblk`. Make sure the next script uses the correct URI.

```
sudo mkdir /mnt/nvme
sudo mount /dev/nvme0n1p1 /mnt/nvme
sudo chown -R [user]:[user] /mnt/nvme
sudo chmod -R 755 /mnt/nvme
echo '/dev/nvme0n1p1 /mnt/nvme ext4 defaults 0 2' | sudo tee -a /etc/fstab
```

0.2. Add paths to .bashrc

During the installation steps, several modules and dependencies are installed manually, or via APT. In the end, your `.bashrc` should have these lines added at the end of the file. We can add them as a first step to make sure you always have the paths and variables available to make the installation smoother :-)

```
export TVM_HOME=/mnt/nvme/tvm
export PYTHONPATH=/mnt/nvme/tvm/python:$PYTHONPATH
export PATH=/usr/local/cuda/bin:/usr/lib/llvm-17/bin:~/.local/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/cuda/lib64:/usr/lib/llvm-17/lib:/lib/aarch64-linux-gnu:$LD_LIBRARY_PATH
```

Activate your environment: `source ~/.bashrc`

1. Swap space configuration

- Allocate 32G of swap on it
- Update fstab to make permanent
- Disable swapfiles on microSD card

1.1. Create swap file on NVME

```
sudo fallocate -l 32G /mnt/nvme/swapfile  # Create an 32GB file on NVMe
sudo chmod 600 /mnt/nvme/swapfile  # 600 ensures only root can read/write the swap file for security.
sudo mkswap /mnt/nvme/swapfile  # Format as swap
sudo swapon /mnt/nvme/swapfile  # Enable swap
echo '/mnt/nvme/swapfile none swap sw 0 0' | sudo tee -a /etc/fstab
```

1.2. Disable ZRAM

The Jetson is configured to have some swap files on the microSD card. We're using the NVME for that because it's a lot faster. This step is optional, but recommended to reduce read/write operations on your microSD card.

```
sudo systemctl disable nvzramconfig
```

Verify with the following command (the zram swap files will stil show up until after next reboot):

```
swapon --show
free -h
```

You can run `sudo reboot now` if you want to fully deactivate the zram, but it's not required.

2. Install general modules

First, we'll install everything we need in order to proceed. In this guide, I don't use a Python virtual environment to make things less confusing for new users. We'll install the dependency anyway if you prefer to use it.

```
sudo apt-get update
sudo apt-get install -y python3.10 python3.10-venv python3-pip libopenblas-dev libtinfo-dev
```

3. TVM cloning and building

A compatible TVM build **requires** LLVM-17, which isn't available via apt-get for the Jetson. We'll start by downloading and installing the binary before building TVM.

3.1. Install LLVM-17

In this step, you will get the binary, unpack it and move it to `/usr/lib/llvm-17`. It's a large file, so it might take a while depending on your network connection.

```
wget https://github.com/llvm/llvm-project/releases/download/llvmorg-17.0.6/clang+llvm-17.0.6-aarch64-linux-gnu.tar.xz
tar xf clang+llvm-17.0.6-aarch64-linux-gnu.tar.xz
sudo mv clang+llvm-17.0.6-aarch64-linux-gnu /usr/lib/llvm-17
```

3.2. Build TVM

These steps, you will pull the TVM source, configure the build and build your own TVM. We'll start off with cloning the source from GitHub:

```
cd /mnt/nvme/
git clone --recursive https://github.com/apache/tvm.git && cd tvm
```

Clean up the build folder and copy a fresh config.cmake to it (If you ever want/need to make a fresh build, this place is where you can start off):

```
rm -rf build && mkdir build && cd build
cp ../cmake/config.cmake .
```

At the time of writing, the commit hash used was `bfb0dd6a161d33c58f67469988244b139366e063`. If issues arise during step 3.2.2, try to use this hash and try again. If compilation still fails, there might be something wrong in your environment; retrace your steps and verify everything is installed correctly

3.2.1. Configure build

Inside the `/mnt/nvme/tvm/build` directory, we have a `config.cmake` file that is used to set flags; use the following to setup the config to use CUDA, Setup the correct CUDA architecture and use recommended settings for MLC-LLM. You can copy/paste this entire block and run it. The lines that start with '-' will be printed to your console and can be ignored; I added those as a means of commenting what everything is for.

Note for Jetson Users:

- Jetson Orin Nano uses **CUDA Architecture 87** (`Ampere`).
- Jetson AGX Orin uses **87 + 89** (if you have both GPU clusters).
- Verify your architecture with:

```
nvcc --version
```

Note: We enable CUTLASS here for TVM, but MLC-LLM will disable it during its build (see Section 4.3). I'm still looking for a way to enable CUTLASS with MLC-LLM, and enabling it now will save us some time in the future.

```
echo - LLVM is a must dependency
echo "set(USE_LLVM \"llvm-config --ignore-libllvm --link-static\")" >> config.cmake
echo "set(HIDE_PRIVATE_SYMBOLS ON)" >> config.cmake
echo - GPU SDKs, turn on if needed
echo "set(USE_CUDA   ON)" >> config.cmake
echo "set(USE_ROCM   OFF)" >> config.cmake
echo "set(USE_METAL  OFF)" >> config.cmake
echo "set(USE_VULKAN OFF)" >> config.cmake
echo "set(USE_OPENCL OFF)" >> config.cmake
echo - Below are options for CUDA, turn on if needed
echo - CUDA_ARCH is the cuda compute capability of your GPU.
echo - Examples: 89 for 4090, 90a for H100/H200, 100a for B200.
echo - Reference: https://developer.nvidia.com/cuda-gpus
echo "set(CMAKE_CUDA_ARCHITECTURES 87)" >> config.cmake
echo "set(USE_CUBLAS ON)" >> config.cmake
echo "set(USE_CUTLASS ON)" >> config.cmake
echo "set(USE_THRUST ON)" >> config.cmake
echo "set(USE_NVTX OFF)" >> config.cmake
echo "set(USE_HIPBLAS OFF)" >> config.cmake
echo - Below is the option for ROCM, turn on if needed
echo "set(USE_RPC ON)" >> config.cmake
echo "set(USE_GRAPH_EXECUTOR ON)" >> config.cmake
```

3.2.2. Compile and install

Now that we have the source code and the build configuration ready, it's time to build. This takes a while (15-20 minutes) on the Jetson. The following command is ran inside `/mnt/nvme/tvm/build`, you're probably still in this folder:

```
cmake -DLLVM_CONFIG=/usr/lib/llvm-17/bin/llvm-config .. && make -j$(nproc) && cd ..
```

TVM relies on tvm-ffi, so we need to install that aswell:

```
pip3 install setuptools_scm
pip3 install ninja
pip3 install cython
pip3 install psutil
cd /mnt/nvme/tvm/3rdparty/tvm-ffi
pip3 install --user --no-build-isolation .
```

3.2.3. Verify build

Check if we have all modules, the following command should show `libtvm_allvisible.so`, `libtvm_runtime.so` and `libtvm.so`:

```
cd /mnt/nvme/tvm/build && ls -lh libtvm*
```

3.2.3.1. Confirm that TVM is properly installed as a python package and provide the location of the TVM python package

```
python3 -c "import tvm; print(tvm.__file__)"
```

should print something like `/some-path/lib/python3.13/site-packages/tvm/__init__.py`

3.2.3.2. Verify build flags

```
python3 -c "import tvm; print('\n'.join(f'{k}: {v}' for k, v in tvm.support.libinfo().items()))"
```

should print the build flags we've set in 3.2.1

3.2.3.3. Verify CUDA

```
python3 -c "import tvm; print(tvm.cuda().exist)"
```

should print `True`

4. MLC-LLM build

With TVM built and verified, we can move to MLC-LLM!

4.1. Install Rust

Rust will be installed in this step, it will add itself to your `~/.bashrc` aswell (`. "$HOME/.cargo/env"` will be added) and enable itself with the source command.

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
source ~/.bashrc
rustup default stable
```

4.2. Clone MLC-LLM

Clone the MLC-LLM source code:

```
cd /mnt/nvme/
git clone --recursive https://github.com/mlc-ai/mlc-llm.git
cd mlc-llm
mkdir -p build && cd build
```

4.3. Configure build settings for MLC-LLM

In this step, we're creating a build configuration file. Make sure you're inside `/mnt/nvme/mlc-llm/build`.

When running gen_cmake_config, a couple of questions are asked. Make sure you reply with the following answers:

- TVM_SOURCE_DIR: Your TVM home directory (`/mnt/nvme/tvm` in this guide)
- Use CUDA: `y`

The rest of the answers should be answered with `n`. **We've built TVM with CUTLASS enabled and we're not using it here**. CUTLASS (CUDA Templates for Linear Algebra Subroutines) in a nutshell speeds up linear algebra operations which are basically the heart of transformer models. On the Jetson Orin Nano with its limited VRAM, I was unable to build MLC-LLM with CUTLASS enabled and chose to disable it. If, in the future I manage to build MLC-LLM with CUTLASS enabled, I will update this guide, but on the Jetsons' Ampere architecture, it can cause some instability and doesn't really speed things up too much (we'll use cuBLAS instead as it gives a tiny boost without instability issues).

```
python3 ../cmake/gen_cmake_config.py

Enter TVM_SOURCE_DIR in absolute path. If not specified, 3rdparty/tvm will be used by default: /mnt/nvme/tvm
Use CUDA? (y/n): y
Use CUTLASS? (y/n): n
Use CUBLAS? (y/n): y
Use ROCm? (y/n): n
Use Vulkan? (y/n): n
Use Metal (Apple M1/M2 GPU) ? (y/n): n
Use OpenCL? (y/n) n
```

4.4. Build MLC-LLM

In this step, we're going to build MLC-LLM. There's 1 thing we need to do though, and that is to relax the flashinfer version as it's set to use **exactly** 0.4.0 right now. This version isn't available via pip3, so we'll use another one by doing the following:

4.4.1. Update flashinfer dependency version

```
cd /mnt/nvme/mlc-llm/python
```

Edit requirements.txt and change `flashinfer-python==0.4.0` to `flashinfer-python>=0.4.0`

Mandatory disclaimer: Using >=0.4.0 may introduce compatibility issues. If you encounter errors, try flashinfer-python==0.4.0 with a manual wheel install. At the time of writing though, it worked perfectly!

4.4.2. Install flashinfer

```
pip3 install flashinfer-python
pip3 install -e .
```

4.4.3. Build MLC-LLM

```
cd /mnt/nvme/mlc-llm/build/
cmake .. && make -j $(nproc) && cd ..
```

4.4.4. Verify build

Verify whether MLC-LLM is installed correctly by running:

```
python3 -c "import mlc_llm; print(mlc_llm.__version__)"
```

Should print something like `0.1.dev0`

```
python3 -c "from mlc_llm import cuda; print(cuda().exist)"
```

Should print `True`

Once you've reached this place, you're all set!

5. Troubleshooting

Common Issues and Fixes

Issue	Likely Cause	Solution
`cmake` fails with CUDA errors	Incorrect `CMAKE_CUDA_ARCHITECTURES`	Set to `87` for Jetson Orin Nano.
`pip3 install` fails	Missing dependencies	Run `sudo apt-get install -y libopenmpi-dev libssl-dev`.
`import tvm` fails	Python paths not set	Verify `PYTHONPATH` in `.bashrc`.
Out of memory during build	Insufficient swap	Verify `free -h` shows 32G swap.
CUTLASS errors in MLC-LLM	CUTLASS enabled in MLC-LLM	Re-run `gen_cmake_config.py` with `n`.

6. Starting a model

To start a model there are multiple options. In this guide, we'll explore two different options:

- Directly via command line using `mlc_llm`
- Via Python

Models for MLC need to be converted for MLC. MLC-AI provides models that are ready to download and use. Models are large and we'll need to install `git-lfs` first:

```
sudo apt-get install git-lfs
git lfs install
```

6.2 Starting from command line

To start a model from command line, use this command (Note the order of parameters, they matter!).

```
mlc_llm chat --device cuda:0 --overrides "context_window_size=1024;prefill_chunk_size=512" HF://mlc-ai/Hermes-3-Llama-3.2-3B-q4f16_1
```

This will download the model from HuggingFace, compile it into a module and run it. The first run will take a while, but you'll be greeted with a prompt! The context_window_size and prefill_chunk_size are values that work well within the boundaries of my Jetson Orin Nano, feel free to play with these values and see how far you can push your device :-)

Below are some tests with different models, some are smaller (less parameters) and some are bigger. Try to find some small models and play with them to see which model clicks with you :-) If the chat becomes super slow, the context limit reached. The Python script in the next chapter manages the context for us and that will fix it.

6.2.1. Test with Hermes-3-Llama-3.2-3B-q4f16_1-MLC

```
mlc_llm chat --device cuda:0 --overrides "context_window_size=1024;prefill_chunk_size=512" HF://mlc-ai/Hermes-3-Llama-3.2-3B-q4f16_1
```

```
>>> Hi there!
Greetings! I am Hermes 3, a sentient AI designed to assist you in any way I can. How may I be of service today?
>>> Can you tell me about polar bears?
Certainly. Polar bears are the largest land carnivores and are the only bear species native to the Arctic. They are well adapted to

Polar bears are excellent swimmers and spend much of their time in the water. Their large paws are also adapted for walking on ice.

They are top predators and play a critical role in the Arctic ecosystem. However, they are facing threats from climate change, loss

I hope this information is helpful to you. Let me know if you have any other questions.
>>> /stats
prefill: 90.5 tok/s, decode: 31.6 tok/s
```

6.2.2. Test with Llama-3-8B-Instruct-q4f16_1-MLC

```
mlc_llm chat --device cuda:0 --overrides "context_window_size=1024;prefill_chunk_size=512" HF://mlc-ai/Llama-3-8B-Instruct-q4f16_1-M
```

```
>>> Hi there!
Hello! It's nice to meet you. Is there something I can help you with, or would you like to chat for a bit? I'm here to assist you wi
>>> Can you tell me about polar bears?
Polar bears! They're such magnificent creatures. Here are some fun facts about them:

* Polar bears (Ursus maritimus) are the largest land carnivores on Earth, with adult males weighing up to 1,700 pounds (770 kg) and
* They have two layers of fur: a thick undercoat and a longer, guard hair layer that helps to reflect sunlight and keep them warm in
* Polar bears are expert swimmers and have been known to swim for hours or even days at a time. They can swim up to 60 miles (97 kil
* Their diet consists mainly of seals, which they hunt by waiting at the edge of breathing holes or stalking them on the ice. They a
* Polar bears have an excellent sense of smell, which they use to detect seals and other prey. They can even detect the scent of a s
* Despite their size, polar bears are incredibly agile and can run at speeds of up to 25 miles per hour (40 kilometers per hour) whe
* Unfortunately, polar bears are vulnerable to climate change, as the melting of sea ice reduces their habitat and makes it harder f

I hope you found these facts interesting! Do you have any other questions about polar bears?
>>> /stats
prefill: 48.0 tok/s, decode: 14.4 tok/s
```

6.2.3. Test with Mistral-7B-Instruct-v0.3-q4f16_1-MLC

```
mlc_llm chat --device cuda:0 --overrides "context_window_size=1024;prefill_chunk_size=512" HF://mlc-ai/Mistral-7B-Instruct-v0.3-q4f1
```

```
>>> Hi there!
Hello! I'm here to help you in the best way possible, with care, respect, and truth. I strive to ensure that our conversation is use

Please note that while I aim to provide accurate and helpful responses, I'm an AI model, and there might be instances where I don't

Lastly, I'm programmed to continue learning, improving, and evolving, so your feedback and suggestions are essential to me. If there

Enjoy our conversation! 😊
>>> Can you tell me about polar bears?
Polar bears (Ursus maritimus) are fascinating creatures, often associated with the icy Arctic and subarctic regions of the Northern

Polar bears live primarily on a diet of seals, which they hunt using their powerful senses, swimming abilities, and sharp claws. Occ

The polar bear population is divided into 19 subpopulations, some of which are facing significant threat from climate change. Meltin

Here's an interesting fact: Polar bears have a black, slit-like structure in their eyes called a nictitating membrane. Essentially,

If you have more questions about polar bears or any other topic, feel free to ask, and I'm happy to help! ��
>>> /stats
prefill: 29.0 tok/s, decode: 10.6 tok/s
```

6.3 Starting using a Python script

In this chapter we'll explore downloading a model, converting it ourselves and running it via Python. This will allow you to try models outside of the mlc-ai model zoo and even give you a small performance boost (as the model is compiled using optimal parameters). The Python script will manage the context aswell, this will make the model 'forget' older messages, but it'll keep the context well within the limits (You might have ran into the LLM becoming slow all of a sudden (or crashing with an error), this is the context limit that got reached).

6.3.1. Download a model

```
mkdir -p /mnt/nvme/models/ && cd git clone https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.3
```

This will take a while, git-lfs is fetching large files and there's no feedback. Use `top` to see if there's a git process working!

6.3.2 Convert model weights

Here we're creating the MLC model. Take special note of the folder names; the MLC one has an `-MLC` postfix. This can be anything but I like to keep the original model name as reference. You can play with quantization here aswell, but q4f16_1 is a nice value to keep RAM usage low and keep good quality.

Quantization Options:

- `q4f16_1` : Best balance of speed/quality (~4.2GB VRAM for 7B).
- `q4f32_1` : Higher quality, but slower (~5GB VRAM).
- `q3f16_1` : Faster (~~15 tok/s), but lower quality (~~3.5GB VRAM).

```
mlc_llm convert_weight /mnt/nvme/models/Mistral-7B-Instruct-v0.3/ --quantization q4f16_1 -o /mnt/nvme/models/Mistral-7B-Instruct-v0.
```

6.3.3 Generate model config

Use gen_config to create a configuration for the model we just created and note the folders (the `-MLC` folder is the output location). The value for quantization must be the same value we used in the previous step.

The command below has two parameter values that are specific to Mistral models: `--conv-template mistral_default` and `--model-type mistral` . For Llama3 models, we would use `--conv-template llama-3` and `--model-type llama-3` .

```
mlc_llm gen_config /mnt/nvme/models/Mistral-7B-Instruct-v0.3/ --quantization q4f16_1 --conv-template mistral_default --model-type mi
```

We now have a `mlc-chat-config.json` inside `/mnt/nvme/models/Mistral-7B-Instruct-v0.3-MLC/` . Since the default values are too big for the Jetson, we'll edit it. Change all occurrences of the following values (usually only 2):

```
"context_window_size": 1024,      // Defaulted to 32768 (too big for Jetson!)
"prefill_chunk_size": 512,        // Defaulted to 8192 (too large)
"max_batch_size": 1,              // Defaulted to 128 (too high)
"sliding_window_size": 1024,      // Optional, but recommended
"temperature": 0.7,               // Default is 1.0 (too random)
"top_p": 0.9,                     // Default is 1.0 (no filtering)
"repetition_penalty": 1.1         // Default is 1.0 (no penalty)
```

In this example, sliding_window_size is a value that's supported by Mistral. If your model doesn't support it, leave it at `-1`

6.3.4. Compile to model_lib

Last step: compile the MLC model into a module! cudagraph=1 Enables CUDA Graphs for lower latency at the cost of slightly more RAM usage.

```
mkdir /mnt/nvme/libs

mlc_llm compile /mnt/nvme/models/Mistral-7B-Instruct-v0.3-MLC/mlc-chat-config.json \
  --device cuda \
  --opt "cudagraph=1" \
  -o /mnt/nvme/libs/Mistral-7B-Instruct-v0.3-cuda.so
```

6.3.5. Create and run script to interact with the model!

Save this script as a `.py` file:

```
#!/usr/bin/env python3
import argparse
import threading
import time
import sys
import os
from itertools import cycle
from typing import List, Dict, Any
from mlc_llm import MLCEngine
from mlc_llm.serve.config import EngineConfig

# Constants
DEFAULT_MODEL_PATH = "/mnt/nvme/models/Mistral-7B-Instruct-v0.3-MLC"
DEFAULT_LIB_PATH = "/mnt/nvme/libs/Mistral-7B-Instruct-v0.3-cuda.so"
TEMPERATURE = 0.3
TOP_P = 0.9
CONTEXT_LIMIT = 1000

def spinner(stop_event: threading.Event) -> None:
    """Display a loading spinner until stop_event is set."""
    for c in cycle(['|', '/', '-', '\\']):
        if stop_event.is_set():
            break
        sys.stdout.write(f'\rLoading model {c}')
        sys.stdout.flush()
        time.sleep(0.1)

def main() -> None:
    parser = argparse.ArgumentParser()
    parser.add_argument("--model", default=DEFAULT_MODEL_PATH)
    parser.add_argument("--lib", default=DEFAULT_LIB_PATH)
    args = parser.parse_args()

    # Validate paths
    if not os.path.exists(args.model):
        print(f"Error: Model path does not exist: {args.model}")
        sys.exit(1)
    if not os.path.exists(args.lib):
        print(f"Error: Library path does not exist: {args.lib}")
        sys.exit(1)

    print("Loading model...")
    stop_event = threading.Event()
    spinner_thread = threading.Thread(target=spinner, args=(stop_event,))
    spinner_thread.start()

    try:
        engine_config = EngineConfig()
        engine = MLCEngine(
            model=args.model,
            model_lib=args.lib,
            engine_config=engine_config,
        )
    except Exception as e:
        print(f"\nFailed to load model: {e}")
        stop_event.set()
        spinner_thread.join()
        sys.exit(1)

    stop_event.set()
    spinner_thread.join()
    print("\rModel loaded!          ")

    messages: List[Dict[str, str]] = [
            {"role": "system", "content":"""
        Always assist with care, respect, and truth. Respond with utmost utility yet securely.
        Avoid harmful, unethical, prejudiced, or negative content. Ensure replies promote fairness and positivity.
        """}
    ]

    running = True
    while running:
        try:
            user_input = input("\nUser: ")
        except EOFError:
            print("\nGoodbye!")
            break

        if user_input.lower() in ["quit", "exit", "bye"]:
            print("Goodbye!")
            running = False
        else:
            messages.append({"role": "user", "content": user_input})
            print("Assistant: ", end="", flush=True)
            full_response = ""
            token_count = 0
            start_time = time.time()

            try:
                response = engine.chat.completions.create(
                    messages=messages,
                    temperature=TEMPERATURE,
                    top_p=TOP_P,
                    stream=True,
                    stop=["<|im_end|>", "</s>"], # Correct for Mistral
                )
                for chunk in response:
                    delta = chunk.choices[0].delta
                    if delta.content and delta.content.strip():
                        print(delta.content, end="", flush=True)
                        full_response += delta.content
                        token_count += 1

                input_tokens = len(user_input.split())
                if input_tokens + token_count > CONTEXT_LIMIT:
                        print("\nWarning: Approaching context limit! Resetting history.")
                        messages = messages[:1]

                except Exception as e:
                    print(f"\nError: {e}")
                    messages = messages[:1]
                    continue

                print(f"\nTokens: {token_count} | Speed: {token_count / max(time.time() - start_time, 0.1):.1f} tok/s")
                messages.append({"role": "assistant", "content": full_response})
                messages = messages[:1] + messages[-2:]

        print("\nDone.")
        sys.exit(0)

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

6.3.6. Run the model and have fun!

Run using `python3 <filename>.py`

The output looks different from the command line, this is because we're not really doing much formatting, but the performance should be better. 3 tokens per second might not look like much, but it's about 30% increased for the same model :-)

```
 User: Hi there!
Assistant: Hello! I'm here to help you with information, answer questions, and engage in friendly and respectful conversations. I'm
Tokens: 67 | Speed: 13.2 tok/s

User: Can you tell me about polar bears?
Assistant: Polar bears (Ursus maritimus) are the largest land carnivores on Earth. They live primarily in the Arctic regions, and th
Tokens: 478 | Speed: 13.7 tok/s
```

7. Troubleshooting

Common Issues

Issue	Fix
`RuntimeError: CUDA out of memory`	Reduce `context_window_size` to 512.
`FileNotFoundError: model_lib`	Re-run `mlc_llm compile`.
Slow prefill (~5 tok/s)	Set `prefill_chunk_size=256`.
Weird tokens (e.g., `[INST]`)	Check `--conv-template` for your model.