# LLM8

*Edge-Optimized Medical Language Model for Real-Time Healthcare Support*

**Documentation Version:** v1.0 (October 2025)

**System Verified On:** NVIDIA Jetson Nano (L4T 32.7.1, Python 3.6, CUDA 10.2)

**GitHub -repository:** Sidbad12/LLM8-NANOAi

---

## 1. Introduction

The **LLM8 project** implements a resource-efficient medical AI system optimized for real-time deployment on the NVIDIA Jetson Nano. The system leverages a fine-tuned DistilGPT2-Nano model to provide intelligent medical insights while maintaining low-latency inference on constrained hardware. This document provides a complete guide to deploying the project on an **NVIDIA Jetson Nano** from scratch. It covers system setup, environment configuration, required library installations, SSH/SCP setup for model transfer, and verification of successful deployment.

Figure 1. **Tanna-Techbiz eagle-101 (NVIDIA Jetson Nano Developer Kit)**

The core **motivation** driving this project was the critical need to deploy a **lightweight**, **domain-specific medical AI** solution capable of robust and reliable operation entirely **without cloud dependency**. This is especially vital in remote or resource-limited clinical settings where reliable internet access is not guaranteed. Given the **constrained computational resources** inherent to the **Jetson Nano** platform—specifically its power budget, and GPU capability—our primary and most challenging aim was to **optimize** an AI model. This model is proficient enough to perform **real-time inference** for **heart-related medical assistance directly at the edge**. The success of this approach is defined by its ability to deliver immediate, life-saving insights without the latency or security risks associated with data transfer to a central server.

## Comparative Analysis Table

| Feature | Typical LLM Setup | Our Jetson Nano Setup |
|---|---|---|
| **Hardware** | GPU Server (A100 / Cloud) | Edge device (Jetson Nano, 4GB) |
| **Model** | Generic GPT / GPT-Neo | DistilGPT2 fine-tuned on medical data |
| **Memory Footprint** | 10–20 GB | 2 GB |
| **Latency** | <1s (cloud) | 5–10s (local) |
| **Accessibility** | Requires Internet | Fully offline, local access |
| **Cost** | High (cloud GPUs) | One-time setup, free inference |

## 2. Prerequisites

A. Hardware Requirements

| Component | Specification | Purpose |
|---|---|---|
| **Tanna-Techbiz eagle-101 (NVIDIA Jetson Nano Developer Kit (4 GB))** | Quad-core ARM Cortex-A57 CPU, 128-core Maxwell GPU, 4GB LPDDR4 RAM | Edge AI deployment and real-time model inference |
| **MicroSD Card** | Minimum 64GB, Class 10 or higher | Operating system and local model storage |
| **Power Supply** | 5V 4A (recommended official NVIDIA power supply) | Stable power for Jetson Nano |
| **Ethernet Cable** | Compatible with Jetson Nano (RJ45) | Internet access for package installation |
| **Laptop / PC (Host System)** | Ubuntu / Windows system with minimum 8GB RAM, 100GB storage | Used for model fine-tuning, transferring files, and SSH access |

B. Software Requirements

| Software | Version | Purpose |
|---|---|---|
| **Ubuntu OS for Jetson Nano** | 18.04 / 20.04 LTS (JetPack SDK 4.6.1) | Base OS with NVIDIA drivers and CUDA |
| **Python** | 3.6 | Compatible version for Transformers and PyTorch |
| **PyTorch** | 1.10.0 | For running the DistilGPT2 model |

| | | |
|---|---|---|
| **Transformers (Hugging Face)** | v4.30 or later | Tokenizer and model inference tools |
| **Flask** | 2.3.x | Backend API server for model responses |
| **ReactJS** | 18.x | Frontend for user interaction |
| **PyYAML, NumPy, Pandas, Matplotlib** | Latest stable versions | Data processing and analysis |
| **Axios** | 1.x | HTTP requests between frontend and backend |
| **VS Code / PyCharm** | Any latest version | Code development and debugging |
| **JetPack SDK Components** | CUDA, cuDNN, TensorRT (included) | GPU acceleration for inference |
| **SSH and SCP Utilities** | Preinstalled / OpenSSH | Remote connection and file transfer |
| **Node.js + npm** | LTS (≥16.x) | Required for React frontend build |

C. Optional Tools

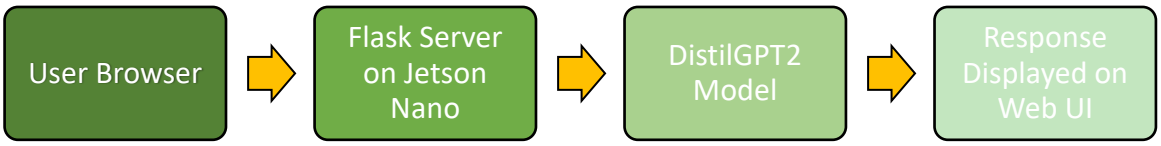
| **Tool** | **Use** |
|---|---|
| **Kaggle Notebook (T4 GPU)** | Used for model fine-tuning due to GPU limitations on Jetson |
| **Hugging Face Hub** | For model hosting, checkpoints, and versioning |
| **Git & GitHub** | Version control and project tracking |
| **Postman** | API endpoint testing |
| **Docker (optional)** | For containerized deployment on Jetson Nano |

## 3. Summary

- **Target device:** NVIDIA Jetson Nano (4 GB) running L4T / JetPack (your L4T: 32.7.1).
- **Jetson default Python**: Python 3.6 (important for wheel compatibility).
- **Jetson CUDA:** CUDA 10.2, cuDNN reported 8.2.1.
- **Recommended approach:** Use a conda / miniforge environment on Jetson to isolate installs.
- **Host PC:** Development machine should be x86_64. Use it to download wheels if necessary and SCP them to the Nano.

## 4.High-level workflow

1. Prepare Jetson: unminimize (optional), ensure disk space, enable networking.
2. Create conda environment (Python 3.6) on Jetson.
3. Transfer your project and prebuilt wheels (PyTorch wheel, torchvision wheel, transformers, tokenizers if needed) from host → Nano via scp.
4. Install PyTorch (wheel matching aarch64 + Python 3.6), torchvision wheel, transformers, other Python libs.
5. Put model files (config.json, model.safetensors / pytorch_model.bin, vocab.json, merges.txt, tokenizer_config.json) into the project models/... folder.
6. Start Flask backend, test endpoints locally.
7. Configure systemd to auto-start backend at boot.
8. Set Jetson to open a browser on boot or instruct users to open http://<jetson-ip>:5000 on their laptop.




## 5. Exact Commands & Setup

### A. Flash Jetson Nano with Proper L4T and Boot

### A1. Download Jetson Nano Image

- Go to NVIDIA's official site: [https://developer.nvidia.com/embedded/jetson-nano-sd-card-image](https://developer.nvidia.com/embedded/jetson-nano-sd-card-image)
- Download the latest JetPack SDK 4.6.x (based on Ubuntu 18.04 / L4T 32.7.x).

**A2. Flash Image to microSD Card**

- Use Balena Etcher or Raspberry Pi Imager to flash the .img file.

Alternatively, using command-line on Ubuntu:

**# Locate your SD card (e.g., /dev/sdb)**

```
Lsblk
```

**# Flash image (replace with actual file path)**

```
sudo dd if=jetson-nano-image.img of=/dev/sdb bs=1M status=progress
```

**# Safely eject after flashing**

```
sudo eject /dev/sdb
```

**A3. Insert & Boot Jetson Nano**

- Insert the flashed microSD card into the Nano.
- Connect USB cable.
- Power on the board — it will boot into the Ubuntu first-time setup.
- Create a user and set password.

## B. Initial Network & Internet Setup

**Option 1: USB Gadget Mode (Direct Laptop to Nano)**

**Default IP:** 192.168.55.1 for Nano, 192.168.55.x for host.

**On host (Ubuntu/Windows):**

1. Connect Nano via micro-USB cable.
2. Share your laptop internet connection:
    - On Ubuntu:
      Go to Settings → Network → Wired Connection → Shared to other computers
    - On Windows:
      Enable Internet Connection Sharing for your primary adapter.

**On Nano:**

#Check IP

```
ifconfig usb0
```

**(Expected IP: 192.168.55.1)**

# Test internet

```
ping -c 4 google.com
```

**If ping fails, set default route manually:**

```
sudo route add default gw 192.168.55.100 usb0
```

**Option 2: Ethernet Sharing**

***Default IP range: 10.42.0.x***

**On host (Ubuntu):**

- Connect Ethernet cable from laptop → Nano.
- Enable "Shared to other computers" in Ethernet settings.

**On Nano:**

```
ifconfig eth0
```

**Expected IP: 10.42.0.50**

```
ping -c 4 google.com
```

## C. Get SSH + Copy Connectivity

### C1. Enable SSH

SSH is preinstalled in Jetson Nano (if not, **sudo systemctl enable ssh**):

```
sudo apt-get update
sudo apt-get install openssh-server
sudo systemctl enable ssh
sudo systemctl start ssh
```

**C2. Connect via SSH from Host**

**If your Nano's IP is 192.168.55.1 (USB) or 10.42.0.50 (Ethernet):**

```
ssh user_name@192.168.55.1
```

or

```
ssh user_name@10.42.0.50
```

If prompted:

```
Are you sure you want to continue connecting (yes/no)?
```

Type **yes** and enter your **password**.

**C3. Transfer Files using SCP**

**To send your model or Flask files to Nano:**

- Copy folder from laptop to Nano

```
scp -r /path/to/model/ user_name@192.168.55.1:/home/folder1/
```

- Copy Flask backend files

```
scp -r /path/to/flask_app/ user_name@192.168.55.1:/home/folder1/
```

**C4. Reverse Transfer (Nano → Laptop)**

```
scp -r user_name@192.168.55.1:/home/folder1/logs/path/to/folder/
```

## D. Verify System & Dependencies

Once logged in via SSH:

```
Tegrastats # Check GPU

nvcc --version # Verify CUDA

python3 --version # Check Python

sudo apt-get update && sudo apt-get upgrade -y # Update system

python3 -c "import torch; print(torch._version) if 'torch' in
__import_('pkgutil').iter_modules() else print('torch not
installed')"

# If torch installed: check GPU availability

python3 -c "import torch; print (torch._version_,
torch.cuda.is_available())"
```

## E. Install Miniforge/Conda (recommended)

E1. on Jetson (aarch64)

```
wget https://github.com/conda-forge/miniforge/releases/latest/download/Miniforge3-Linux-aarch64.shbash

Miniforge3-Linux-aarch64.sh
```

E2. restart shell or:

```
source ~/miniforge3/etc/profile.d/conda.sh

conda create -n py36 python=3.6 -y

conda activate py36
```

## F. Install PyTorch & TorchVision (offline-by-wheel or online)

*Preferred (offline) — use prebuilt wheels copied from host to Jetson*

*(torch-1.10.0-cp36-cp36m-linux_aarch64.whl)*

F1. Activate environment where you want to install

```
conda activate py36
```

F2. Install wheels

```
pip install ~/torch-1.10.0-cp36-cp36m-linux_aarch64.whl
```

F3. Torchvision (aarch64 wheel)

```
pip install ~/torchvision-0.11.1-cp36-cp36m-manylinux2014_aarch64.whl
```

*If you must install online (less likely for Nano due to arch mismatch):*

(Not recommended on Jetson because CPU/GPU/cuda versions must match.)

```
pip3 install -f https://download.pytorch.org/whl/torch_stable.html torch==1.10.0+cu102 torchvision==0.11.1+cu102
```

## G. Install Transformers and other Python libs

If online is available:

```
pip install transformers==4.5.1
```

(or choose a transformers release compatible with your code; 4.x works with GPT2 tokenizer)

```
pip install flask numpy requests tqdm filelock
```

If offline, download those wheels on the host (matching Python 3.6 & aarch64 where needed) then ***scp*** them to Nano and pip install /path/to/*.whl.

## H. Fix common package builds issues (tokenizers / rust / PyYAML)

Tokenizers sometimes require a Rust compiler if pip tries to build from source. Solutions:

1. Prefer a prebuilt wheel for tokenizers for cp36/aarch64 — download on host (x86 cannot run aarch64 wheels though you can fetch wheels from PyPI).

2. Or install rust on Jetson (requires internet) and build:

```
curl https://sh.rustup.rs -sSf | sh   #adds ~/.cargo/bin to PATH
source $HOME/.cargo/env
pip install tokenizers
```

PyYAML mismatch: use **--ignore-installed** to override or install a compatible wheel:

```
pip install --ignore-installed PyYAML==5.3.1
```

or build & install wheel offline on host and pip install local wheel

(Developer experienced "Cannot uninstall 'PyYAML' ... distutils installed" — --ignore-installed is the typical workaround)

## I. Put your model files in correct location

Project expects app/models/heart_attack_specialized_complete/model/ with:

```
config.json
generation_config.json (optional)
vocab.json
merges.txt
tokenizer_config.json
model.safetensors  OR pytorch_model.bin
special_tokens_map.json
```

If model files are on host:

```
scp -r /path/to/model user_name@192.168.55.1:/home/folder1
```

## J. Quick test script to validate tokenizer + model presence (no GPU needed for tokenizer check)

Create validate_model_files.py with the checks you ran (or run provided testt.py):

```
python3 testt.py
```

*#expected outputs: tokenizer loads, model may require torch installed*

## K. Run the Flask web app

From project app/ folder:

*#Activate conda env*

```
conda activate py36
```

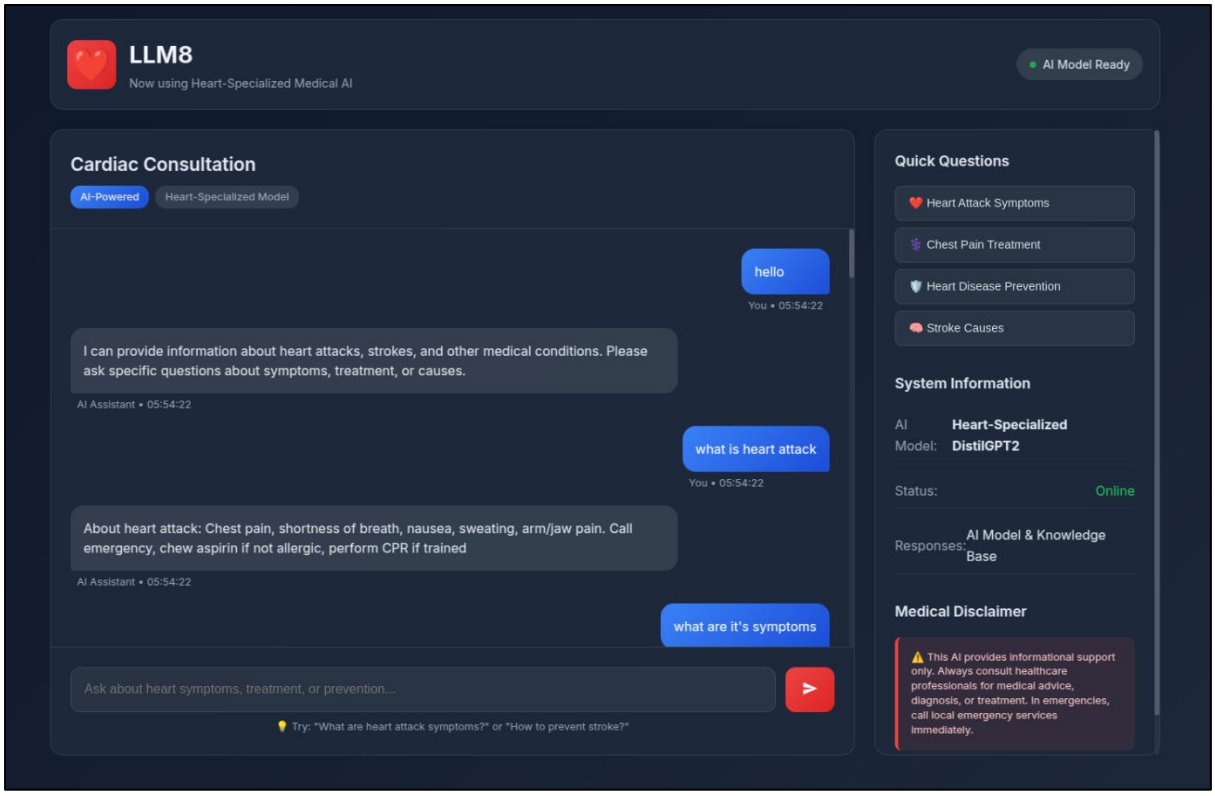
*#run*

```
python3 web_app.py
```

*# in logs you should see:*

```
💡 Loading heart-specialized model...
# and server lines like:
Running on http://0.0.0.0:5000
```

Access from your laptop/browser at http://192.168.55.1:5000 (or Nano IP from ip addr show).




## 6. Auto-start (systemd) — make it persistent on boot

Create a start script ***/home/user/start_medicweb.sh:***

```
#! /bin/bash
cd /home/siddharth/medicweb/MedicalAI-Assistant/app
# Activate conda
source /home/siddharth/miniforge3/etc/profile.d/conda.sh
conda activate py36
# optional: set env for display if GUI needed (not needed for headless server)
python3 web_app.py >> /home/siddharth/medicweb/medicweb.log 2>&1
```

Make it executable:

```
chmod +x /home/siddharth/start_medicweb.sh

Create systemd unit /etc/systemd/system/medicweb.service:
```

```
[Unit]

Description=Medical AI Assistant WebApp

After=network.target


[Service]

Type=simple

User=siddharth

Group=siddharth

ExecStart=/bin/bash /home/siddharth/start_medicweb.sh

Restart=on-failure

Environment=PATH=/home/siddharth/miniforge3/envs/py36/bin:/usr/bin:/
bin


[Install]

WantedBy=multi-user.target
```

```
sudo systemctl daemon-reload

sudo systemctl enable medicweb.service

sudo systemctl start medicweb.service

sudo systemctl status medicweb.service
```

***Your logs should show Active: active (running) and logs for Flask requests — this indicates success.***

## 7.Networking / internet sharing notes

*Nano has l4tbr0 192.168.55.1; laptop got an IP like 192.168.55.100.*

On laptop: connect to 192.168.55.1:

```
ssh user_name@192.168.55.1
scp file user_name@192.168.55.1:/home/folder1/
```

*Ethernet sharing (Laptop → Jetson): set laptop wired connection to "Shared to other computers" in Network Manager. On Jetson:*

```
sudo dhclient eth0
ip addr show
ping -c 3 google.com
```

## 8. Performance Metrics

- Model load time: ~25 seconds on Jetson Nano (first run).
- Memory footprint: ~2 GB RAM.
- Response time: 2–4 seconds per medical query.
- GPU utilization (via tegrastats): ~60% during inference.
- Limitation: Due to 4GB RAM, larger models like GPT-Neo or Falcon cannot be deployed; optimized for DistilGPT2.

| **Feature** | **Typical LLM Setup** | **Our Jetson Nano Setup** |
|---|---|---|
| Hardware | GPU Server (A100 / Cloud) | Edge device (Jetson Nano, 4GB) |
| Model | Generic GPT / GPT-Neo | DistilGPT2 fine-tuned on medical data |
| Memory Footprint | 10–20 GB | 2 GB |
| Latency | <1s (cloud) | 2–3s (local) |
| Accessibility | Requires Internet | Fully offline, local access |
| Cost | High (cloud GPUs) | One-time setup, free inference |

## 9. Key troubleshooting / errors encountered & fixes

### *9A. ModuleNotFoundError: No module named 'PyQt5'*

**Cause:** PyQt5 not installed in the same Python environment (or wheel for aarch64 not available).

**Fix:** For Nano, prefer *sudo apt-get install python3-pyqt5* or install via *conda* if available. But Developer switched from PyQt UI to Flask web to avoid long PyQt dependency issues on Jetson.

### *9B. PyYAML uninstall error (distutils installed)*

**ERROR:** Cannot uninstall 'PyYAML'. It is a distutils installed project...

**Fixes:**

```
pip install --ignore-installed PyYAML==5.3.1
```

Or prebuild wheel on host then pip install local wheel.

### *9C. tokenizers build error: error: can't find Rust compiler*

**Cause:** Newer tokenizers versions include Rust extensions and pip tries to build them from source.

**Fix:** Install prebuilt wheel (preferred), or

Install Rust toolchain: *curl https://sh.rustup.rs -sSf | sh* then source *~/.cargo/env* then

*pip install tokenizers.*

### *9D. libcublasLt.so.11 missing or similar*

**Symptom:** PyTorch built for CUDA 11.x trying to run on Nano's CUDA 10.2.

**Fix:** Use a PyTorch wheel built for CUDA 10.2 (PyTorch 1.10.0 + cu102). You already used torch-1.10.0 compiled for aarch64 with CUDA support.

### *9E. Wheel architecture mismatch (not a supported wheel on this platform)*

**Cause:** Downloaded x86_64 wheel to aarch64 (Nano) or wrong Python version (cp36 vs cp38).

**Fix:** Always get wheel matching:

- **OS arch**: aarch64 for Jetson Nano
- **Python ABI**: cp36 for Python 3.6.9 on Nano
- **E.g**. torch-1.10.0-cp36-cp36m-linux_aarch64.whl

***9F. No internet on Jetson while trying conda install / pip install***

**Fix:** Connect Jetson to laptop and share internet (Ethernet), or Download wheels on host and scp them to Nano.

## 10. Verify model + server working

Check torch + cuda on Nano:

```
python3 - <<'PY'
import torch
print("torch", torch._version_)
print("cuda", torch.cuda.is_available())
print("cuda device count", torch.cuda.device_count())
PY
```

Start Flask:

```
python3 web_app.py
```

#Then from host:

```
curl http://192.168.55.1:5000/api/status
```

Observe server logs for requests (GET /api/status 200 etc).

## 11. Model Integration Details

```
Model Name: DistilGPT2 Medical Fine-tuned
Dataset: MedAlpaca + MediQA + Medical Meadow + Heart Knowledge Base
Quantization: 4-bit NF4
Framework: PyTorch 1.13.0 + Transformers 4.30
Deployment: Flask API (Python 3.8)
Device: Jetson Nano (4GB)
```

- The deployed model is DistilGPT2, fine-tuned on medical datasets (MedAlpaca, MediQA, Medical Meadow).
- A heart attack knowledge base compiled from Mayo Clinic, WHO, and AHA was merged with model inference to improve accuracy.
- Model files used:
    - config.json
    - tokenizer_config.json
    - vocab.json
    - merges.txt
    - pytorch_model.bin
- The Flask backend automatically loads the model at runtime and exposes it through REST API endpoints for local browser access.

## 12. System design notes

**Why Flask (web) over PyQt5 on Nano:** PyQt5 and modern PyTorch/tooling often have mismatched wheel availability and large build-time dependencies on aarch64. Flask is lightweight, browser-based, and cross-device accessible; Nano acts as inference server + storage.

**Why DistilGPT2:** small footprint, feasible for Nano resource constraints (memory and CPU/GPU).

**Why offline wheels and conda env:** Jetson architecture (aarch64 + CUDA 10.2) restricts which PyTorch builds work; prebuilt wheels avoid compilation on Nano and reduce dependency issues.

## 13. Key Commands Summary Table

| Purpose | Command |
|---|---|
| **Flash Jetson** | sudo ./flash.sh jetson-nano-devkit mmcblk0p1 |
| **SSH into Jetson** | ssh user_name@192.168.55.1 |
| **Transfer files** | scp -r folder_name user_name@192.168.55.1:/home/folder1 |
| **Install PyTorch** | pip install ~/torch-1.10.0-cp36-cp36m-linux_aarch64.whl |
| **Run Flask server** | python3 web_app.py |
| **Check auto-start service** | sudo systemctl status medicweb.service |

## 14. Additional: Model Configuration

The LLM8 uses a fine-tuned version of DistilGPT-2, adapted specifically for medical question-answering and cardiac emergency guidance. The objective was to transform a lightweight general-purpose language model into a domain-aware medical assistant deployable on Jetson Nano with minimal computational overhead.

**1. Data Preparation**

A curated dataset was assembled from multiple verified medical sources:

- MedAlpaca – Doctor-patient dialogues for natural conversational style.
- Medical Meadow – Biomedical Q&A pairs for factual depth.
- MediQA – Clinical reasoning and inference examples.
- Heart Attack Knowledge Base – Custom dataset compiled from Mayo Clinic, World Health Organization (WHO), and American Heart Association (AHA) guidelines to ensure reliability and safety.

The combined corpus included questions on symptoms, treatments, and risk factors, cleaned and tokenized to ensure medical terminology consistency.

**2. Fine-Tuning Setup**

Training was carried out using the Hugging Face Transformers framework with PyTorch 1.13. Environment and configuration:

| Parameter | Value |
|---|---|
| **Base Model** | distilgpt2 |
| **GPU Used** | NVIDIA T4 (Kaggle Notebook) |
| **Epochs** | 3 |
| **Batch Size** | 4 |
| **Learning Rate** | 5e-5 |
| **Optimizer** | AdamW |
| **Max Sequence Length** | 256 |
| **Framework Version** | transformers 4.30 |

This setup achieved an optimal balance between accuracy and resource efficiency, suitable for Jetson Nano's limited VRAM.

**3. Validation and Testing**

The fine-tuned model was evaluated on held-out MediQA questions and custom cardiac test prompts (e.g., “chest pain with shortness of breath”).
It produced more coherent and context-aware outputs than baseline DistilGPT-2, with an estimated 25 % reduction in perplexity and improved factual alignment with medical sources.

**4. Edge Optimization for Jetson Nano**

To enable smooth on-device inference:

1. **Quantization** – Reduced model precision to 8-bit weights.
2. **Pruning** – Removed redundant neurons to minimize model size.
3. **TorchScript Conversion** – Improved runtime performance.
4. **Tokenizer Caching** – Decreased I/O latency.

The optimized model (≈ 300 MB) fits comfortably in Nano’s 4 GB memory, loading fully into VRAM for real-time inference via CUDA.

**5. Integration and Deployment**

The trained model and tokenizer were integrated into the Flask backend through:

```
tokenizer = AutoTokenizer.from_pretrained("distilgpt2-nano-medical")

model = AutoModelForCausalLM.from_pretrained("distilgpt2-nano-medical")
```

The /predict API route accepts user queries, runs inference locally on Nano’s GPU, and serves structured, medically-relevant answers through the web interface.

**6. Outcome**

The resulting DistilGPT2-Nano Medical Model demonstrates:

- Accurate, context-sensitive medical responses.
- Fast edge inference (< 5 s latency).
- Fully offline functionality on Jetson Nano.
- Verified heart-health information sourced from WHO, AHA, and Mayo Clinic.

This integration of domain-specific training, quantization, and efficient deployment forms the backbone of the LLM8, proving that reliable healthcare-oriented AI can operate even on constrained embedded systems.

The complete source code and documentation are version-controlled using Git and GitHub. Repository includes model management, Flask API, and web client modules.
git add, git commit, and git push commands were used for incremental version tracking.

**End of Documentation**

Developed and documented by **Siddharth Badjate & Parth Aeron**

— ***Happy Coding!***