nanochat-enterprise

Day 1:

Andrej Karpathy created the repo nanochat. Here, our objective is to build nanochat for enterprises. The following conversation was exchanged with the Open Agent Manager in Antigravity.

Developer: Our objective is to build nanochat-enterprise, an AI-powered chat application for enterprise use. Currently all executions are happening from speedrun.sh, but we want to move to a more enterprise-ready setup. We would like to build a FAST API controller, could you help us with that?

AI: That's great! I'm glad the new FastAPI controller meets your requirements.

The server is up and running in the background. You can now use the Enterprise Nanochat API to manage your training jobs remotely or locally with better process isolation.

If you need any help extending the API with more features (like real-time websocket logs or user authentication) or integrating it with a frontend, just let me know!

Happy coding! 🚀

speedrun .sh

```
#!/bin/bash
echo
# This script is the "Best ChatGPT clone that $100 can buy",
# It is designed to run in ~4 hours on 8XH100 node at $3/GPU/hour.

# 1) Example launch (simplest):
# bash speedrun.sh
# 2) Example launch in a screen session (because the run takes ~4 hours):
# screen -L -Logfile speedrun.log -S speedrun bash speedrun.sh
# 3) Example launch with wandb logging, but see below for setting up wandb
first:
# WANDB_RUN=speedrun screen -L -Logfile speedrun.log -S speedrun bash
speedrun.sh

# Default intermediate artifacts directory is in ~/.cache/nanochat
export OMP_NUM_THREADS=1
export NANOCHAT_BASE_DIR="$HOME/.cache/nanochat"
mkdir -p $NANOCHAT_BASE_DIR

#
---------------------------------------------------------------------------
---
# Python venv setup with uv

# install uv (if not already installed)
command -v uv &> /dev/null || curl -LsSf https://astral.sh/uv/install.sh |
sh
# create a .venv local virtual environment (if it doesn't exist)
[ -d ".venv" ] || uv venv
# install the repo dependencies
uv sync --extra gpu
# activate venv so that `python` uses the project's venv instead of system
python
source .venv/Scripts/activate
```

```
#
-----------------------------------------------------------------------
---
# wandb setup
# If you wish to use wandb for logging (it's nice!, recommended).
# 1) Make sure to first log in to wandb, e.g. run:
#    `wandb login`
# 2) Set the WANDB_RUN environment variable when running this script,
e.g.:
#    `WANDB_RUN=d26 bash speedrun.sh`
if [ -z "$WANDB_RUN" ]; then
    # by default use "dummy" : it's handled as a special case, skips
logging to wandb
    WANDB_RUN=dummy
fi

#
-----------------------------------------------------------------------
---
# During the course of the run, we will be writing markdown reports to the
report/
# directory in the base dir. This command clears it out and writes a
header section
# with a bunch of system info and a timestamp that marks the start of the
run.
python -m nanochat.report reset

#
-----------------------------------------------------------------------
---
# Tokenizer

# Download the first ~2B characters of pretraining dataset
# look at dev/repackage_data_reference.py for details on how this data was
prepared
# each data shard is ~250M chars
# so we download 2e9 / 250e6 = 8 data shards at this point
# each shard is ~100MB of text (compressed), so this is about ~800MB of
data on disk
python -m nanochat.dataset -n 8
```

```
# Immediately also kick off downloading more shards in the background
while tokenizer trains
# See comment below for why 240 is the right number here
python -m nanochat.dataset -n 240 &
DATASET_DOWNLOAD_PID=$!
# train the tokenizer with vocab size 2**16 = 65536 on ~2B characters of
data
python -m scripts.tok_train --max_chars=2000000000 --vocab_size=65536
# evaluate the tokenizer (report compression ratio etc.)
python -m scripts.tok_eval

#
------------------------------------------------------------------------
---
# Base model (pretraining)

# The d20 model is 561M parameters.
# Chinchilla says #tokens = 20X #params, so we need 561e6 * 20 = 11.2B
tokens.
# Assume our tokenizer is 4.8 chars/token, this is 11.2B * 4.8 ~= 54B
chars.
# At 250M chars/shard, this is 54B / 250M ~= 216 shards needed for
pretraining.
# Round up to 240 for safety. At ~100MB/shard, this downloads ~24GB of
data to disk.
# (The total number of shards available in the entire dataset is 1822.)
echo "Waiting for dataset download to complete..."
wait $DATASET_DOWNLOAD_PID

# Number of processes/GPUs to use
NPROC_PER_NODE=8

# pretrain the d20 model
torchrun --standalone --nproc_per_node=$NPROC_PER_NODE -m
scripts.base_train -- --depth=20 --target_param_data_ratio=20
--run=$WANDB_RUN
# evaluate the model on a larger chunk of train/val data and draw some
samples
torchrun --standalone --nproc_per_node=$NPROC_PER_NODE -m
scripts.base_loss
```

```
# evaluate the model on CORE tasks
torchrun --standalone --nproc_per_node=$NPROC_PER_NODE -m
scripts.base_eval

#
-----------------------------------------------------------------------
---
# Midtraining (teach the model conversation special tokens, tool use,
multiple choice)

# download 2.3MB of synthetic identity conversations to impart a
personality to nanochat
# see dev/gen_synthetic_data.py for details on how this data was prepared
and to get a sense of how you can easily tune it
curl -L -o $NANOCHAT_BASE_DIR/identity_conversations.jsonl
https://karpathy-public.s3.us-west-2.amazonaws.com/identity_conversations.
jsonl

# run midtraining and eval the model
torchrun --standalone --nproc_per_node=$NPROC_PER_NODE -m
scripts.mid_train -- --run=$WANDB_RUN
torchrun --standalone --nproc_per_node=$NPROC_PER_NODE -m
scripts.chat_eval -- -i mid

#
-----------------------------------------------------------------------
---
# Supervised Finetuning (domain adaptation to each sequence all by itself
per row)

# train sft and re-eval right away (should see a small bump)
torchrun --standalone --nproc_per_node=$NPROC_PER_NODE -m scripts.chat_sft
-- --run=$WANDB_RUN
torchrun --standalone --nproc_per_node=$NPROC_PER_NODE -m
scripts.chat_eval -- -i sft

# chat with the model over CLI! Leave out the -p to chat interactively
# python -m scripts.chat_cli -p "Why is the sky blue?"

# even better, chat with your model over a pretty WebUI ChatGPT style
```

```
# python -m scripts.chat_web

#
-----------------------------------------------------------------------
---
# Reinforcement Learning. Optional, and currently only on GSM8K
# (optional)

# run reinforcement learning
# torchrun --standalone --nproc_per_node=$NPROC_PER_NODE -m
scripts.chat_rl -- --run=$WANDB_RUN
# eval the RL model only on GSM8K
# torchrun --standalone --nproc_per_node=$NPROC_PER_NODE -m
scripts.chat_eval -- -i rl -a GSM8K

#
-----------------------------------------------------------------------
---
# Generate the full report by putting together all the sections
# report.md is the output and will be copied to current directory for
convenience
python -m nanochat.report generate
```

Antigravity Execution

Here is the list of files it created.

- Server
  - api.py
  - config.py
  - main.py
  - manager.py

In manager, it created two classes: Job and JobManager.
In config, it created environment variables.
In main, it initialized FastAPI; it added CORS Middleware; and also health check.
In api, it created the following methods:

- `@router.post("/jobs/setup")`
- `@router.post("/jobs/tokenizer")`
- `@router.post("/jobs/pretrain")`
- `@router.post("/jobs/midtrain")`
- `@router.post("/jobs/sft")`
- `@router.get("/jobs/{job_id}")`
- `@router.get("/jobs")`

Starting the server

```
.\.venv\Scripts\python -m uvicorn server.main:app --host 127.0.0.1 --port 8000
--reload
```

Breakdown of the command:

- `.\.venv\Scripts\python`: Uses the Python executable from your local virtual environment ensuring all dependencies are found.
- `-m uvicorn`: Runs the Uvicorn web server module.
- `server.main:app`: points to the `app` object in server/main.py.
- `--reload`: (Optional) Auto-reloads the server if you make code changes (useful for development).