**Repository structure**

hacker-news-gpt/

│

├─ mainrun/ # main Python package

│ ├─ \_\_init\_\_.py

│ ├─ config/ # configuration files

│ │ ├─ \_\_init\_\_.py

│ │ ├─ hyperparams.yaml

│ │ └─ training.yaml

│ ├─ data/ # dataset handling

│ │ ├─ \_\_init\_\_.py

│ │ └─ dataset.py

│ ├─ model/ # models

│ │ ├─ \_\_init\_\_.py

│ │ ├─ gpt.py # your GPT model

│ │ └─ base.py # base classes for future models

│ ├─ utils/ # utility functions

│ │ ├─ \_\_init\_\_.py

│ │ └─ tokenizer.py

│ ├─ train.py # main training loop

│ ├─ train\_helper.py # helper functions (e.g., get\_batch, evaluate)

│ └─ logs.py # logging configuration

│

├─ notebooks/ # optional: Jupyter notebooks for analysis

├─ scripts/ # CLI or data preprocessing scripts

├─ requirements.txt

├─ README.md

└─ setup.py # optional, if you want pip installable package

**2️⃣ Configuration using OmegaConf**

* Keep all hyperparameters and training settings in **YAML** files under config/.
* Example:

**hyperparams.yaml**

hyperparams:

seed: 1337

epochs: 7

block\_size: 128

batch\_size: 64

vocab\_size: 16000

n\_layer: 6

n\_head: 8

d\_model: 512

dropout: 0.1

lr: 6e-3

weight\_decay: 0.0

val\_frac: 0.1

Then in train.py:

from omegaconf import OmegaConf

cfg = OmegaConf.load("mainrun/config/hyperparams.yaml")

hparams = OmegaConf.to\_container(cfg.hyperparams, resolve=True)

**3️⃣ Current task**

* **Dataset:** julien040/hacker-news-posts
* **Goal:** Predict the next token in a list of post titles.
* **Pipeline:**
  1. Load titles → train\_titles, val\_titles
  2. Tokenize using BPE tokenizer
  3. Concatenate titles with <eos> as separator
  4. Encode into token IDs → train\_ids, val\_ids
  5. Train GPT model to predict **next token** at each position

**4️⃣ Future extensions**

* Add more models under model/ (e.g., Transformer-XL, LSTM).
* Each model inherits from a **base class** (base.py) to standardize forward and generate.
* Config for each model can be separate YAML files (e.g., config/gpt.yaml, config/lstm.yaml).
* Logging, evaluation, and batch preparation can stay generic in train\_helper.py.

💡 This structure makes it easy to:

* Swap or add models
* Experiment with hyperparameters via YAML
* Keep code modular and maintainable
* Integrate with W&B or other logging systems

{

"lr": {

"value": 0.006

},

"seed": {

"value": 1337

},

"\_wandb": {

"value": {

"e": {

"bhi0ro2op8rjx3bgiyxya707rna2pyvj": {

"os": "Linux-5.15.167.4-microsoft-standard-WSL2-aarch64-with-glibc2.35",

"git": {

"commit": "c89d8a937e669cc9c88914a6eb4aa8988b90ad88",

"remote": "https://github.com/tedasdf/mainrun.git"

},

"args": [

"--block\_size",

"64",

"--lr",

"0.006",

"--n\_layer",

"4",

"--dropout",

"0.1",

"--weight\_decay",

"0.1",

"--d\_model",

"128",

"--batch\_size",

"16"

],

"disk": {

"/": {

"used": "5911773184",

"total": "1081101176832"

}

},

"host": "73f795ed57cd",

"root": "/workspace/mainrun",

"email": "teedsingyau@gmail.com",

"memory": {

"total": "8198082560"

},

"python": "CPython 3.10.12",

"program": "/workspace/mainrun/train.py",

"codePath": "mainrun/train.py",

"writerId": "bhi0ro2op8rjx3bgiyxya707rna2pyvj",

"cpu\_count": 10,

"startedAt": "2025-08-27T10:28:34.182379Z",

"executable": "/usr/bin/python3",

"codePathLocal": "train.py",

"cpu\_count\_logical": 10

}

},

"m": [],

"t": {

"1": [

1,

49,

51

],

"2": [

1,

49,

51

],

"3": [

16

],

"4": "3.10.12",

"5": "0.21.1",

"12": "0.21.1",

"13": "linux-aarch64"

},

"code\_path": "code/mainrun/train.py",

"cli\_version": "0.21.1",

"python\_version": "3.10.12"

}

},

"epochs": {

"value": 7

},

"n\_head": {

"value": 8

},

"d\_model": {

"value": 512

},

"dropout": {

"value": 0.1

},

"n\_layer": {

"value": 6

},

"log\_file": {

"value": "./logs/mainrun.log"

},

"val\_frac": {

"value": 0.1

},

"batch\_size": {

"value": 64

},

"block\_size": {

"value": 128

},

"num\_titles": {

"value": 100000

},

"vocab\_size": {

"value": 16000

},

"weight\_decay": {

"value": 0

},

"evals\_per\_epoch": {

"value": 3

},

"model\_arhitecture": {

"value": "gpt"

}

}

* Set up github action to connect to a GPU workstation for streamlined production
* And automatically update the code in the GPU workstation
* Plus git commit logs automatically

Research report :

ON the geometry of semantics in Next token prediction :

* reveals the NTP optimization implicitly guides models to encode concepts via singular value decomposition factos of a centered data -sparsity matrix that captures next word co-ocucurance pattenrs
* We find that the most important SVD factors are learned first during training, motivating using spectral clustering of embeddings to identify human-interpretable semantics, including both classical k-means and a new orthant-based method directly motivated by our interpretation of concepts.
* . Overall, our work bridges distributional semantics, neural collapse geometry, and neural network training dynamics, providing insights into how NTP’s implicit biases shape the emergence of meaning representations in language models.
* Geometric emergence of latent concepts: We demonstrate that latent linguistic concepts learned by a large neural model through NTP emerge as principal components in the singular value decomposition of a centered data-sparsity matrix ˜S
  + So structure naturally shapes the geometry of learned representations : concepts can be recovered as weighted combination of word and context embeddings
  + the characterization of the unconstrained-features model (UFM) by Zhao et al. (2024), revealing how embeddings inherently organize around latent concept dimensions.

Background:

TART : Token-based Architecture Transformer for Neural Network Performance Prediction

ALPHA Zip: Neural Network-Enhanced Lossless Text Compression

Introduces a lossless text compression approach using a Large Language Model .

Two key steps:

* prediction using a dense neural network architecture
* compressing the predicted ranks with standard compression algorithms

1. by applying a compression algorithm over the outputs from the transformer block
2. compresses the text to a binary bit stream
3. show by utilising accelerated linear algebra complication combined with an optimal neural model size