```
from google.colab import files
import zipfile, os

# Upload your MLPColab.zip manually from local machine
uploaded = files.upload()  # choose MLPColab.zip

# Extract
zip_path = "MLPColab.zip"
extract_dir = "/content/MLPColab"
os.makedirs(extract_dir, exist_ok=True)

with zipfile.ZipFile(zip_path, "r") as zf:
    zf.extractall(extract_dir)

os.chdir(extract_dir)
print("✅ Files extracted to:", os.getcwd())
!ls -R | head -40
```

Choose files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving MLPColab.zip to MLPColab.zip
✅ Files extracted to: /content/MLPColab
.:
MLPColab

./MLPColab:
data_prep.py
dataset.py
README.md
requirements.txt
streamlit_app.py
train_mlp.py
utils.py
visualize_embeddings.py
```

```
!pip install torch torchvision tqdm streamlit pyngrok
```

```
Requirement already satisfied: torch in /usr/local/lib/python3.12/dist-p
Requirement already satisfied: torchvision in /usr/local/lib/python3.12/
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-pa
Collecting streamlit
  Downloading streamlit-1.51.0-py3-none-any.whl.metadata (9.5 kB)
Collecting pyngrok
  Downloading pyngrok-7.4.1-py3-none-any.whl.metadata (8.1 kB)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dis
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/l
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/c
Requirement already satisfied: sympy>=1.13.3 in /usr/local/lib/python3.1
Requirement already satisfied: networkx in /usr/local/lib/python3.12/dis
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-
Requirement already satisfied: fsspec in /usr/local/lib/python3.12/dist-
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in /usr/l
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in /usr
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in /usr/l
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in /usr/loca
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in /usr/loca
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in /usr/local
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in /usr/loc
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in /usr/lo
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in /usr/lo
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in /usr/loc
Requirement already satisfied: nvidia-nccl-cu12==2.27.3 in /usr/local/li
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in /usr/local/l
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in /usr/lo
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in /usr/loca
Requirement already satisfied: triton==3.4.0 in /usr/local/lib/python3.1
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-p
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/p
Requirement already satisfied: altair!=5.4.0,!=5.4.1,<6,>=4.0 in /usr/lo
Requirement already satisfied: blinker<2,>=1.5.0 in /usr/local/lib/pytho
Requirement already satisfied: cachetools<7,>=4.0 in /usr/local/lib/pyth
Requirement already satisfied: click<9,>=7.0 in /usr/local/lib/python3.1
Requirement already satisfied: packaging<26,>=20 in /usr/local/lib/pytho
Requirement already satisfied: pandas<3,>=1.4.0 in /usr/local/lib/python
Requirement already satisfied: protobuf<7,>=3.20 in /usr/local/lib/pytho
Requirement already satisfied: pyarrow<22,>=7.0 in /usr/local/lib/python
Requirement already satisfied: requests<3,>=2.27 in /usr/local/lib/pytho
Requirement already satisfied: tenacity<10,>=8.1.0 in /usr/local/lib/pyt
Requirement already satisfied: toml<2,>=0.10.1 in /usr/local/lib/python3
Requirement already satisfied: watchdog<7,>=2.1.5 in /usr/local/lib/pyth
Requirement already satisfied: gitpython!=3.1.19,<4,>=3.0.7 in /usr/loca
Collecting pydeck<1,>=0.8.0b4 (from streamlit)
  Downloading pydeck-0.9.1-py2.py3-none-any.whl.metadata (4.1 kB)
Requirement already satisfied: tornado!=6.5.0,<7,>=6.0.3 in /usr/local/l
Requirement already satisfied: PyYAML>=5.1 in /usr/local/lib/python3.12/
Requirement already satisfied: jsonschema>=3.0 in /usr/local/lib/python3
Requirement already satisfied: narwhals>=1.14.2 in /usr/local/lib/python
Requirement already satisfied: gitdb<5,>=4.0.1 in /usr/local/lib/python3
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/li
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12
```

```
import os, json, requests, numpy as np
from tqdm import tqdm

base_data = "data"
os.makedirs(base_data, exist_ok=True)

datasets = {
    "warpeace": "https://raw.githubusercontent.com/karpathy/char-rnn/master/da
    "linux": "https://raw.githubusercontent.com/karpathy/char-rnn/master/data/
}
```

```
for name, url in datasets.items():
    ddir = os.path.join(base_data, name)
    os.makedirs(ddir, exist_ok=True)

    txt_path = os.path.join(ddir, "input.txt")
    if not os.path.exists(txt_path):
        print(f"⬇️ Downloading {name} dataset...")
        txt = requests.get(url).text
        open(txt_path, "w", encoding="utf-8").write(txt)

    # Build vocab and encoded files (robust fallback)
    text = open(txt_path, "r", encoding="utf-8").read()
    vocab = sorted(list(set(text)))
    stoi = {ch:i for i,ch in enumerate(vocab)}
    itos = {i:ch for ch,i in stoi.items()}
    json.dump({"stoi": stoi, "itos": itos}, open(os.path.join(ddir, "vocab.jso

    data = np.array([stoi[ch] for ch in text], dtype=np.uint16)
    split = int(0.9 * len(data))
    np.save(os.path.join(ddir, "train.npy"), data[:split])
    np.save(os.path.join(ddir, "val.npy"), data[split:])
print("✅ Data ready.")
```

```
⬇️ Downloading warpeace dataset...
⬇️ Downloading linux dataset...
✅ Data ready.
```

```
import os, json, numpy as np
from tqdm import tqdm

base_data = "data"

for name in ["warpeace", "linux"]:
    ddir = os.path.join(base_data, name)
    vocab_json = os.path.join(ddir, "vocab.json")
    raw_txt = os.path.join(ddir, f"{name}.txt")
    tokens_file = os.path.join(ddir, "tokens.ids")

    if not os.path.exists(vocab_json):
        raise FileNotFoundError(f"Missing {vocab_json}")
    if not os.path.exists(raw_txt):
        raise FileNotFoundError(f"Missing {raw_txt}")

    info = json.load(open(vocab_json, "r", encoding="utf-8"))
    stoi = info["stoi"]

    print(f"🔧 Generating tokens.ids for {name} ...")
    with open(raw_txt, "r", encoding="utf-8") as f:
        text = f.read()

    # Simple tokenization (char-level)
    tokens = [stoi[ch] for ch in tqdm(text, desc=f"Tokenizing {name}") if ch i

    np.savetxt(tokens_file, tokens, fmt="%d")
    print(f"✅ Saved {len(tokens)} tokens → {tokens_file}")
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
/tmp/ipython-input-2151976901.py in <cell line: 0>()
     13         raise FileNotFoundError(f"Missing {vocab_json}")
     14     if not os.path.exists(raw_txt):
---> 15         raise FileNotFoundError(f"Missing {raw_txt}")
     16 
     17     info = json.load(open(vocab_json, "r", encoding="utf-8"))

FileNotFoundError: Missing data/warpeace/warpeace.txt
```

```
!python train_mlp.py --dataset warpeace --epochs 30 --batch_size 64 --lr 1e-3 -
!python train_mlp.py --dataset linux --epochs 30 --batch_size 64 --lr 1e-3 --sa
```

```
Device: cpu
Traceback (most recent call last):
  File "/content/MLPColab/MLPColab/train_mlp.py", line 131, in <module>
    main(args)
  File "/content/MLPColab/MLPColab/train_mlp.py", line 52, in main
    train_ds = NextWordDataset(args.dataset, context_len=args.context_len
               ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/content/MLPColab/MLPColab/dataset.py", line 20, in __init__
    with open(tokens_path, "r", encoding="utf-8") as f:
         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
FileNotFoundError: [Errno 2] No such file or directory: 'data/warpeace/to
Device: cpu
Traceback (most recent call last):
  File "/content/MLPColab/MLPColab/train_mlp.py", line 131, in <module>
    main(args)
  File "/content/MLPColab/MLPColab/train_mlp.py", line 52, in main
    train_ds = NextWordDataset(args.dataset, context_len=args.context_len
               ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/content/MLPColab/MLPColab/dataset.py", line 20, in __init__
    with open(tokens_path, "r", encoding="utf-8") as f:
         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
FileNotFoundError: [Errno 2] No such file or directory: 'data/linux/token
```

```
from pyngrok import ngrok
import subprocess, time

# Kill old sessions
!fuser -k 8501/tcp || echo "No old Streamlit session"

# Start Streamlit (must exist in same dir)
```

```
process = subprocess.Popen(["streamlit", "run", "app.py", "--server.port", "85
time.sleep(6)

# Connect to ngrok tunnel
public_url = ngrok.connect(8501)
print("✅ Streamlit running at:", public_url)
```

```
# LINUX training (30 epochs) - smaller batch to avoid OOM
python train_mlp.py \
  --dataset linux \
  --context_len 5 \
  --embed_dim 64 \
  --hidden_size 512 \
  --n_hidden 1 \
  --activation relu \
  --dropout 0.2 \
  --batch_size 64 \
  --epochs 30 \
  --lr 1e-3 \
  --save_dir checkpoints/linux \
  --save_every 5 \
  --val_ratio 0.1
```

```
echo "War & Peace checkpoints:"
ls -la checkpoints/warpeace || true
echo "Linux checkpoints:"
ls -la checkpoints/linux || true
```

Warpeace

```
# determine model file (best.pt preferred)
if [ -f checkpoints/warpeace/best.pt ]; then MODEL_W=checkpoints/warpeace/best
echo "Using War model: $MODEL_W"
python visualize_embeddings.py --dataset warpeace --model_path "$MODEL_W" --to
```

Linux

```
if [ -f checkpoints/linux/best.pt ]; then MODEL_L=checkpoints/linux/best.pt; e
echo "Using Linux model: $MODEL_L"
python visualize_embeddings.py --dataset linux --model_path "$MODEL_L" --top_k

from IPython.display import Image, display
print("War & Peace embedding viz:")
display(Image('checkpoints/warpeace/warpeace_tsne.png'))
print("Linux embedding viz:")
display(Image('checkpoints/linux/linux_tsne.png'))
```

Optional (below)

```
%%bash
python - <<'PY'
import torch, json, sys
from pathlib import Path

def load_ckpt(p):
    ck = torch.load(p, map_location='cpu')
    return ck

def simple_generate(ckpt_path, seed_text="the", k=30, context_len=5, temperatu
    ck = load_ckpt(ckpt_path)
    vocab = ck['vocab']
    args = ck.get('args', {})
    # Build model skeleton exactly like train_mlp MLPNextWord
    import torch.nn as nn
    class MLPNextWord(nn.Module):
        def __init__(self, vocab_size, context_len=5, embed_dim=64, hidden_size
            super().__init__()
            self.embedding = nn.Embedding(vocab_size, embed_dim, padding_idx=0
            layers = []
            in_dim = context_len * embed_dim
            for _ in range(n_hidden):
                layers.append(nn.Linear(in_dim, hidden_size))
                layers.append(nn.ReLU() if activation=="relu" else nn.Tanh())
                in_dim = hidden_size
            layers.append(nn.Linear(in_dim, vocab_size))
            self.net = nn.Sequential(*layers)
        def forward(self, x):
            e = self.embedding(x)
            flat = e.view(e.size(0), -1)
            return self.net(flat)
    margs = args
    model = MLPNextWord(vocab_size=len(vocab), context_len=margs.get('context_
    model.load_state_dict(ck['model_state'], strict=False)
    model.eval()
    word2idx = {w:i for i,w in enumerate(vocab)}
    idx2word = {i:w for i,w in enumerate(vocab)}
    tokens = seed_text.lower().split()
    ids = [word2idx.get(t, word2idx.get("<UNK>",0)) for t in tokens]
    out = tokens[:]
    import numpy as np
    for _ in range(k):
        context = ids[-context_len:] if len(ids)>=context_len else [word2idx.ge
        x = torch.tensor([context], dtype=torch.long)
        logits = model(x)[0].detach().numpy()
        # temperature sampling
        if temperature<=0:
            nid = int(logits.argmax())
        else:
            probs = np.exp(logits/temperature)
            probs = probs / probs.sum()
            nid = int(np.random.choice(len(probs), p=probs))
        out.append(idx2word.get(nid, "<UNK>"))
        ids.append(nid)
    return " ".join(out)
```

```
# adjust these paths as appropriate
war_ckpt = Path("checkpoints/warpeace/best.pt")
if not war_ckpt.exists():
    war_ckpt = Path("checkpoints/warpeace/final.pt")
linux_ckpt = Path("checkpoints/linux/best.pt")
if not linux_ckpt.exists():
    linux_ckpt = Path("checkpoints/linux/final.pt")

print("War sample:")
print(simple_generate(str(war_ckpt), seed_text="the prince", k=30, context_len
print("\nLinux sample:")
print(simple_generate(str(linux_ckpt), seed_text="static inline", k=30, contex
PY
```

.app py created

```
%%writefile app.py
import streamlit as st
import torch, os
st.set_page_config(page_title="MLP Next-Word Generator", layout="wide")

st.title("MLP Next-Word Generator — War & Peace & Linux")
st.markdown("Choose a dataset and a saved checkpoint (best.pt or final.pt), the

DATASETS = {
    "War and Peace": "checkpoints/warpeace",
    "Linux kernel": "checkpoints/linux"
}

dataset = st.selectbox("Dataset", list(DATASETS.keys()))
model_dir = st.text_input("Model dir (auto)", value=DATASETS[dataset])
model_file = st.text_input("Model file (default best.pt)", value=os.path.join(
if not os.path.exists(model_file):
    st.warning(f"Model file {model_file} not found. Please set correct path (e

seed_text = st.text_input("Seed text:", "the")
context_len = st.number_input("Context length", min_value=1, max_value=20, valu
k = st.number_input("Number of words to generate", min_value=1, max_value=500,
temperature = st.slider("Temperature", min_value=0.0, max_value=2.0, value=1.0

@st.cache_resource
def load_ckpt(path):
    ckpt = torch.load(path, map_location='cpu')
    return ckpt

def build_model_from_ckpt(ckpt):
    import torch.nn as nn
    args = ckpt.get('args', {})
    vocab = ckpt['vocab']
    class MLPNextWord(nn.Module):
        def __init__(self, vocab_size, context_len=5, embed_dim=64, hidden_size
            super().__init__()
            self.embedding = nn.Embedding(vocab_size, embed_dim, padding_idx=0
            layers = []
            in_dim = context_len * embed_dim
            for _ in range(n_hidden):
                layers.append(nn.Linear(in_dim, hidden_size))
                layers.append(nn.ReLU() if activation=="relu" else nn.Tanh())
                in_dim = hidden_size
            layers.append(nn.Linear(in_dim, vocab_size))
            self.net = nn.Sequential(*layers)
        def forward(self, x):
            e = self.embedding(x)
            flat = e.view(e.size(0), -1)
            return self.net(flat)
    m = MLPNextWord(vocab_size=len(vocab), context_len=args.get('context_len',
    m.load_state_dict(ckpt['model_state'], strict=False)
    m.eval()
    return m, vocab

if st.button("Load model & generate"):
    if not os.path.exists(model_file):
        st.error("Model file not found. Please correct model path.")
    else:
        ckpt = load_ckpt(model_file)
        model, vocab = build_model_from_ckpt(ckpt)
        w2i = {w:i for i,w in enumerate(vocab)}
        i2w = {i:w for i,w in enumerate(vocab)}
        tokens = seed_text.strip().lower().split()
        ids = [w2i.get(t, w2i.get("<UNK>",0)) for t in tokens]
        out = tokens[:]
        import numpy as np
        for _ in range(k):
            context = ids[-context_len:] if len(ids)>=context_len else [w2i.ge
            x = torch.tensor([context], dtype=torch.long)
            logits = model(x)[0].detach().numpy()
            if temperature<=0:
                nid = int(logits.argmax())
            else:
                probs = np.exp(logits/temperature)
                probs = probs / probs.sum()
                nid = int(np.random.choice(len(probs), p=probs))
            out.append(i2w.get(nid, "<UNK>"))
            ids.append(nid)
        st.subheader("Generated text")
        st.write(" ".join(out))
```

```
!./ngrok authtoken YOUR_NGROK_AUTHTOKEN


!kill $(lsof -t -i:8501) >/dev/null 2>&1 || echo "no process"

#  run streamlit in background
import subprocess, time, os
proc = subprocess.Popen(["streamlit", "run", "app.py", "--server.port", "8501"]
time.sleep(6)

#  create tunnel and print URL
from pyngrok import ngrok
public_url = ngrok.connect(8501)
```

```
    print("Streamlit public URL:", public_url)
```