```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import os
print("Current directory:", os.getcwd())
print("Drive contents:", os.listdir('/content/drive/MyDrive/'))
```

```
Current directory: /content
Drive contents: ['Colab Notebooks', '.ipynb_checkpoints', 'Dataset', 'hdvjb d', 'outputData']
```

```
# dataset_dir = '/content/drive/MyDrive/openwebtext_dataset'
# os.makedirs(dataset_dir, exist_ok=True)
# print(f"Created directory: {dataset_dir}")
```

```
Created directory: /content/drive/MyDrive/openwebtext_dataset
```

```
!pip install -q datasets huggingface_hub tqdm zstandard
```

```
# from datasets import load_dataset
# import gzip, json
# from tqdm.auto import tqdm

# # Load the OpenWebText mirror (compatible with latest datasets)
# ds = load_dataset("vietgpt/openwebtext_en", split="train", streaming=True)

# out_path = "/content/drive/MyDrive/Dataset/openwebtext.jsonl.gz"   # change if needed
# cnt = 0

# with gzip.open(out_path, "wt", encoding="utf-8") as fout:
#     for doc in tqdm(ds, desc="Writing docs"):
#         fout.write(json.dumps(doc, ensure_ascii=False) + "\n")
#         cnt += 1
#         # optional: stop early for testing
#         # if cnt >= 100000: break

# print(f"✅ Done — written {cnt} documents to {out_path}")
```

```
Resolving data files: 100%                                    20/20 [00:00<00:00, 1416.99it/s]
Writing docs:       8013769/? [1:53:05<00:00, 1399.77it/s]
✅ Done — written 8013769 documents to /content/drive/MyDrive/Dataset/openwebtext.jsonl.gz
```

```
import gzip, json

cnt = 0
with gzip.open("/content/drive/MyDrive/Dataset/openwebtext.jsonl.gz", "rt", encoding="utf-8") as fin:
    for _ in fin:
        cnt += 1
print("Total documents:", cnt)
```

```
Show hidden output
```

```
!pip install --upgrade pip
!pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu121
```

```
!pip install transformers accelerate bitsandbytes
```

Show hidden output

```
!nvidia-smi
```

Show hidden output

```
!pip install -q torch torchvision torchaudio --extra-index-url https://download.pytorch.org/whl/cu121
!pip install -q transformers datasets accelerate bitsandbytes sentencepiece
```

```
import torch
print("CUDA available:", torch.cuda.is_available())
print("GPU name:", torch.cuda.get_device_name(0))
print("Memory allocated:", round(torch.cuda.memory_allocated(0)/1024**3, 2), "GB")
```

```
CUDA available: True
GPU name: Tesla T4
Memory allocated: 0.0 GB
```

```
!pip install -q transformers accelerate bitsandbytes sentencepiece
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61.3/61.3 MB 13.9 MB/s eta 0:00:00
```

```
from transformers import AutoModelForCausalLM, AutoTokenizer

MODEL_NAME = "TinyLlama/TinyLlama-1.1B-intermediate-step-1431k-3T"

tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)

model = AutoModelForCausalLM.from_pretrained(
    MODEL_NAME,
    device_map="auto",          # puts layers on GPU
    load_in_4bit=True,          # quantize to 4-bit
    torch_dtype="auto",
)

print("Loaded:", MODEL_NAME)
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
tokenizer_config.json: 100%  776/776 [00:00<00:00, 30.1kB/s]
tokenizer.model: 100%  500k/500k [00:01<00:00, 357kB/s]
tokenizer.json:  1.84M/? [00:00<00:00, 39.2MB/s]
special_tokens_map.json: 100%  414/414 [00:00<00:00, 10.5kB/s]
config.json: 100%  560/560 [00:00<00:00, 55.1kB/s]
The `load_in_4bit` and `load_in_8bit` arguments are deprecated and will be removed in the future versio
model.safetensors: 100%  4.40G/4.40G [01:25<00:00, 127MB/s]
generation_config.json: 100%  129/129 [00:00<00:00, 14.4kB/s]
Loaded: TinyLlama/TinyLlama-1.1B-intermediate-step-1431k-3T
```

```
inputs = tokenizer("Hello, how are you?", return_tensors="pt").to(model.device)
outputs = model.generate(**inputs, max_new_tokens=50)
print(tokenizer.decode(outputs[0], skip_special_tokens=True))
```

```
Hello, how are you?


    A: You can use the following code:
    import sys
    import time

    def print_hello(name):
        print("Hello, " + name)

    def print_bye(name):
        print("
```

```
# # ================== DRIVE MOUNT ==================
# from google.colab import drive
# drive.mount('/content/drive', force_remount=True)

# # ================== CONFIG ==================
# OUT_DIR  = "/content/drive/MyDrive/outputDatadir"
# INPUT_PATH = "/content/drive/MyDrive/Dataset/openwebtext.jsonl.gz"
# MODEL_NAME = "microsoft/phi-1_5"

# N_LINES = 15          # always take 15 lines
# GEN_TOKENS = 256
# MAX_LEN = 2048        # phi-1_5 context window
# # ============================================

# import os, re, gzip, json
# import torch
# from transformers import AutoTokenizer, AutoModelForCausalLM

# os.makedirs(OUT_DIR, exist_ok=True)

# # ---------- Load model ----------
# print("Loading model:", MODEL_NAME)
# tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
# model = AutoModelForCausalLM.from_pretrained(
#     MODEL_NAME,
#     device_map="auto",
```

```
#     load_in_4bit=True,
#     torch_dtype="auto"
# )
# model.eval()
# device = next(model.parameters()).device
# MODEL_MAX_LEN = tokenizer.model_max_length or MAX_LEN
# print(f"Loaded {MODEL_NAME} on {device} (max_len={MODEL_MAX_LEN})")

# # ---------- Input helper ----------
# def open_input(path):
#     if path.endswith(".gz"):
#         return gzip.open(path, "rt", encoding="utf-8", errors="ignore")
#     return open(path, "r", encoding="utf-8", errors="ignore")

# def extract_text(line):
#     try:
#         obj = json.loads(line)
#         if isinstance(obj, dict) and "text" in obj:
#             return obj["text"]
#         if isinstance(obj, str):
#             return obj
#         if isinstance(obj, dict):
#             for v in obj.values():
#                 if isinstance(v, str) and len(v) > 20:
#                     return v
#     except Exception:
#         return None
#     return None

# # ---------- Build prompt ----------
# def build_prompt(lines):
#     instr = (
#         "You are a text rewriter. For each input line, output in this exact format:\n"
#         "Q: <original line>\n"
#         "A: <rewritten line>\n\n"
#         f"Do this for exactly {len(lines)} lines. No extra commentary.\n\n"
#         "=== INPUT LINES ===\n"
#     )
#     return instr + "\n".join(lines) + "\n\n=== OUTPUT ===\n"

# # ---------- Run LLM ----------
# def run_llm(prompt):
#     try:
#         inputs = tokenizer(prompt, return_tensors="pt", truncation=True, max_length=MODEL_MAX_LEN).to(dev
#         with torch.no_grad():
#             outputs = model.generate(
#                 **inputs,
#                 max_new_tokens=GEN_TOKENS,
#                 do_sample=True,
#                 temperature=0.7,
#                 top_p=0.9,
#                 eos_token_id=tokenizer.eos_token_id,
#                 pad_token_id=tokenizer.pad_token_id or tokenizer.eos_token_id,
#             )
#         out_ids = outputs[0][inputs["input_ids"].shape[1]:]
#         text = tokenizer.decode(out_ids, skip_special_tokens=True).strip()
#         return text
#     except Exception as e:
#         return f"ERROR: {e}"

# # ---------- Main test ----------
# lines = []
# with open_input(INPUT_PATH) as fin:
#     for _ in range(N_LINES * 5):   # read extra to skip empties
#         raw = fin.readline()
#         if raw == "":
```

```
#             break
#         txt = extract_text(raw)
#         if txt:
#             lines.append(txt.strip())
#         if len(lines) >= N_LINES:
#             break

# if not lines:
#     raise ValueError("No valid lines found in dataset!")

# print(f"Collected {len(lines)} lines, sending to model...")

# prompt = build_prompt(lines)
# output = run_llm(prompt)

# out_file = os.path.join(OUT_DIR, "qa_batch_test.txt")
# with open(out_file, "w", encoding="utf-8") as f:
#     f.write(output)

# print("\nSaved Q/A batch to:", out_file)
# print("\n--- Preview ---\n")
# print(output[:800])
```

```
Mounted at /content/drive
Loading model: microsoft/phi-1_5
The `load_in_4bit` and `load_in_8bit` arguments are deprecated and will be removed in the future versio
Loaded microsoft/phi-1_5 on cuda:0 (max_len=2048)
Collected 15 lines, sending to model...

Saved Q/A batch to: /content/drive/MyDrive/outputDatadir/qa_batch_test.txt

--- Preview ---

her more immediate goals as president.

“First, we have to get the economy back on track. We have to create more jobs and restore confidence in

She added that she has a detailed plan to do that, and that she will work with her team to get it done.

“Second, we have to make sure that every American has access to quality healthcare. We have to make sur

She added that she has a plan to do that, and that she will work with her team to get it done.

“Third, we have to make sure that we are working together to solve the climate crisis. We have to make
```

```
# ================== DRIVE MOUNT ==================
from google.colab import drive
drive.mount('/content/drive', force_remount=True)

# ================== CONFIG ==================
OUT_DIR  = "/content/drive/MyDrive/data"
INPUT_PATH = "/content/drive/MyDrive/Dataset/openwebtext.jsonl.gz"
MODEL_NAME = "microsoft/phi-1_5"

N_LINES = 15          # always 15 lines per batch
GEN_TOKENS = 256
MAX_LEN = 2048
# ============================================

import os, re, gzip, json, time
from pathlib import Path
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM

os.makedirs(OUT_DIR, exist_ok=True)
```

```
# ---------- Load model ----------
print("Loading model:", MODEL_NAME)
tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
model = AutoModelForCausalLM.from_pretrained(
    MODEL_NAME,
    device_map="auto",
    load_in_4bit=True,
    torch_dtype="auto"
)
model.eval()
device = next(model.parameters()).device
MODEL_MAX_LEN = tokenizer.model_max_length or MAX_LEN
print(f"Loaded {MODEL_NAME} on {device} (max_len={MODEL_MAX_LEN})")

# ---------- Helpers ----------
def open_input(path):
    if path.endswith(".gz"):
        return gzip.open(path, "rt", encoding="utf-8", errors="ignore")
    return open(path, "r", encoding="utf-8", errors="ignore")

def extract_text(line):
    try:
        obj = json.loads(line)
        if isinstance(obj, dict) and "text" in obj:
            return obj["text"]
        if isinstance(obj, str):
            return obj
        if isinstance(obj, dict):
            for v in obj.values():
                if isinstance(v, str) and len(v) > 20:
                    return v
    except Exception:
        return None
    return None

def build_prompt(lines):
    instr = (
        "You are a text rewriter. For each input line, output in this exact format:\n"
        "Q: <original line>\n"
        "A: <rewritten line>\n\n"
        f"Do this for exactly {len(lines)} lines. No extra commentary.\n\n"
        "=== INPUT LINES ===\n"
    )
    return instr + "\n".join(lines) + "\n\n=== OUTPUT ===\n"

def run_llm(prompt):
    try:
        inputs = tokenizer(prompt, return_tensors="pt", truncation=True, max_length=MODEL_MAX_LEN).to(devic
        with torch.no_grad():
            outputs = model.generate(
                **inputs,
                max_new_tokens=GEN_TOKENS,
                do_sample=True,
                temperature=0.7,
                top_p=0.9,
                eos_token_id=tokenizer.eos_token_id,
                pad_token_id=tokenizer.pad_token_id or tokenizer.eos_token_id,
            )
        out_ids = outputs[0][inputs["input_ids"].shape[1]:]
        text = tokenizer.decode(out_ids, skip_special_tokens=True).strip()
        return text
    except Exception as e:
        return f"ERROR: {e}"

# ---------- Main loop ----------
```

```
    def main():
        batch_idx = 1
        buf = []

        with open_input(INPUT_PATH) as fin:
            for line in fin:
                txt = extract_text(line)
                if not txt:
                    continue
                txt = txt.strip()
                if not txt:
                    continue

                buf.append(txt)
                if len(buf) >= N_LINES:
                    # build prompt + run model
                    prompt = build_prompt(buf)
                    print(f"\n>>> Processing batch {batch_idx} (lines={len(buf)})...")
                    output = run_llm(prompt)

                    # save output
                    out_file = os.path.join(OUT_DIR, f"qa_batch_{batch_idx:05d}.txt")
                    with open(out_file, "w", encoding="utf-8") as f:
                        f.write(output)
                    print("✓ Saved:", out_file)

                    # reset for next batch
                    buf = []
                    batch_idx += 1
                    time.sleep(0.5)

            # process last partial batch
            if buf:
                prompt = build_prompt(buf)
                print(f"\n>>> Processing final batch {batch_idx} (lines={len(buf)})...")
                output = run_llm(prompt)
                out_file = os.path.join(OUT_DIR, f"qa_batch_{batch_idx:05d}.txt")
                with open(out_file, "w", encoding="utf-8") as f:
                    f.write(output)
                print("✓ Saved:", out_file)

        print("\nAll done. Total batches:", batch_idx)

    # --------- Run ---------
    main()
```

Show hidden output

```
    import os

    # Path to your output directory
    OUT_DIR = "/content/drive/MyDrive/data"

    total_chars = 0
    file_count = 0

    for fname in os.listdir(OUT_DIR):
        fpath = os.path.join(OUT_DIR, fname)
        if os.path.isfile(fpath) and fname.endswith(".txt"):
            with open(fpath, "r", encoding="utf-8", errors="ignore") as f:
                total_chars += len(f.read())
            file_count += 1

    print(f"Scanned {file_count} files.")
```

```
print(f"Total characters across all files: {total_chars:,}")
```

```
Scanned 945 files.
Total characters across all files: 1,008,820
```

```
# # because of less compute the outfiles are 914 which is extremely small for training a tokenizer
# # ================== MOUNT DRIVE ==================

# from google.colab import drive
# drive.mount('/content/drive')

# # ================== INSTALL & IMPORT ==================
# !pip install datasets -q

# from datasets import load_dataset
# import json, os

# # ================== CONFIG ==================
# OUT_PATH = "/content/drive/MyDrive/data/arxiv_cs_subset.jsonl"  # where to save
# DATASET = "CShorten/ML-ArXiv-Papers"  # or "ashish-chouhan/arxiv_cs_papers"

# # ================== DOWNLOAD DATASET ==================
# print(f"Downloading {DATASET}...")
# ds = load_dataset(DATASET)

# print("Dataset loaded. Example row:")
# print(ds["train"][0])

# # ================== SAVE TO DRIVE ==================
# with open(OUT_PATH, "w", encoding="utf-8") as fout:
#     for row in ds["train"]:
#         fout.write(json.dumps(row, ensure_ascii=False) + "\n")

# print(f"\n✓ Saved CS subset to: {OUT_PATH}")
# print(f"Size on disk: {os.path.getsize(OUT_PATH)/1024/1024:.2f} MB")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/dri
Downloading CShorten/ML-ArXiv-Papers...
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
README.md: 100%  986/986 [00:00<00:00, 73.2kB/s]
ML-Arxiv-Papers.csv: 100%  147M/147M [00:01<00:00, 110MB/s]
Generating train split: 100%  117592/117592 [00:01<00:00, 59214.67 examples/s]
Dataset loaded. Example row:
{'Unnamed: 0.1': 0, 'Unnamed: 0': 0.0, 'title': 'Learning from compressed observations', 'abstract': '

✓ Saved CS subset to: /content/drive/MyDrive/data/arxiv_cs_subset.jsonl
Size on disk: 148.43 MB
```

```
# # ================== MOUNT DRIVE ==================
# from google.colab import drive
# drive.mount('/content/drive')

# # ================== INSTALL KAGGLE API ==================
# !pip install kaggle -q
```

```
# import os

# # ================== CONFIG ==================
# OUT_DIR = "/content/drive/MyDrive/arxiv_dataset"
# os.makedirs(OUT_DIR, exist_ok=True)

# # Make sure you have your Kaggle API key saved as kaggle.json in your Drive
# KAGGLE_JSON = "/content/drive/MyDrive/kaggle.json"

# # ================== SETUP KAGGLE CREDENTIALS ==================
# !mkdir -p ~/.kaggle
# !cp {KAGGLE_JSON} ~/.kaggle/
# !chmod 600 ~/.kaggle/kaggle.json

# # ================== DOWNLOAD FULL ARXIV SNAPSHOT ==================
# print("Downloading full arXiv dataset (~2.5 GB)...")
# !kaggle datasets download -d Cornell-University/arxiv -p {OUT_DIR}

# # ================== UNZIP ==================
# print("Unzipping dataset...")
# !unzip -q {OUT_DIR}/arxiv.zip -d {OUT_DIR}

# print("\n✓ Done! Dataset saved to:", OUT_DIR)
# !ls -lh {OUT_DIR}
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/dri
Downloading full arXiv dataset (~2.5 GB)...
Dataset URL: https://www.kaggle.com/datasets/Cornell-University/arxiv
License(s): CC0-1.0
Downloading arxiv.zip to /content/drive/MyDrive/arxiv_dataset
 99% 1.48G/1.49G [00:12<00:00, 130MB/s]
100% 1.49G/1.49G [00:12<00:00, 126MB/s]
Unzipping dataset...

✓ Done! Dataset saved to: /content/drive/MyDrive/arxiv_dataset
total 6.0G
-rw------- 1 root root 4.6G Aug 30 23:53 arxiv-metadata-oai-snapshot.json
```