🚀 45-Minute Hands-On: LLMs with Hugging Face (Colab/Jupyter)

Last updated: 2025-09-01 05:29

Goals

- Run a small **instruction-tuned LLM** with 🤗 Transformers
- Use the **pipeline** API
- Tune decoding (temperature, top-p, top-k)
- Build a tiny **chat loop**
- Batch prompts → CSV

```
# 1) Install dependencies
!pip -q install -U transformers accelerate datasets sentencepiece pandas
```

```
# 2) Imports & device
import torch, time
from transformers import AutoTokenizer, AutoModelForCausalLM, pipeline
device = "cuda" if torch.cuda.is_available() else "cpu"
print("Device:", device)
```

```
Device: cuda
```

Model choice

We try **TinyLlama/TinyLlama-1.1B-Chat-v1.0** and fall back to **distilgpt2** if needed.

```
# 3) Load model
model_id = "TinyLlama/TinyLlama-1.1B-Chat-v1.0"
fallback_model_id = "distilgpt2"

def load_model(model_name):
    try:
        tok = AutoTokenizer.from_pretrained(model_name, use_fast=True)
        mdl = AutoModelForCausalLM.from_pretrained(
            model_name,
            torch_dtype=torch.float16 if device == "cuda" else torch.float32,
            device_map="auto" if device == "cuda" else None
        )
        return tok, mdl, model_name
    except Exception as e:
        print("Primary failed:", e, "\nFalling back to", fallback_model_id)
        tok = AutoTokenizer.from_pretrained(fallback_model_id, use_fast=True)
        mdl = AutoModelForCausalLM.from_pretrained(
            fallback_model_id,
            torch_dtype=torch.float16 if device == "cuda" else torch.float32,
            device_map="auto" if device == "cuda" else None
        )
        return tok, mdl, fallback_model_id

tokenizer, model, active_model_id = load_model(model_id)
print("Loaded:", active_model_id)
```

```
tokenizer_config.json: 1.29k/? [00:00<00:00, 131kB/s]
tokenizer.model: 100% 500k/500k [00:00<00:00, 1.07MB/s]
tokenizer.json: 1.84M/? [00:00<00:00, 65.2MB/s]
special_tokens_map.json: 100% 551/551 [00:00<00:00, 67.9kB/s]
config.json: 100% 608/608 [00:00<00:00, 41.5kB/s]
model.safetensors: 100% 2.20G/2.20G [00:36<00:00, 133MB/s]
generation_config.json: 100% 124/124 [00:00<00:00, 8.87kB/s]
Loaded: TinyLlama/TinyLlama-1.1B-Chat-v1.0
```

Quickstart with `pipeline`

```
# 4) Text generation quickstart
gen = pipeline("text-generation", model=model, tokenizer=tokenizer)
prompt = "Explain what a Knowledge Graph is in healthcare, in 3 concise sentences."
out = gen(prompt, max_new_tokens=120, do_sample=True, temperature=0.7, top_p=0.9, pad_token_id=tokenizer.eos_token_id)[0][
print(out)
```

```
Device set to use cuda:0
Explain what a Knowledge Graph is in healthcare, in 3 concise sentences. A Knowledge Graph is a comprehensive represen
```

Tokenization peek

```
# 5) Tokenization
text = "Large Language Models can draft emails and summarize clinical notes."
ids = tokenizer(text).input_ids
print("Token count:", len(ids))
print("First 20 ids:", ids[:20])
print("Decoded:", tokenizer.decode(ids))
```

```
Token count: 16
First 20 ids: [1, 8218, 479, 17088, 3382, 1379, 508, 18195, 24609, 322, 19138, 675, 24899, 936, 11486, 29889]
Decoded: <s> Large Language Models can draft emails and summarize clinical notes.
```

Decoding controls (temperature/top-p/top-k)

```
# 6) Compare decoding
base_prompt = "Give 3 short tips for writing reproducible data science code:"
settings = [
    {"temperature": 0.2, "top_p": 0.95, "top_k": 50},
    {"temperature": 0.8, "top_p": 0.9, "top_k": 50},
    {"temperature": 1.1, "top_p": 0.85, "top_k": 50},
]
for i, s in enumerate(settings, 1):
    t0 = time.time()
    out = gen(base_prompt, max_new_tokens=100, do_sample=True, temperature=s["temperature"], top_p=s["top_p"], top_k=s["to
    print(f"\n--- Variant {i} | temp={s['temperature']} top_p={s['top_p']} top_k={s['top_k']} ---")
    print(out)
    print(f"(latency ~{time.time()-t0:.2f}s)")
```

```
--- Variant 1 | temp=0.2 top_p=0.95 top_k=50 ---
Give 3 short tips for writing reproducible data science code: 1. Use functions to encapsulate your code and make it ea
(latency ~2.48s)

--- Variant 2 | temp=0.8 top_p=0.9 top_k=50 ---
Give 3 short tips for writing reproducible data science code: 1. Always start by defining your problem and your hypoth
(latency ~2.79s)

--- Variant 3 | temp=1.1 top_p=0.85 top_k=50 ---
Give 3 short tips for writing reproducible data science code: a. Use descriptive variable names. b. Keep code clean an
(latency ~1.44s)
```

Minimal chat loop

```
# 7) Simple chat helper
def build_prompt(history, user_msg, system="You are a helpful data science assistant."):
    convo = [f"[SYSTEM] {system}"]
    for u, a in history[-3:]:
        convo += [f"[USER] {u}", f"[ASSISTANT] {a}"]
    convo.append(f"[USER] {user_msg}\n[ASSISTANT]")
    return "\n".join(convo)

history = []

def chat_once(user_msg, max_new_tokens=128, temperature=0.7, top_p=0.9):
    prompt = build_prompt(history, user_msg)
    inputs = tokenizer(prompt, return_tensors="pt").to(model.device)
    with torch.no_grad():
        tokens = model.generate(**inputs, max_new_tokens=max_new_tokens, do_sample=True, temperature=temperature, top_p=to
    text = tokenizer.decode(tokens[0], skip_special_tokens=True)
    reply = text.split("[ASSISTANT]")[-1].strip()
    history.append((user_msg, reply))
    print(reply)

chat_once("In one sentence, what is transfer learning?")
chat_once("Name two risks when fine-tuning small LLMs on tiny datasets.")
chat_once("Suggest one mitigation for each risk.")
```

```
In that case, transfer learning can still be useful. It is a powerful technique that can help you avoid retraining a l
Sure, here are two risks that can arise when fine-tuning small LLMs on tiny datasets:
1. Limited data: Small datasets can limit the amount of data that you can use to train your LLM. This can lead to unde
To address the first risk, here are some mitigations:
1. Use a larger dataset: To address the second risk, here are some mitigations:
1. Use a larger dataset: Large datasets can provide a more diverse training set and help to mitigate the risk of overf
```

Batch prompts → CSV

```
# 8) Batch prompts and save
import pandas as pd, time
prompts = [
    "Write a tweet (<=200 chars) about reproducible ML.",
    "One sentence: why eval metrics matter beyond accuracy.",
    "List 3 checks before deploying a model to production.",
    "Explain temperature vs. top-p to a PM."
]
rows = []
for p in prompts:
    t0 = time.time()
    out = gen(p, max_new_tokens=100, do_sample=True, temperature=0.7, top_p=0.9, pad_token_id=tokenizer.eos_token_id)[0]["ge
    rows.append({"prompt": p, "output": out, "latency_s": round(time.time()-t0, 2)})
df = pd.DataFrame(rows)
df
```

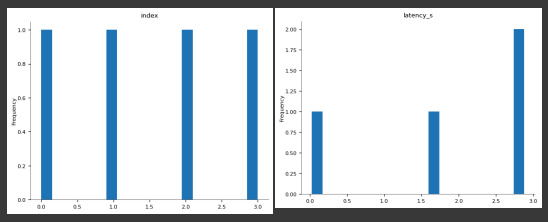
1 to 4 of 4 entries Filter

index	prompt	output	latency_s
0	Write a tweet (<=200 chars) about reproducible ML.	Write a tweet (<=200 chars) about reproducible ML. Use a clear and concise writing style, and include relevant links to resources or articles. Make sure to include a call-to-action encouraging readers to learn more about reproducible ML and how to implement it in their own work.	1.68
1	One sentence: why eval metrics matter beyond accuracy.	One sentence: why eval metrics matter beyond accuracy. Title: Evaluating Deep Learning Models: A Comprehensive Overview Abstract: Deep Learning models are powerful tools for solving complex problems, but they are often evaluated using metrics such as accuracy, precision, recall, and F1 score. These metrics are important, but they don't always provide a complete picture of a model's performance. In this article, we'll explore some alternative evaluation metrics that can help you understand how a model performs beyond	2.88
2	List 3 checks before deploying a model to production.	List 3 checks before deploying a model to production. 1. Validate data quality: Ensure that the data used to train the model is accurate, complete, and clean. 2. Check for potential biases: Verify that the model does not have any unintended biases that could affect the model's predictions. 3. Validate model accuracy: Confirm that the model performs well on a validation dataset to ensure that it is accurate and reliable. 4. Check for overfitting: Ens	2.88
3	Explain temperature vs. top-p to a PM.	Explain temperature vs. top-p to a PM.	0.03

Show 10 per page

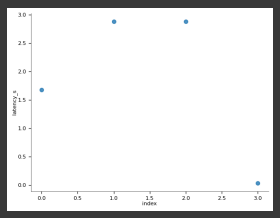
Like what you see? Visit the data table notebook to learn more about interactive tables.

Distributions

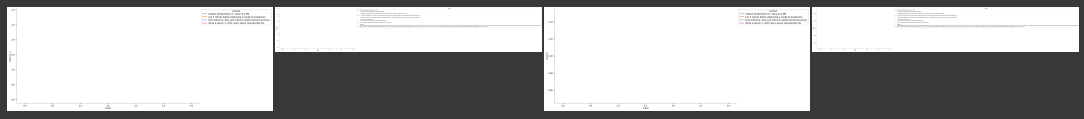


Categorical distributions

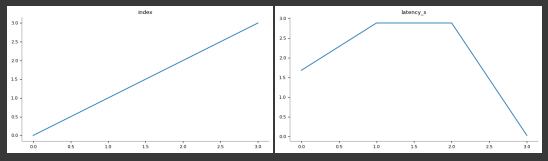
2-d distributions



Time series



Values

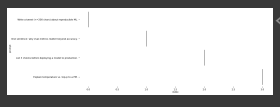


2-d categorical distributions

Faceted distributions

```
<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hu
```

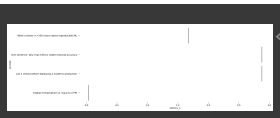


```
<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hu
```

```
<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hu
```



```
<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hu
```

Next steps: Generate code with df | View recommended plots | New interactive sheet

Double-click (or enter) to edit

```
# 8b) Save to CSV (download from left sidebar in Colab)
out_path = "/content/hf_llm_batch_outputs.csv"
df.to_csv(out_path, index=False)
print("Saved to:", out_path)
```

```
Saved to: /content/hf_llm_batch_outputs.csv
```

Double-click (or enter) to edit

Double-click (or enter) to edit

Which settings felt most factual vs. most creative

The “factual vs. creative” balance is mostly controlled by sampling parameters when you call model.generate.

More Factual / Deterministic

temperature=0.0 or very low (≤0.3) → Always picks the most likely token → more “factual” but repetitive.

do_sample=False → Turns off randomness, makes outputs deterministic for the same input.

top_p=1.0 and top_k=0 (default) → No restriction beyond model probabilities.

Effect: The model behaves more like a “fact retriever” → good for definitions, explanations, and structured answers.

More Creative / Varied

temperature=0.8–1.2 → Adds randomness, allows more surprising word choices.

do_sample=True → Enables stochastic sampling instead of always picking the top token.

top_p=0.8–0.9 (nucleus sampling) → Limits choices to a smaller set of high-probability tokens → encourages diversity but keeps coherence.

top_k=40–100 → Only pick from the top K most likely tokens → curbs rare/outlandish outputs.

Effect: The model gets “imaginative” → good for brainstorming, storytelling, or exploring unusual phrasings.

What would you use for instructions that must be followed closely?

For instructions that must be followed closely, set do_sample=False and temperature=0.0 so the model is fully deterministic. This forces it to always pick the most likely token, keeping answers factual and consistent.