Large Language Models in Data Science

Week 2: Hugging Face & Transformers — Using Pretrained Models

Sebastian Mueller
Aix-Marseille Université
2025-2026

Session Overview

Lecture (1.5h)

1. HF ecosystem: Hub and Transformers
2. Installing and authenticating
3. Model selection and checkpoints
4. Pipelines for quick inference
5. Tokenizers, models, and generation APIs
6. Devices, memory, and caching

Lab (1.5h)

- Run pipelines for common NLP tasks
- Manual tokenization + forward pass
- Batch over a dataset with `datasets`
- Pin and cache a specific checkpoint

What is Hugging Face?

- **Hub (Definition)**: A public registry of *models*, *datasets*, and *spaces*. Each repo has a model card with usage, license, and metadata.
- **Why the Hub?** It centralizes discovery and reuse so we do not start from scratch.
- **Transformers (Definition)**: A Python library that downloads checkpoints from the Hub and provides high-level and low-level APIs for inference and training.
- **Datasets / Tokenizers (Definition)**: Libraries for efficient data loading and fast tokenization used by Transformers.
- **Focus today**: *Using pretrained models* safely and reproducibly (not contributing or fine-tuning yet).

Install and Authenticate

- **Goal**: Set up the minimal toolchain to run models locally or via hosted inference.
- **Install (Definition)**: Add the client libraries to your Python environment.
- **Authentication token (Definition)**: A personal access token that grants read access to private/gated repos and hosted endpoints.
- **Why authenticate?** Some checkpoints are gated due to license or size; hosted APIs require identifying the caller.
- **Cache (Definition)**: Local folder for downloaded configs/tokenizers/weights; defaults to `/.cache/huggingface`.
- **Why cache?** Avoid re-downloading and enable offline use. Configure via `HF_HOME` or `TRANSFORMERS_CACHE`.

Model Selection on the Hub

- **Model card (Definition)**: A README describing task(s), license, intended use, and usage examples.
- **Why read it?** Ensures the checkpoint matches your task and license constraints.
- **Task tags (Definition)**: Standard labels like `text-classification`, `summarization`, `text-generation`, `embeddings`.
- Prefer active repos with clear evals and permissive licenses where appropriate.

Checkpoints & Revisions: Ensuring Reproducibility

The Problem: Your results might change overnight

If you just use a model name like `"gpt2"`, you're getting the *latest* version. If the author updates it, your experiment from last week may produce different results.

Checkpoint (Definition)

- A specific trained model release (e.g., ‘meta-llama/Llama-2-7b-chat-hf‘).

Revision (Definition)

- A *specific version* of that model's checkpoint files, identified by a unique ID (a git commit hash).

The Solution: Pin Your Revision!

- **Why pin?** It guarantees you are using the **exact same** model files, ensuring your work is reproducible across machines and over time.

Model Safety: Code Execution Risks

The Problem: Models from the Hub are code from the internet!

Loading a model isn't just loading numbers; sometimes it can execute Python code. You need to know when and why this is happening.

Risk: Custom Code

- Some models require extra Python code from the repo to work.
- When you set `trust_remote_code=True`, you allow `transformers` to download and **run that code**.
- **Be careful!** Only do this if you trust the author.

Solution: Safe Tensors

- The old (`pytorch_model.bin`)is a known security risk that can execute code.
- **Safetensors** (`.safetensors`) is a new format that **cannot execute code**. It only contains the model's numbers (weights).

Pipelines: One-Liners for Common Tasks

- **Pipeline (Definition)**: A preconfigured wrapper that applies the right tokenizer, model, and postprocessing for a task.
- **Why use it?** Fast baseline and fewer moving parts for first runs.
- **Why these?** They cover common evaluation-style tasks: labeling, summarizing, and embeddings.

Pipelines: Examples

```python
from transformers import pipeline

# Sentiment analysis
clf = pipeline("text-classification",
               model="distilbert-base-uncased-finetuned-sst-2-english")
clf(["I love this!", "This is terrible..."])

# Masked language modeling (fill-mask)
mlm = pipeline("fill-mask", model="bert-base-uncased")
mlm("Paris is the [MASK] of France.")

# Text generation
gen = pipeline("text-generation", model="gpt2")
gen("Once upon a time", max_new_tokens=40, do_sample=True, temperature=0.8)
```

Other Useful Pipelines

```
# Zero-shot classification
zsc = pipeline("zero-shot-classification",
               model="facebook/bart-large-mnli")
zsc("The stock rallied 5%.", candidate_labels=["finance", "sports"])

# Summarization
summ = pipeline("summarization", model="facebook/bart-large-cnn")
summ(long_article_text, max_length=128)

# Embeddings (feature extraction)
emb = pipeline("feature-extraction", model="sentence-transformers/all-MiniLM-L6-v
vec = emb("A short sentence.", pooling="mean", normalize=True)
```

Manual Tokenization and Forward Pass

- **Tokenizer (Definition)**: Converts text to token IDs and attention masks that the model understands.
- **Model head (Definition)**: A task-specific layer (e.g., classification) on top of a base encoder/decoder.
- **Why manual mode?** More control over batching, padding, truncation, and outputs.
- `AutoTokenizer` / `AutoModelFor` select correct classes from the checkpoint config; inspect `config` for labels and limits.

Manual Tokenization and Forward Pass - Python example

```python
import torch
from transformers import AutoTokenizer, AutoModelForSequenceClassification

name = "distilbert-base-uncased-finetuned-sst-2-english"
tok = AutoTokenizer.from_pretrained(name)
model = AutoModelForSequenceClassification.from_pretrained(name)

batch = tok(["I love this!", "This is terrible..."],
           padding=True, truncation=True, return_tensors="pt")
with torch.no_grad():
    out = model(**batch)
probs = out.logits.softmax(-1)
probs
```

Generation with `generate()`

```python
from transformers import AutoModelForCausalLM, AutoTokenizer

name = "gpt2"
tok = AutoTokenizer.from_pretrained(name)
lm = AutoModelForCausalLM.from_pretrained(name)

prompt = "In data science, transformers are"
inputs = tok(prompt, return_tensors="pt")
outputs = lm.generate(**inputs,
                      max_new_tokens=64,
                      do_sample=True,
                      temperature=0.7,
                      top_p=0.9,
                      repetition_penalty=1.1)
print(tok.decode(outputs[0], skip_special_tokens=True))
```

Decoding: tuning generation parameters

- **Decoding (Definition)**: Strategy to choose next tokens (sampling vs. beam search).
- **Why tune it?** Balance creativity vs. determinism and mitigate repetition.
- **max_new_tokens**: Upper bound on generated tokens beyond the prompt. Higher = longer outputs; set a cap to avoid runaways.
- **temperature**: Scales logits before sampling. Lower (e.g., 0.2) = conservative; higher (e.g., 0.8) = more diverse.
- **top_p** (nucleus): Sample from the smallest set whose cumulative prob $\geq p$. Lower p = safer, fewer risky tokens.
- **top_k**: Sample from top-k probable tokens. Lower k = safer; can be combined with `top_p`.
- **repetition_penalty**: >1.0 discourages repeating tokens/phrases (e.g., 1.05–1.2). Too high can harm fluency.

Sampling vs. Beam Search

- **Sampling (Definition)**: At each step, draw the next token at random from a truncated distribution (controlled by `temperature`, `top_p`, `top_k`).
- **Beam search (Definition)**: Keep the top N partial sequences (*beams*) by cumulative log-probability; expand each beam and keep the best until stopping.
- **Determinism**: Sampling is non-deterministic (varies run-to-run); beam search is deterministic for fixed settings.
- **Diversity**: Sampling is more diverse/creative; beam search is safer but can be generic or repetitive.
- **Speed**: Sampling is fast; beam search is slower, roughly proportional to `num_beams`.
- **Controls**: Sampling uses `temperature`, `top_p`, `top_k`. Beam search uses `num_beams`, `length_penalty`, `early_stopping`.
- **When to use**: Creative writing/brainstorming $\rightarrow$ sampling. Single best answer (e.g., translation/QA) $\rightarrow$ greedy/beam search.

Devices, Dtypes, and Memory

- **Device map (Definition)**: Automatic placement of model layers across CPU/GPU/MPS via Accelerate.
- **Why?** Fit larger models and use available accelerators without manual plumbing.
- **Dtype (Definition)**: Numeric precision used for weights and activations (e.g., `float32`, `float16`).
- **Why lower precision?** Save memory and increase throughput with minimal quality drop.
- **Quantization (Definition)**: Load weights in 8/4-bit (`bitsandbytes`) to further reduce memory.

Caching, Offline, and Reproducibility

- **Cache dirs (Definition)**: `TRANSFORMERS_CACHE`, `HF_HOME` control where files are stored.
- **Why set them?** Keep caches on faster disks or shared locations.
- **Offline mode (Definition)**: `HF_HUB_OFFLINE=1` forces use of local cache only.
- **Why offline?** Stable experiments and air-gapped environments.
- Pin exact versions with `revision=` and store `model.config.to_dict()` with results for traceability.

Hosted Inference with `InferenceClient`

```python
from huggingface_hub import InferenceClient

client = InferenceClient(
    model="facebook/bart-large-cnn", token=os.getenv("HF_TOKEN")
)
summary = client.summarization("""Long article text here...""")
print(summary)
```

- **Inference API (Definition)**: Managed endpoints for common tasks; billable and rate-limited.
- **Why use it?** Zero local setup for quick demos or when hardware is unavailable.

Key Takeaways

- Start with **pipelines** for quick wins; drop to tokenizers/models for control.
- Pick checkpoints by **task, license, and metrics**; pin with `revision=`.
- Use **device_map** and dtypes to fit memory and speed constraints.
- Cache wisely; enable offline mode for reproducibility.

Glossary Cheat Sheet

- **Hub**: Registry of models/datasets/spaces with model cards.
- **Transformer**: Library for loading checkpoints and running tasks.
- **Checkpoint**: Released weights/config for a model; pin with `revision=`.
- **Pipeline**: One-line task runner that bundles tokenizer+model.
- **Tokenizer**: Maps text to token IDs and attention masks.
- **Device map**: Automatic layer placement across CPU/GPU/MPS.
- **Dtype/Quantization**: Numeric precision and compressed weight formats.
- **Dataset**: Table-like data with fast transforms and batching.
- **Cache/Offline**: Local storage and network-free operation.