The Hugging Face Guide

From Beginner to Intermediate: A Visual Approach

Mausam Kar

Contents

1 Introduction to Hugging Face **1**
- 1.1 What is Hugging Face? . . . 1
- 1.2 A Brief History: Before Hugging Face . . . 1
- 1.3 Why is it so Popular? . . . 1
- 1.4 Industry Adoption . . . 2

2 Core AI Terminologies **3**
- 2.1 The AI Hierarchy . . . 3
- 2.2 Tokens & Tokenization . . . 4
- 2.3 Pre-trained Models vs. Fine-tuning . . . 4
- 2.4 Inference . . . 4

3 Transformers Made Easy **5**
- 3.1 The Intuition: Attention Mechanism . . . 5
- 3.2 Why Transformers replaced RNNs . . . 5
- 3.3 High-Level Architecture . . . 5
- 3.4 Encoder vs. Decoder Models . . . 6

4 The Hugging Face Ecosystem **7**
- 4.1 The Core Libraries . . . 7
- 4.2 The Hub . . . 7
- 4.3 Spaces: Showcasing Your Work . . . 7

5 Installation & Setup **8**
- 5.1 Prerequisites . . . 8
- 5.2 Virtual Environments . . . 8
- 5.3 Installing Libraries . . . 8
- 5.4 Hardware Acceleration (Optional) . . . 9
- 5.5 Where are models stored? . . . 9

6 Using Hugging Face Locally **10**
- 6.1 Sentiment Analysis . . . 10
- 6.2 Text Summarization . . . 10
- 6.3 Question Answering . . . 11
- 6.4 Zero-Shot Classification . . . 11

7 Using Hugging Face via API **12**
- 7.1 What is an API? . . . 12
- 7.2 Local vs. API . . . 12
- 7.3 Using the Inference API . . . 12

8 Downloading Models **14**
8.1 Understanding Model IDs . . . 14
8.2 How Caching Works . . . 14
8.3 Loading Specific Components . . . 14
8.4 Model Cards . . . 15
8.5 Version Control & Reproducibility . . . 15

9 Real-World Project Ideas **16**
9.1 1. Resume ATS Checker . . . 16
9.2 2. AI Chatbot Companion . . . 16
9.3 3. Multilingual Review Translator . . . 16
9.4 4. "Too Long; Didn't Read" News Summarizer . . . 17
9.5 5. Intelligent Flashcard Generator . . . 17

10 Terminology Cheat Sheet **18**

11 Common Mistakes & Best Practices **19**
11.1 1. The "CPU vs. GPU" Trap . . . 19
11.2 2. Ignoring Model Size vs. Performance . . . 19
11.3 3. Breaking API Rate Limits . . . 19
11.4 4. Not Checking the License . . . 20
11.5 5. Forgetting Tokenizer Padding . . . 20

12 Learning Roadmap **21**
12.1 Recommended Resources . . . 21

Chapter 1

Introduction to Hugging Face

1.1 What is Hugging Face?

Hugging Face is often referred to as the **"GitHub of Machine Learning"**. It is an open-source platform and community that provides tools to build, train, and deploy Machine Learning (ML) models, primarily focused on Natural Language Processing (NLP), but rapidly expanding into Computer Vision and Audio.

At its core, Hugging Face democratizes Artificial Intelligence (AI) by making state-of-the-art models accessible to everyone, not just tech giants. Before Hugging Face, using state-of-the-art models required writing hundreds of lines of complex TensorFlow or PyTorch code. Now, it takes just a few lines.

Definition: Hugging Face Hub

The central repository where anyone can share and discover models, datasets, and demo apps (Spaces). It hosts hundreds of thousands of pre-trained models, allowing you to "stand on the shoulders of giants" rather than starting from scratch.

1.2 A Brief History: Before Hugging Face

To appreciate Hugging Face, we must look at what came before:

- **Rule-Based Systems:** Early chatbots relied on hard-coded rules (If user says "Hi", reply "Hello"). They failed at handling nuances.
- **Statistical Models:** Used math to predict probability of words. They were better but struggled with context.
- **RNNs & LSTMs:** The first neural networks that could handle sequences. However, they were slow to train and forgot information in long sentences.

Then, in 2017, Google released the **Transformer** paper ("Attention Is All You Need"), which revolutionized the field. Hugging Face released the `transformers` library to make this new architecture easy to use for everyone.

1.3 Why is it so Popular?

In the past, using a powerful model like BERT or GPT required complex code, massive computing power, and weeks of training. Hugging Face changed this with the `transformers` library.

1. **Ease of Use:** You can download and use a state-of-the-art model in just 3 lines of Python code.
2. **Open Source:** Most models are free to use and modify. The community ethos is strong, with "open science" being a core value.
3. **Community Driven:** Researchers from Meta, Google, and Microsoft publish their models on the Hub. When Facebook released LLaMA, it was on Hugging Face within hours.

1.4 Industry Adoption

Top companies use Hugging Face for various tasks:

- **Google & Microsoft:** For research and model hosting.
- **Grammarly:** For text correction, grammar checking, and tone improvement.
- **Salesforce:** For code generation (CodeT5) and customer service AI.
- **Intel & NVIDIA:** optimize their hardware to run Hugging Face models efficiently.

Tip

Hugging Face isn't just for NLP anymore! It now supports Computer Vision (images), Audio processing, and even Reinforcement Learning. You can find models for identifying objects in images, transcribing speech to text, and playing video games.

Chapter 2

Core AI Terminologies

To understand Hugging Face, we must first understand the hierarchy of AI concepts. Many beginners use terms like AI, ML, and DL interchangeably, but they have distinct meanings.

2.1 The AI Hierarchy

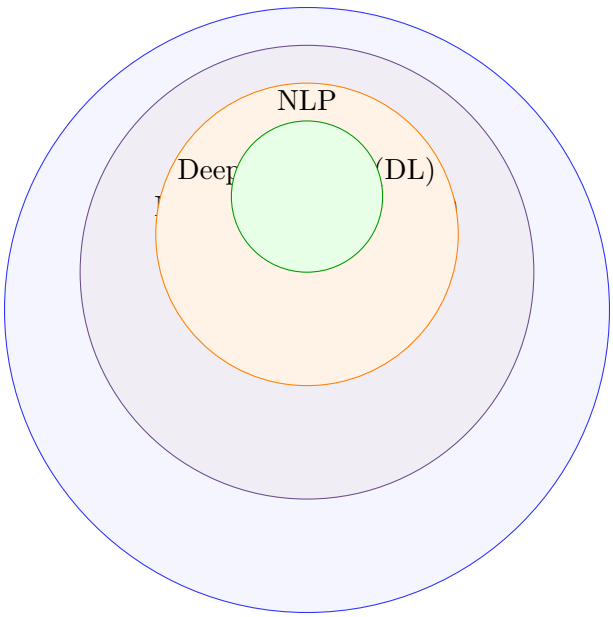

Figure 2.1: The relationship between AI, ML, Deep Learning, and NLP.

- **AI (Artificial Intelligence):** The broad conceptual umbrella. It refers to any technique that enables computers to mimic human intelligence, including logic, rules, and learning.
- **ML (Machine Learning):** A subset of AI that uses specific algorithms to learn patterns from data (e.g., predicting house prices based on size and location) without being explicitly programmed for every rule.
- **DL (Deep Learning):** A subset of ML inspired by the structure of the human brain (neural networks). It uses many layers of artificial neurons to solve complex problems like image recognition and language modeling.
- **NLP (Natural Language Processing):** The field of AI focused on understanding, interpreting, and generating human language (text and speech).

2.2 Tokens & Tokenization

Computers don't understand text; they understand numbers. Tokenization is the process of breaking text into smaller chunks called *tokens*.

There are different ways to tokenize:

- **Word-based:** Splitting by spaces. (e.g., "AI", "is", "cool"). Problem: Vocabulary becomes huge.
- **Character-based:** Splitting by letters. (e.g., "A", "I"). Problem: Sequences become too long.
- **Subword-based (Standard):** A balance. Common words are kept whole, rare words are split. (e.g., "tokenization" → "token", "##ization").

Example: Tokenization Example

Input: "Hugging Face is acts cool!"
Tokens: ["Hugging", "Face", "is", "acts", "cool", "!"]

Each token is then converted into a unique ID number, which is what the model actually processes.

2.3 Pre-trained Models vs. Fine-tuning

This is the most important concept in modern NLP: **Transfer Learning**.

Definition: Pre-trained Model (The Generalist)

A model that has been trained on a massive amount of data (like the whole internet, Wikipedia, Books) to learn general language patterns. It understands grammar, slang, and reasoning.
Analogy: A student who has graduated from high school with general knowledge.

Definition: Fine-tuning (The Specialist)

Taking a pre-trained model and training it further on a small, specific dataset (e.g., medical records, legal documents) to maximize performance for a specific task.
Analogy: The high school graduate going to medical school to become a doctor.

2.4 Inference

Inference is simply the act of **using** the model to make a prediction on new data. When you type a prompt into ChatGPT and it replies, that is inference.

Warning

Training (especially pre-training) takes a massive amount of compute power (GPUs) and money. Inference (using) takes much less, often running on a single GPU or even a CPU.

Chapter 3

Transformers Made Easy

The **Transformer** architecture is the engine behind the modern AI revolution. But how does it work?

3.1 The Intuition: Attention Mechanism

Imagine you are at a loud cocktail party. There are hundreds of people talking at once. However, you can focus on just one person's voice regarding the conversation you are having, filtering out the background noise. This is "Attention."

In NLP, when you translate a sentence, you don't just translate word-by-word; you look at the whole sentence to understand the context.

Example: Context Matters

"I went to the **bank** to deposit money."
"I sat on the river **bank**."

In these sentences, the word "bank" has completely different meanings. Old models (like RNNs) struggled with this because they looked at words one by one. **Transformers look at all words at once** and pay "attention" to the relevant ones to understand the context. In the first sentence, the model pays attention to "deposit" and "money" to understand that "bank" refers to a financial institution.

3.2 Why Transformers replaced RNNs

- **Parallelism:** RNNs process words sequentially (Step 1, then Step 2). This is slow. Transformers process the whole sentence at once. This allows for massive parallelization on GPUs.
- **Long-term Dependency:** RNNs forget the beginning of a long paragraph by the time they reach the end. Transformers "see" everything simultaneously, maintaining context over long distances.

3.3 High-Level Architecture

Here is the simplified flow of data in a Transformer model.

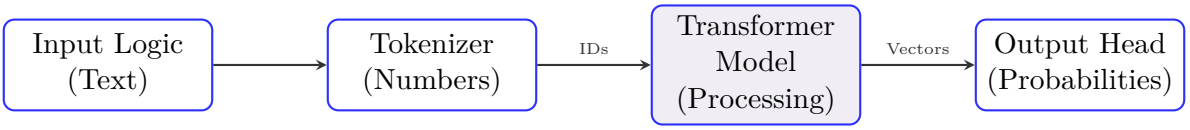

Figure 3.1: How data flows through a Transformer model.

1. **Input:** Raw text.
2. **Tokenizer:** Converts text to numbers (Input IDs).
3. **Transformer:** Processes these numbers using layers of attention to create "Embeddings" (rich numerical representations of meaning).
4. **Output Head:** Converts the processed data into a final prediction (e.g., "Positive Sentiment" or the next word in a sentence).

3.4 Encoder vs. Decoder Models

Transformers come in three flavors:

- **Encoder-only (e.g., BERT):** Great for "understanding" text. Used for classification, sentiment analysis, and answering questions.
- **Decoder-only (e.g., GPT):** Great for "generating" text. Used for auto-complete and chatbots.
- **Encoder-Decoder (e.g., T5, BART):** Good for tasks that strictly map Input $\rightarrow$ Output, like Translation or Summarization.

Chapter 4

The Hugging Face Ecosystem

Hugging Face is not just one library; it is a collection of tools designed to work together seamlessly.

4.1 The Core Libraries

`transformers` The main library. It contains the model architectures (BERT, GPT, T5) and allows you to download pre-trained weights.

`tokenizers` A super-fast library (written in Rust) to convert text into numbers for the models.

`datasets` Provides easy access to thousands of datasets for training. It handles downloading, caching, and memory management efficiently.

`accelerate` A helper library to run your code on any hardware (CPU, GPU, TPU) without changing your code.

`evaluate` Tools to measure how good your model is (Accuracy, BLEU score, etc.).

4.2 The Hub

The Hugging Face Hub is like the App Store for AI.

- **Models:** Over 500,000 public models.
- **Datasets:** Structured data for training.
- **Spaces:** A place to host and showcase your ML demos.

4.3 Spaces: Showcasing Your Work

You can deploy your model as a web app using **Gradio** or **Streamlit** directly on Hugging Face Spaces.

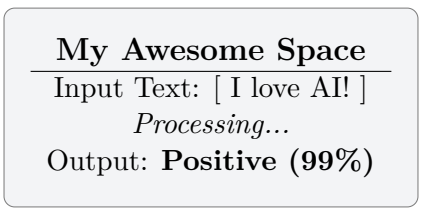

Figure 4.1: A simple representation of a Gradio interface.

Chapter 5

Installation & Setup

Setting up your environment correctly is the first step to success. This chapter will guide you through creating a professional development setup.

5.1 Prerequisites

You need Python installed on your system. We recommend Python 3.8 or higher. You can check your version by running:

```
python --version
```

5.2 Virtual Environments

It is highly recommended to use a virtual environment. Installing libraries globally can break your system tools or cause version conflicts.

Definition: Virtual Environment

A self-contained directory that contains a specific Python installation for a project. It isolates your project's dependencies from others.

```
# 1. Open your terminal/command prompt

# 2. Create environment named 'hf-env'
python -m venv hf-env

# 3. Activate it (Windows)
hf-env\Scripts\activate

# 3. Activate it (Mac/Linux)
source hf-env/bin/activate
```

Listing 5.1: Creating a Virtual Environment

5.3 Installing Libraries

Once your environment is active (you should see `(hf-env)` in your terminal), install the Hugging Face libraries using `pip`.

```
# Updates pip to the latest version
pip install --upgrade pip

# Install the core libraries
pip install transformers datasets torch
```

Listing 5.2: Installing Transformers and PyTorch

Tip

We install `torch` (PyTorch) because it is the most popular backend for Hugging Face. The library also works with TensorFlow (`tensorflow`) and JAX (`jax`).

5.4 Hardware Acceleration (Optional)

If you have an NVIDIA GPU, you should install the specific version of PyTorch that supports CUDA to speed up training by 10-50x. Visit https://pytorch.org/get-started/locally/ to get the exact command for your hardware.

5.5 Where are models stored?

Hugging Face models can be several gigabytes in size. When you download a model, it is stored in your cache directory to avoid downloading it again.

- **Default Location:** `~/.cache/huggingface/hub`

You can change this by setting the `HF_HOME` environment variable if your C: drive runs out of space.

Chapter 6

Using Hugging Face Locally

The easiest way to use Hugging Face is through the `pipeline()` function. It handles all the complexity of pre-processing, model inference, and post-processing, wrapping it all into a single function call.

6.1 Sentiment Analysis

Let's begin with a classic NLP task: analyzing if a sentence is positive or negative.

```python
from transformers import pipeline

# 1. Download the model automatically
# By default, this loads a DistilBERT model finetuned on sst-2
classifier = pipeline("sentiment-analysis")

# 2. Use it on a single sentence
result = classifier("I love learning about AI!")
print(result)
# Output: [{'label': 'POSITIVE', 'score': 0.999}]

# 3. Use it on a list of sentences (Process in batch)
results = classifier([
    "This works great!",
    "I am not happy with the service."
])
print(results)
# Output: [
#   {'label': 'POSITIVE', 'score': 0.99...},
#   {'label': 'NEGATIVE', 'score': 0.98...}
# ]
```

Listing 6.1: Sentiment Analysis with Pipeline

6.2 Text Summarization

Summarizing a long paragraph into a few sentences is a "Sequence-to-Sequence" task.

```python
summarizer = pipeline("summarization")

text = """
Hugging Face is a company that develops tools for building applications
    using machine learning.
```

```
It is most notable for its transformers library built for natural
    language processing applications
and its platform that allows users to share machine learning models and
     datasets.
"""

# max_length controls the output size in tokens
summary = summarizer(text, max_length=30, min_length=10, do_sample=
    False)
print(summary[0]['summary_text'])
```

Listing 6.2: Summarization

6.3 Question Answering

The model can extract an answer from a given context.

```
qa_model = pipeline("question-answering")

context = "My name is Mausam and I live in Bangalore."
question = "Where do I live?"

answer = qa_model(question=question, context=context)
print(answer)
# Output: {'score': 0.98, 'start': 31, 'end': 40, 'answer': 'Bangalore
    '}
```

Listing 6.3: Question Answering

6.4 Zero-Shot Classification

This is magic. You can classify text into categories the model has *never seen before* during training.

```
classifier = pipeline("zero-shot-classification")

text = "Star Wars is a space opera franchise created by George Lucas."
candidate_labels = ["politics", "science", "movies"]

res = classifier(text, candidate_labels)
print(res)
# The model correctly assigns the highest score to "movies"
# even specifically trained on this sentence.
```

Listing 6.4: Zero-Shot Classification

Tip

The `pipeline` function automatically chooses a default model for the task. You can specify a specific model using the `model="model_name"` argument.

Chapter 7

Using Hugging Face via API

Sometimes, you don't want to download a huge model locally. Maybe your laptop doesn't have a GPU, or you just want to test something quickly. This is where the **Inference API** comes in.

7.1 What is an API?

Definition: API (Application Programming Interface)

Think of an API as a waiter in a restaurant. You (the code) ask the waiter (API) for a specific dish (prediction). The waiter goes to the kitchen (server), gets the dish, and brings it back. You don't need to know how to cook (run the model).

7.2 Local vs. API

Local (Pipeline)	**API (Serverless)**
Running on YOUR computer	Running on Hugging Face servers
Needs RAM/GPU	No hardware required
Works offline	Requires internet
Free	Free (with rate limits) or Paid

7.3 Quick Setup: Getting a Token

To use the API, you identify yourself using a "Token" (like a password).

1. Go to https://huggingface.co/settings/tokens
2. Click "New token".
3. Give it a name (e.g., "Book Project") and select "Read" role.
4. Copy the string starting with `hf_...`.

7.4 Using the Inference API (Python)

Below is a standard function to call ANY model on the Hub.

```
import requests

API_URL = "https://api-inference.huggingface.co/models/google-bert/bert-base-uncased"
```

```
headers = {"Authorization": "Bearer hf_xxxxxxxxxxxxxxxxxxxxxxx"}

def query(payload):
    response = requests.post(API_URL, headers=headers, json=payload)
    return response.json()

output = query({
    "inputs": "The quick brown fox jumps over the lazy dog.",
})
print(output)
```

Listing 7.1: Universal API Function

7.5 Advanced Examples

The API isn't just for text! You can use it for images too.

7.5.1 Text Generation (GPT-2)

```
API_URL = "https://api-inference.huggingface.co/models/gpt2"
# Payload parameters allow you to control creativity
data = query({
    "inputs": "Once upon a time,",
    "parameters": {"max_new_tokens": 50, "temperature": 0.7}
})
```

7.5.2 Image Captioning

```
API_URL = "https://api-inference.huggingface.co/models/Salesforce/blip-
    image-captioning-base"
# You would send raw image bytes here
with open("cat.jpg", "rb") as f:
    data = f.read()
response = requests.post(API_URL, headers=headers, data=data)
```

7.6 Common Errors & Troubleshooting

- **503 Service Unavailable:** The model is "cold" (loading into memory). Wait 10 seconds and try again.
- **401 Unauthorized:** Your token is wrong. Check if you copied it correctly.
- **400 Bad Request:** Your input format is wrong (e.g., sending text to an image model).

Tip

The "Serverless" Inference API is free but rate-limited. If you need to hit it 100 times a second for a real App, you should upgrade to "Inference Endpoints" which gives you a dedicated GPU.

Chapter 8

Downloading Models

8.1 Understanding Model IDs

Every model on the Hub has a unique ID formatted as `username/model-name`.

- `google-bert/bert-base-uncased` (Organization / Model)
- `gpt2` (Official library models may not have a username)
- `facebook/bart-large-cnn`

8.2 How Caching Works

When you run `pipeline("sentiment-analysis")`, Hugging Face checks your local cache first.

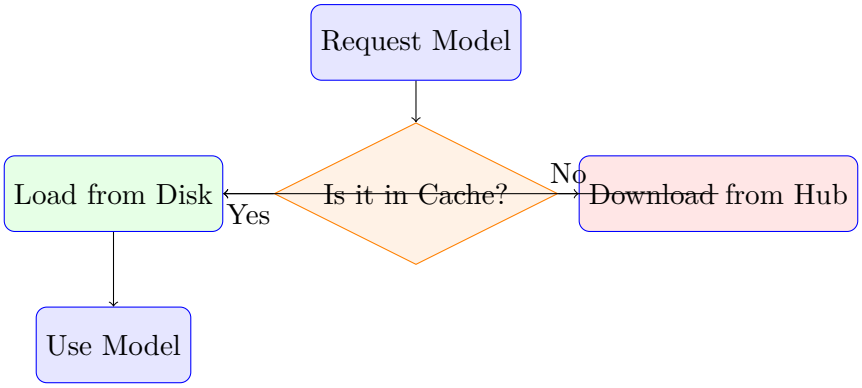

Figure 8.1: The Caching Mechanism

8.3 Loading Specific Components

Instead of using `pipeline`, you can load the tokenizer and model separately for more control.

```
from transformers import AutoTokenizer,
    AutoModelForSequenceClassification

model_name = "distilbert-base-uncased-finetuned-sst-2-english"

# 1. Load Tokenizer
tokenizer = AutoTokenizer.from_pretrained(model_name)

# 2. Load Model
```

```
model = AutoModelForSequenceClassification.from_pretrained(model_name)
```

Listing 8.1: Manual Loading

8.4 Model Cards

Every model on the Hub comes with a **Model Card**, which is a `README.md` file explaining:

- **Model Description:** What the model does.
- **Intended Use:** What limitations it has (e.g., "Do not use for medical advice").
- **Training Data:** What data it was trained on.

Best Practice: Always read the Model Card before using a model in production!

8.5 Version Control & Reproducibility

Hugging Face is built on top of **Git**. This means every change to a model is tracked. To ensure your code always uses the *exact same version* of a model (even if the author updates it later), you can use the `revision` parameter.

```
# Load a specific version of the model using the Commit Hash
model = AutoModel.from_pretrained(
    "bert-base-uncased",
    revision="a8041bf617dce44"
)
```

Listing 8.2: Loading a Specific Commit

This guarantees that your application will behave exactly the same way in the future, protecting you from breaking changes.

Chapter 9

Real-World Project Ideas

Bridging the gap between theory and practice is essential. Here are five project ideas you can build using Hugging Face models, ranging from beginner to intermediate.

9.1 1. Resume ATS Checker

The Problem: Job applicants want to know if their resume matches a job description.
The Solution: Use a **Zero-Shot Classification** model. These models can classify text into arbitrary categories without specific training examples.
Workflow:

1. **Input:** The full text of a candidate's resume (PDF to Text).
2. **Labels:** Skills required (e.g., "Python", "Data Analysis", "Project Management").
3. **Model:** `facebook/bart-large-mnli` (Robust and accurate).
4. **Output:** A probability score for each skill. High scores mean the resume is a good match.

Key Library: `transformers.pipeline("zero-shot-classification")`

9.2 2. AI Chatbot Companion

The Problem: Building a chatbot from scratch is hard and rule-based bots are boring.
The Solution: Use a pre-trained conversational model trained on movie scripts or internet dialogue.
Workflow:

1. **Model:** `microsoft/DialoGPT-medium` or `facebook/blenderbot-400M-distill`.
2. **Loop:** Take user input → Tokenize → Generate Response → Decode.
3. **Challenge:** Keep a "history" list of past inputs so the bot remembers the conversation context. Without history, the bot has short-term memory loss.

9.3 3. Multilingual Review Translator

The Problem: An app has reviews in Spanish, French, and Hindi, but the analytics team only speaks English.
The Solution: A universal translation pipeline.
Workflow:

1. **Language ID:** Use `papluca/xlm-roberta-base-language-detection` to detect the language.
2. **Translate:** Use `facebook/m2m100_418M` (Many-to-Many) to translate any language to English.
3. **Analyze:** improved sentiment analysis on the translated text.

9.4 4. "Too Long; Didn't Read" News Summarizer

The Problem: News articles are too long and full of fluff.
The Solution: An abstractive summarizer that rewrites the core meaning.
Workflow:

1. **Scraper:** Use `BeautifulSoup` to get text from a URL.
2. **Model:** `sshleifer/distilbart-cnn-12-6` (Fast and light).
3. **Settings:** Set `max_length=60` to force a conciseness.

9.5 5. Intelligent Flashcard Generator

The Problem: Creating study notes takes time.
The Solution: Convert textbook photos into Q&A pairs using Vision + Language models.
Workflow:

1. **OCR:** Use `microsoft/trocr-base-handwritten` to read text from an image.
2. **Question Generation:** Use a T5 model fine-tuned on "answer2question" tasks (like `mrm8488/t5-base-finetuned-question-generation-ap`).
3. **Output:** A JSON file of Question-Answer pairs ready for Anki.

Example: Project Flow Visualization

Resume Text
↓
Model Layer (BART-MNLI)
↓
"Python: 99%, Java: 12%"

Chapter 10

Terminology Cheat Sheet

A quick reference for common AI and Hugging Face terms.

Term	Simple Explanation
Model	The AI program (the brain) that makes predictions.
Weights	The learned numbers inside the model (its memory).
Checkpoint	A saved version of the model during training.
BOS / EOS	Special tokens marking the **B**eginning and **E**nd **O**f a **S**entence.
Epoch	One full cycle of training through the entire dataset.
Batch Size	How many examples the model looks at at once.
Learning Rate	How fast the model learns (too fast = simple mistakes, too slow = takes forever).
Loss	The error score (lower is better).
Fine-tuning	Tweaking a pre-trained model for your specific data.
Zero-shot	Using a model to do a task it wasn't explicitly trained for.

Chapter 11

Common Mistakes & Best Practices

Even experienced developers make mistakes when switching to modern NLP. Here is a curated list of pitfalls and how to avoid them.

11.1 1. The "CPU vs. GPU" Trap

Warning

Do NOT try to run large models (like Llama-2-70b or Falcon-40b) on a standard laptop CPU. It will either crash your RAM or take minutes to generate a single word.

The Reality: Deep Learning is matrix multiplication heavy. GPUs are designed for this; CPUs are not.
Solution:

- Start with smaller models (`distilbert`, `t5-small`) for local development.
- Use Google Colab (free T4 GPU) or Kaggle Kernels.
- Use `device="cuda"` or `device="mps"` (Mac) in your pipeline.

11.2 2. Ignoring Model Size vs. Performance

Bigger is not always better. A massive model might give 99% accuracy, but a tiny model might give 98% accuracy while running 100x faster.

- **Bert-Base (110M params):** Good balance. Good for almost all production use-cases.
- **Bert-Large (340M params):** Slightly better accuracy, much slower.
- **DistilBERT (66M params):** 40% smaller, 60% faster, retains 97% of BERT's performance.

Best Practice: Always start small! Scale up only if accuracy is insufficient.

11.3 3. Breaking API Rate Limits

If you use the free Inference API in a production loop, you will get blocked (Error 429).

```
# BAD PRACTICE
for text in million_sentences:
    # This will get you temporarily banned!
    requests.post(API_URL, json=text)
```

Solution:

- Use local interference if you have hardware.
- Pay for an "Inference Endpoint" (dedicated GPU).
- Use `time.sleep()` between requests (slow).

11.4 4. Not Checking the License

Just because a model is on the Hub doesn't mean it's open source for *commercial* use.

- **Apache 2.0 / MIT:** Safe for almost anything.
- **Llama Community License:** Has restrictions on user count (>700M users).
- **Creative Commons Non-Commercial (CC-BY-NC):** You cannot sell a product using this.

Always check the `LICENSE` file on the Model Card.

11.5 5. Forgetting Tokenizer Padding

When processing batches of sentences, they must be the same length. If you don't pad them, your code will crash.

```
# Correct way to handle batching
tokenizer(sentences, padding=True, truncation=True)
```

This ensures all sequences are padded with zeros to match the longest sentence in the batch.

Chapter 12

Learning Roadmap

Congratulations on finishing this guide! Where do you go from here?

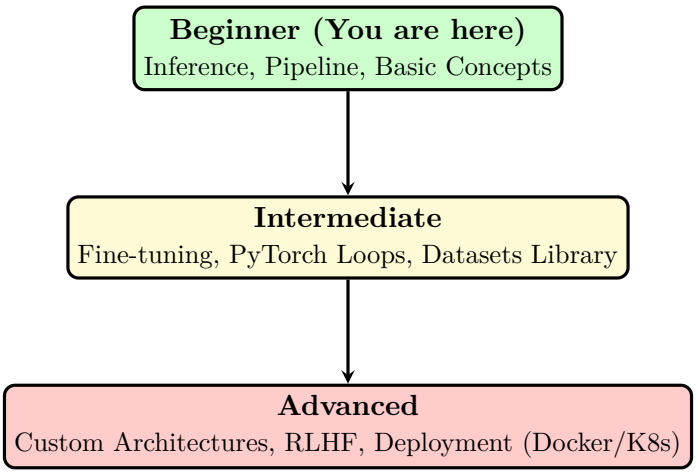

Figure 12.1: Your Path Forward

12.1 Recommended Resources

- **Hugging Face Course:** `https://huggingface.co/course` (The Bible of HF).
- **Fast.ai:** Practical Deep Learning for Coders.
- **ArXiv.org:** To read the latest research papers (for advanced users).

Keep building, keep learning!