### What’s the difference?

### NLP (Natural Language Processing) is the broader field focused on enabling computers to understand, interpret, and generate human language. NLP encompasses many techniques and tasks such as sentiment analysis, named entity recognition, and machine translation.

### LLMs (Large Language Models) are a powerful subset of NLP models characterized by their massive size, extensive training data, and ability to perform a wide range of language tasks with minimal task-specific training. Models like the Llama, GPT, or Claude series are examples of LLMs that have revolutionized what’s possible in NLP.

### **NLP is a field of linguistics and machine learning focused on understanding everything related to human language. The aim of NLP tasks is not only to understand single words individually, but to be able to understand the context of those words.**

### **Slide 1: Title Slide**

**Title:** Natural Language Processing and Large Language Models  
 **Subtitle:** Understanding the Evolution of Language Understanding in AI  
 **Presented by:** [Your Name / Institution]  
 **Date:** [Insert Date]

### **Slide 2: Agenda**

* What is NLP?
* Common NLP Tasks
* Rise of Large Language Models (LLMs)
* Capabilities of LLMs
* Challenges & Limitations
* Why Language is Hard for Machines
* Summary & Next Steps

### **Slide 3: What is Natural Language Processing (NLP)?**

* **Definition:** NLP is a field at the intersection of linguistics and machine learning, focused on understanding and generating human language.
* **Goal:** Not just understanding individual words, but understanding **context**.

### **Slide 4: Common NLP Tasks**

|  |  |
| --- | --- |
| **Task Type** | **Examples** |
| Sentence Classification | Sentiment analysis, spam detection, grammatical correctness |
| Word Classification | Part-of-speech tagging, named entity recognition |
| Text Generation | Auto-completion, masked word prediction |
| Question Answering | Extracting answer from context |
| Text-to-Text Tasks | Translation, summarization |

### **Slide 5: Beyond Text: Multimodal NLP**

* **Speech recognition:** Audio to text
* **Vision-language:** Describing images with natural language
* **Example:** Voice assistants, image captioning

### **Slide 6: Rise of Large Language Models (LLMs)**

* **What is an LLM?** A model trained on massive datasets to understand and generate human-like text
* **Examples:** GPT, LLaMA, Claude, PaLM
* **Paradigm Shift:** From task-specific models → general-purpose language models

### **Slide 7: Key Characteristics of LLMs**

* **Scale:** Billions of parameters
* **Generalization:** Multi-task capability without retraining
* **In-Context Learning:** Learns from prompt examples
* **Emergent Abilities:** Surprising new behaviors at scale

### **Slide 8: Why LLMs Are Transformational**

* Single model, many uses:  
  + Translation
  + Summarization
  + Coding
  + Q&A
  + Story writing
* **Democratization of NLP capabilities**

### **Slide 9: Limitations of LLMs**

* **Hallucinations:** Confidently incorrect outputs
* **Lack of true understanding:** Purely statistical
* **Bias:** Reflects societal and data biases
* **Context Limits:** Finite input size
* **Computational Cost:** Training & inference are expensive

### **Slide 10: Why is Language Hard for Machines?**

* **Ambiguity:** "I saw her duck" (animal or action?)
* **Context & Nuance:** Cultural, emotional, sarcastic
* **Similarity Judgments:** "I am hungry" vs. "I am sad" – obvious to humans, not to machines
* **Representation Challenge:** Text must be converted into something machines can learn from

### **Slide 11: Despite Progress, Challenges Remain**

* **True comprehension** is still elusive
* **Sarcasm, humor, regional language** remain hard
* Continuous improvement through:  
  + Better architectures
  + Richer datasets
  + Responsible fine-tuning

### **Slide 12: Summary**

* NLP is foundational to AI language understanding
* LLMs have transformed what’s possible in NLP
* Many real-world applications already in use
* But limitations still exist: ethical, technical, and cognitive

### **Slide 13: What’s Next?**

* Deep dive into Transformer architecture
* Understanding embeddings, attention, and pretraining
* Exploring fine-tuning and prompt engineering
* Hands-on with Hugging Face tools

### **Slide 14: Thank You**

**Q&A** *“Ask a Question” – Let's explore together!*

### **🛋️ Detailed Modules**

#### **Module 1: Introduction to Hugging Face**

* The rise of Transformers and their impact
* Hugging Face’s mission and community
* Core libraries:  
  + transformers: model loading & usage
  + datasets: standard NLP dataset access
  + evaluate: evaluation metrics
  + accelerate: multi-GPU/memory optimization
  + gradio/streamlit: app deployment interfaces

#### **Module 2: Hugging Face Hub**

* Model page overview: description, usage, tags
* Dataset page overview: tasks, data preview, scripts
* Spaces overview: what, why, and how
* Model card creation & community contributions

#### **Module 3: Transformers Library**

* Installation:

pip install transformers

* Using pipeline():

from transformers import pipeline

classifier = pipeline("sentiment-analysis")

classifier("Hugging Face is awesome!")

* Core classes:  
  + AutoTokenizer
  + AutoModel, AutoModelForSequenceClassification, etc.
  + AutoConfig
* Using from\_pretrained() and save\_pretrained()

#### **Module 4: Tokenizers**

* Role of tokenizers in NLP
* Pretrained vs custom tokenizers
* Types:  
  + WordPiece (BERT)
  + Byte-Pair Encoding (GPT-2)
  + Unigram (XLNet)
* Tokenization methods:

tokens = tokenizer("Hello world!", return\_tensors="pt")

* Decoding and detokenizing:

text = tokenizer.decode(tokens['input\_ids'][0])

#### **Module 5: Inference with Transformers**

* Tasks:  
  + Sentiment analysis
  + Named Entity Recognition (NER)
  + Question Answering
  + Text generation (GPT-2)
  + Summarization (T5, BART)
* Example:

qa = pipeline("question-answering")

qa({"context": "Hugging Face creates Transformers.", "question": "Who creates Transformers?"})

#### **Module 6: Fine-tuning Transformers**

* Hugging Face Trainer API:  
  + TrainingArguments
  + Trainer class
  + Training loop customization
* Steps:  
  + Load dataset
  + Preprocess and tokenize
  + Define model
  + Train and evaluate
* Saving model and pushing to hub

#### **Module 7: Datasets Library**

* Loading built-in datasets:

from datasets import load\_dataset

dataset = load\_dataset("imdb")

* Dataset operations:  
  + Filtering
  + Mapping
  + Shuffling
* Tokenization integration

#### **Module 8: Accelerate & PEFT**

* accelerate for easy training on CPU/GPU/TPU
* Basics of mixed precision training
* Parameter-Efficient Fine-Tuning:  
  + LoRA (Low Rank Adaptation)
  + Quantization for smaller memory use

#### **Module 9: Hugging Face Spaces**

* Overview of Gradio and Streamlit
* Deploying a model with a UI
* Uploading to Hugging Face Spaces
* Example sentiment analyzer web app

#### **Module 10: Real-World Projects**

* **Project 1**: Sentiment Analysis Web App (Gradio + BERT)
* **Project 2**: News Summarizer using T5
* **Project 3**: Question Answering Chatbot (DistilBERT + Streamlit)
* **Project 4**: Domain-Specific Classification (custom CSV data)

#### **Module 11: Deployment & Sharing**

* push\_to\_hub() method to upload models/datasets
* Creating model cards
* Using inference API:

from transformers import pipeline

pipe = pipeline(task="text-generation", model="gpt2")

pipe("Once upon a time")

### **🧪 Lab Exercises**

* Load a model and test with pipeline
* Train sentiment model on small dataset (Yelp reviews)
* Tokenizer exploration lab
* Build and host a Gradio app for summarization
* Upload a fine-tuned model to Hugging Face Hub

### **📖 Resources**

* Hugging Face Course:<https://huggingface.co/course>
* Transformers Docs:<https://huggingface.co/docs/transformers>
* Datasets Docs:<https://huggingface.co/docs/datasets>
* Gradio Docs:<https://www.gradio.app/>
* Example Notebooks:<https://github.com/huggingface/notebooks>
* HF YouTube:<https://www.youtube.com/c/HuggingFace>

Let me know if you'd like a slide deck, handouts, or project templates next!

Here are detailed notes on Hugging Face, drawing on the provided sources and including examples where available:

### **Hugging Face: An Overview**

Hugging Face is a **company and open-source community** that primarily focuses on **Natural Language Processing (NLP) and Artificial Intelligence (AI)**. It is widely recognised for its **Transformers library**, which offers tools and pre-trained models for a diverse range of NLP tasks.

**Key Features of Hugging Face**:

* **Libraries**: It includes many libraries, with the **Transformers, Datasets, and Tokenizers** libraries being the most widely used.
* **Model Hub**: A platform where users can **share and download pre-trained models, datasets, and other resources**.
* **Datasets Library**: A dedicated library for a **variety of datasets** specifically for NLP tasks.
* **Spaces**: A platform for **hosting and sharing machine learning demos and applications**.
* **Deployment**: Enables easy **deployment and use of models in production environments**.
* **Community**: Boasts a **strong community** of developers and AI enthusiasts who contribute to the ecosystem.

**Popular Models on Hugging Face**:

* **BERT (Bidirectional Encoder Representations from Transformers)**: A powerful open-source machine learning framework developed by Google for NLP. It excels at **understanding the context of words in sentences** by analysing relationships bidirectionally, helping computers understand ambiguous language.
* **GPT (Generative Pre-trained Transformer)**: Also known as text-to-text transformer.
* **RoBERTa (Robustly Optimized BERT Approach)**: A transformer-based language model that uses self-attention to analyse input sequences. It applies **dynamic masking** and aims to **improve the performance of the BERT model** by addressing its limitations.

### **Core Hugging Face Libraries**

#### **1. Transformers Library**

The **Transformers library** is the **core library for pre-trained models and pipelines** in Hugging Face. It is an **open-source Python library** developed by Hugging Face, known for being modular and extensible.

* **Why Use It?**:
  + **Simplicity**: Simple to use with complex NLP models.
  + **Access to Models**: Provides access to cutting-edge models.
  + **Community Support**: Backed by a large and active community.
  + **Customisation**: Supports customisation and fine-tuning.
  + **Integration**: Can be integrated with other tools.
* **Use Cases**:
  + **Text Classification**: Classifying text into categories, e.g., spam detection in emails.
  + **Named Entity Recognition**: Identifying entities like names, dates, and locations in text.
  + **Machine Translation**: Translating text between different languages (e.g., English to German).
  + **Text Generation**: Generating text using models like GPT.
  + **Question Answering**: Answering questions based on a given context.
* **Installation**:
  + **Using pip**: pip install transformers.
  + **Using Google Colab**: Add an exclamation sign: !pip install transformers.
  + **From GitHub Repository**: Directly install from the Hugging Face GitHub repository.
  + **Example (Google Colab)**:
    - Open Google Colab (collab.research.google.com).
    - Create a new notebook.
    - Type the command !pip install transformers in a cell and click "run cell".

#### **2. Datasets Library**

The **Datasets library** provides **easy access to a wide variety of datasets** for NLP and other machine learning tasks. It is a **Python library developed by Hugging Face** that simplifies working with data for training and evaluating models.

* **Why Use It?**:
  + **Efficiency**: Uses lazy loading and streaming to work with large datasets.
  + **Unified API**: Provides a unified API for processing datasets.
  + **Integration**: Works with the Transformers library and other ML frameworks for better integration and interoperability.
  + **Vast Collection**: Hugging Face provides thousands of datasets (over 350K are mentioned on their website).
  + **Customisation**: Supports custom datasets and pre-processing pipelines.
* **Use Cases**:
  + **Loading and Pre-processing Data**: Easily load and pre-process datasets for tasks like spam detection (text classification), sentiment analysis, and question answering.
  + **Machine Translation**: Datasets are provided for translation purposes.
  + **Named Entity Recognition**: Fulfils named entity recognition purposes.
  + **Custom Datasets**: Load and pre-process your own custom datasets.
* **Installation**:
  + **Using pip**: pip install datasets.
  + **Using Google Colab**: !pip install datasets.
  + **From GitHub Repository**: pip install git+https://github.com/huggingface/datasets.
  + **Example (Google Colab)**:
    - Similar to Transformers installation, type !pip install datasets in a new Google Colab notebook cell and run it.

#### **3. Tokenizers Library**

The **Tokenizers library** is a **fast, efficient, and flexible library for tokenizing text**, often used alongside the Transformers library. **Tokenization** involves splitting text into smaller units (words, subwords, characters) and converting them into numerical representations that machine learning models can process.

* **Why Use It?**:
  + **Speed**: Optimised for fast tokenization, even on large datasets.
  + **Customisation**: Supports custom tokenizers and multiple tokenization algorithms.
  + **Integration**: Integrates seamlessly with other Hugging Face libraries like Transformers.
  + **Easy API**: Provides an easy API for tokenizing, decoding, and managing vocabularies.
  + **Pre-trained Tokenizers**: Access to pre-trained tokenizers.
* **Use Cases**:
  + **Text Classification**: Tokenize textual data for spam detection.
  + **Sentiment Analysis**: Analyse spam and perform sentiment analysis.
  + **Machine Translation**: Used for machine translation.
  + **Text Generation and Question Answering**: Other use cases include text generation and question answering.
  + **Domain-specific Data**: Train and use tokenizers for domain-specific datasets.
* **Installation**:
  + **Using pip**: pip install tokenizers.
  + **Using Google Colab**: !pip install tokenizers.
  + **From GitHub Repository**: pip install git+<GitHub path>.
  + **Example (Google Colab)**:
    - Similar to other libraries, use !pip install tokenizers in a Google Colab notebook and run the cell.

### **Hugging Face Access Token (API Key)**

An **access token** (also referred to as an **API key**) is a **secure string of characters used to access Hugging Face services and resources**.

* **When You Need It**:
  + **Private or Gated Models**: To access private or gated models (e.g., Meta's LLaMA).
  + **Inference API**: When using the Hugging Face Inference API to make API calls.
  + **Uploading Content**: If you are uploading models, datasets, or Spaces to the Hugging Face Hub.
* **When You Do Not Need It**:
  + **Public Models**: When accessing publicly available models (e.g., GPT-2).
  + **Transformers Library**: When using models via the Transformers library, as many are publicly available and downloadable without authentication.
  + **Open-Source Models**: To access freely available open-source models.
* **How to Create It**:
  + Go to the official Hugging Face website (huggingface.co/join).
  + Create an account using your email address and password.
  + Complete your profile (username, name, optional social media links).
  + Verify your email address.
  + Go to your profile and click on "Access Tokens".
  + Click "Create new token", add a token name (e.g., "demo key"), and click "create token".
  + **Important**: Copy and save the token securely, as it will not be visible again after closing the window.

### **Hugging Face Use Cases and Examples**

Hugging Face supports a wide range of use cases across NLP, computer vision, and even multimodal applications.

#### **1. Downloading Data Sets**

A dataset is a collection of structured data used for training, evaluating, or testing machine learning models.

* **Method**: Use the datasets library and the load\_dataset function.
* **Example (IMDb Dataset)**:
  + Install the datasets library in Google Colab (!pip install datasets).
  + Use from datasets import load\_dataset.
  + Load the IMDb dataset: imdb\_dataset = load\_dataset("imdb").
  + Print the dataset to see its structure, which typically splits into 'train' (25k rows) and 'test' (25k rows) with features like 'text' (movie reviews) and 'label' (positive/negative sentiment). A 'unsupervised' split may also exist, often used for pre-training.

#### **2. Downloading Models**

Models can be downloaded using the Transformers library or directly from the Hugging Face Hub.

* **Method**: Use the from\_pretrained method from the Transformers library.
* **Example (BERT Model)**:
  + Install the transformers library in Google Colab (!pip install transformers).
  + Load a pre-trained BERT model: model = AutoModel.from\_pretrained("bert-base-uncased").
  + This downloads the model weights, configuration, and tokenizer.
  + When an input like "hello hugging face" is passed to a BERT base uncased model, the last hidden state output shape typically reflects (batch\_size, sequence\_length, hidden\_size), e.g., (1, 7, 768), where 1 is the batch size, 7 is the tokenized sequence length (including special tokens), and 768 is the hidden size for BERT's base architecture.

#### **3. Sentiment Analysis**

**Sentiment analysis** involves **determining the sentiment expressed in a piece of text** (e.g., positive, negative, or neutral).

* **Types**:
  + **Polarity Detection**: Classifying sentiment as **positive, negative, or neutral** (e.g., "I love this product" is positive, "The service is terrible" is negative, "The package arrived on time" is neutral).
  + **Emotion Detection**: Identifying specific emotions like **anger, joy, frustration, happiness** (e.g., "I'm thrilled about the results" indicates joy, "This is pathetic" indicates anger).
  + **Aspect-Based Sentiment Analysis**: Analysing sentiment towards a **specific product or service aspect** (e.g., "The food was great but the service was slow" – positive for food, negative for service).
  + **Intent Analysis**: Determining the user's intent (e.g., purchase intent, complaint) (e.g., "Where can I buy this product?" indicates purchase intent).
* **Method**: Use the pipeline function from the Transformers library to load a pre-trained sentiment analysis model.
* **Example**:
  + Install transformers and torch (!pip install transformers torch).
  + Load the sentiment analysis pipeline: sentiment\_analyzer = pipeline("sentiment-analysis").
  + Analyse text: results = sentiment\_analyzer(["I love playing and watching cricket", "I hate when Virat Kohli misses a century"]).
  + **Output**: A list of dictionaries, each containing a label (e.g., "POSITIVE", "NEGATIVE") and a score (confidence level between 0 and 1). A score closer to 1 indicates high confidence.

#### **4. Text Classification**

**Text classification** involves categorising text into predefined categories.

* **Difference from Sentiment Analysis**:
  + **Scope**: Sentiment analysis is **narrow and specific to sentiment** (positive, negative, neutral). Text classification labels **depend on the task** (e.g., spam/not spam, different topics like sports, technology).
  + **Use Cases**: Sentiment analysis can be used for positive product reviews. Text classification is used for classifying emails as spam or not spam, or news articles by topic.
* **Method**: Load a pre-trained text classification model using the pipeline function.
* **Example (Spam Detection)**:
  + Install transformers and torch (!pip install transformers torch).
  + Load a pre-trained spam detection model (e.g., pipeline("text-classification", model="some\_spam\_model")).
  + Perform detection on multiple texts:
    - text = ["Congratulations, you have won a 500 INR Amazon gift card! Click here to claim.", "Hi Amit, let's have a meeting tomorrow at 12 p.m."].
    - results = spam\_detector(text).
  + **Output**: Each result includes a label (e.g., "spam", "not spam" based on mapping like Negative=spam, Neutral=not spam, Positive=not spam) and a score (confidence). Low confidence scores indicate model uncertainty.

#### **5. Text Summarisation**

**Text summarisation** uses the Hugging Face Transformers library to condense long articles, documents, or research papers into shorter snippets or key points. Chatbots can also use it to provide concise responses.

* **Method**: Use AutoModelForSeq2SeqLM and AutoTokenizer to load a pre-trained summarisation model.
* **Example**:
  + Install transformers and torch (!pip install transformers torch).
  + Load model and tokenizer (e.g., model = AutoModelForSeq2SeqLM.from\_pretrained("t5-small"), tokenizer = AutoTokenizer.from\_pretrained("t5-small")).
  + Set input text, tokenize it, and use the generate method.
  + **Parameters for generation**:
    - max\_length: Maximum number of tokens in the summary (e.g., 512).
    - min\_length: Minimum number of tokens in the summary.
    - length\_penalty: Encourages longer or shorter summaries (e.g., 2 for longer).
    - num\_beams: Controls beam search width; higher values improve quality but slow inference (e.g., 4).
  + **Output**: A concise summary of the input text.

#### **6. Translation (Text-to-Text Generation)**

**Translation** involves converting text from one language to another (e.g., English to Spanish). It is part of **text-to-text generation**.

* **Text-to-Text Generation vs. Text Generation**:
  + **Text Generation**: Used for **autoregressive text generation**, where the model generates text sequentially, one token at a time (e.g., dialogue systems, text completions).
  + **Text-to-Text Generation**: Used for **sequence-to-sequence tasks** where the model takes an input sequence and generates an output sequence (e.g., summarisation, paraphrasing, translation, question answering, sentiment classification).
* **Method**: Use the Transformers library and models like t5-small.
* **Example**:
  + Install transformers and torch (!pip install transformers torch).
  + Load a pre-trained T5 model: model = T5ForConditionalGeneration.from\_pretrained("t5-small"), tokenizer = T5Tokenizer.from\_pretrained("t5-small").
  + Prepare input text with a task-specific prefix (e.g., f"translate English to Spanish: My name is Amit Diwan and I love cricket.").
  + Tokenize the input, generate translated text using model.generate(), and decode the output tokens.
  + **Output**: The translated text (e.g., "Mi nombre es Amit Diwan y amo el cricket").

#### **7. Question Answering**

**Question answering** uses the Transformers library to find answers to questions within a given context.

* **Method**: Load a pre-trained Question Answering (QA) model and tokenizer, then provide a context and a question.
* **Example**:
  + Install transformers and torch (!pip install transformers torch).
  + Load a pre-trained QA model and tokenizer (e.g., model = AutoModelForQuestionAnswering.from\_pretrained("distilbert-base-uncased-distilled-squad"), tokenizer = AutoTokenizer.from\_pretrained("distilbert-base-uncased-distilled-squad")).
  + Set a context (paragraph of text) and a question (e.g., context = "Amit Diwan is based in Delhi and works as a software engineer.", question = "Where Amit Diwan is based?").
  + Tokenize the input, get the model's prediction (start/end scores), and decode the answer tokens.
  + **Output**: The extracted answer from the context (e.g., "Delhi").

#### **8. Text-to-Image**

**Text-to-image synthesis** involves generating images from textual descriptions.

* **Libraries/Models**: Uses the **Hugging Face Diffusers library** and **Stable Diffusion model**.
  + **Diffusers Library**: An **open-source Python library focused on diffusion models** for generating images, audio, and other data types.
  + **Stable Diffusion**: A **latent diffusion model** designed for high-quality image generation from text prompts, and a very popular generative model.
* **Method**: Load the Stable Diffusion pipeline from the Diffusers library, then pass a text prompt.
* **Example**:
  + Install diffusers, transformers, torch (!pip install diffusers transformers torch).
  + Load the Stable Diffusion pipeline: pipeline = StableDiffusionPipeline.from\_pretrained("stabilityai/stable-diffusion-v1-5").
  + Generate an image from a prompt: prompt = "Flying cars soar over a futuristic cityscape at sunset".
  + The generated image can be saved (e.g., image.save("generated\_image.png")) and downloaded from Google Colab.

#### **9. Text-to-Video**

**Text-to-video synthesis** involves generating a video (sequence of frames) from textual descriptions. It's a complex task combining NLP and generative/diffusion models.

* **Video Generation Frameworks**: Mentions Runway ML, Pika Labs, and DeepMind's Perceiver IO. Libraries like PyTorch or TensorFlow are used to build video generation pipelines.
* **Method**: Uses the Diffusers library and can leverage models like Stable Diffusion to generate individual frames, which are then stitched into a video using libraries like OpenCV.
* **Example**:
  + Install diffusers, transformers, torch, opencv-python, numpy (!pip install diffusers transformers torch opencv-python numpy).
  + Load a text-to-image model from the Diffusers library (e.g., Stable Diffusion).
  + Set a prompt (e.g., "A futuristic cityscape at night with flying cars") and generate multiple individual frames (e.g., 10 frames) using a loop.
  + Append generated frames to a list.
  + Use OpenCV (cv2) and NumPy (np) to stitch these frames into a video (e.g., an .mp4 file).
  + **Output**: The process generates individual PNG frames (e.g., frame\_0.png to frame\_9.png) and an output video file (e.g., output\_video.mp4) that can be downloaded from Google Colab.

Project setup process

1. Pipline
2. Model/tokenizer
3. Pytorch/Tf
4. Save/load
5. modelHub
6. FineTune

Install first --> using pytorch,tf

Pip install transformers

From transformer import pipeline

**PowerPoint Presentation: Mastering Large Language Models (LLMs)**

### **Slide 1: Title Slide**

* **Title:** Mastering Large Language Models (LLMs)
* **Subtitle:** From Basics to Deployment
* **Presented by:** [Your Name/Organization]

### **Slide 2: Course Overview**

* 8 Modules covering NLP, Transformers, LLMs, and Deployment
* Hands-on with tools like Hugging Face, LangChain, OpenAI API
* Real-world projects: summarization, chatbots, RAG-based apps

### **Slide 3: Prerequisites**

* Python Programming
* Basics of Machine Learning
* Intro to Neural Networks
* Optional: NLP Basics

### **Slide 4: Module 1 - Introduction to NLP**

* What is NLP?
* Key tasks: Tokenization, POS tagging, NER
* Rule-based vs Machine Learning approaches

### **Slide 5: Module 2 - Neural Networks for Language**

* Word Embeddings: Word2Vec, GloVe
* RNNs, LSTM, GRU
* Encoder-Decoder architectures

### **Slide 6: Module 3 - Transformers and Attention**

* Problem with RNNs
* Self-attention mechanism
* Transformer architecture
* Positional encoding, Multi-head attention

### **Slide 7: Module 4 - Understanding LLMs**

* What are Language Models?
* BERT, GPT, T5, PaLM, LLaMA
* Pretraining vs Fine-tuning
* Inference and Use Cases

### **Slide 8: Module 5 - Hands-on with LLMs**

* Hugging Face Transformers
* GPT-2/3 for text generation
* Prompt Engineering: Zero-shot, Few-shot
* Fine-tuning small models

### **Slide 9: Module 6 - Retrieval-Augmented Generation (RAG)**

* Concept of RAG
* FAISS/Chroma for vector stores
* LangChain + LLM for QA bots
* Build your custom LLM-based chatbot

### **Slide 10: Module 7 - Ethics, Safety, and Limitations**

* Bias and fairness in LLMs
* Hallucinations and content moderation
* Guardrails and responsible use

### **Slide 11: Module 8 - Production Deployment**

* LLMs via API (OpenAI, Cohere)
* Local hosting & optimization
* LangChain, Streamlit for app interfaces
* LLMOps: Monitoring and Cost Management

### **Slide 12: Projects & Tools**

* Projects:
  + Blog Generator
  + Summarizer
  + RAG Chatbot
* Tools:
  + Hugging Face, LangChain
  + OpenAI, Streamlit, Gradio

### **Slide 13: Recommended Free Courses**

* Hugging Face: <https://huggingface.co/learn/nlp-course>
* DeepLearning.AI: <https://www.deeplearning.ai/short-courses/chatgpt-prompt-engineering-for-developers/>
* Coursera NLP Specialization
* Fast.ai, Karpathy’s Zero to Hero

### **Slide 14: Final Slide**

* **Thank You!**
* Questions?
* Contact: [Your Contact Info]

### **Bonus: Real-World Projects with Hugging Face Pipelines**

#### 1. **Text Generation - Blog Idea Expander**

from transformers import pipeline  
  
pipe = pipeline('text-generation', model='gpt2')  
prompt = "The role of AI in personalized education"  
print(pipe(prompt, max\_length=100)[0]['generated\_text'])

#### 2. **Text Classification - News Sentiment Analyzer**

from transformers import pipeline  
classifier = pipeline("text-classification")  
print(classifier("Stock market is expected to rise sharply next week."))

#### 3. **Summarization - Article Summarizer**

from transformers import pipeline  
summarizer = pipeline("summarization")  
text = """(insert long news article text here)"""  
print(summarizer(text, max\_length=100, min\_length=30, do\_sample=False))

#### 4. **Translation - Language Localizer**

from transformers import pipeline  
translator = pipeline("translation\_en\_to\_fr")  
print(translator("Machine learning will change the future of healthcare."))

#### 5. **Zero-shot Classification - Resume Skill Matching**

from transformers import pipeline  
classifier = pipeline("zero-shot-classification")  
sequence = "He has 5 years of experience in Python and DevOps."  
candidate\_labels = ["Software Development", "DevOps", "Data Science"]  
print(classifier(sequence, candidate\_labels))

#### 6. **Feature Extraction - Semantic Search**

from transformers import pipeline  
extractor = pipeline("feature-extraction")  
features = extractor("OpenAI has transformed the AI ecosystem.")  
print(features[0][0][:10]) # Show first 10 values of first token

#### 7. **Image-to-Text - Caption a Product Photo**

from transformers import pipeline  
captioner = pipeline("image-to-text", model="nlpconnect/vit-gpt2-image-captioning")  
from PIL import Image  
img = Image.open("product.jpg")  
print(captioner(img))

#### 8. **Image Classification - Quality Control for Manufacturing**

from transformers import pipeline  
classifier = pipeline("image-classification")  
img = Image.open("sample.jpg")  
print(classifier(img))

#### 9. **Object Detection - Detect Tools on Factory Floor**

from transformers import pipeline  
from PIL import Image  
img = Image.open("factory.jpg")  
detector = pipeline("object-detection")  
print(detector(img))

#### 10. **Automatic Speech Recognition - Meeting Transcriber**

from transformers import pipeline  
asr = pipeline("automatic-speech-recognition")  
print(asr("meeting\_audio.wav"))

#### 11. **Audio Classification - Detect Alarms in Audio**

from transformers import pipeline  
classifier = pipeline("audio-classification")  
print(classifier("alarm.wav"))

#### 12. **Text-to-Speech - Voice-Enable a Chatbot**

from TTS.api import TTS  
tts = TTS(model\_name="tts\_models/en/ljspeech/tacotron2-DDC", progress\_bar=False, gpu=False)  
tts.tts\_to\_file(text="Hello, your order has been placed successfully!", file\_path="output.wav")

#### 13. **Image+Text to Text - Visual Q&A System**

from transformers import pipeline  
pipe = pipeline("image-to-text", model="Salesforce/blip-image-captioning-base")  
img = Image.open("context\_image.jpg")  
print(pipe(img))