API Driven Cloud Native Solution Notes

Pre-trained Models, Fine-tuned Models, and Transfer Learning
Lesson 8

Understanding AI, ML, Deep Learning, and Generative AI

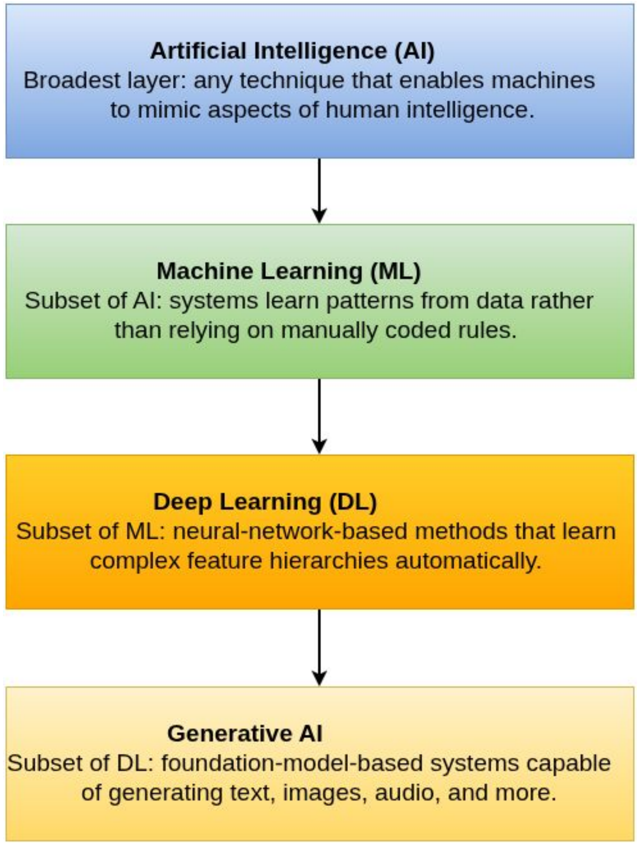
Artificial Intelligence (AI) is the “umbrella” term for all computer methods that simulate human intelligence, ranging from simple programmed rules to advanced learning systems.

Machine Learning (ML) sits just below AI. Instead of being explicitly programmed for each task, ML systems learn to spot patterns by analyzing and practicing on lots of examples. The system’s rules emerge automatically by processing data rather than being written by a programmer.

Deep Learning (DL) is a type of ML. DL uses layered artificial neural networks to uncover complex relationships in data, such as recognizing images or speech. Thanks to huge datasets and powerful computers, DL models can figure out the best features on their own, without human intervention.

Generative AI is a specialised kind of Deep Learning. These models, such as large language models, can generate entirely new text, images, or other content by drawing on what they learned during training. Generative AI relies on state-of-the-art architectures (like transformers) and vast amounts of data to deliver fluent, creative results.

Hierarchy of AI Disciplines


Artificial Intelligence (AI)
Broadest layer: any technique that enables machines to mimic aspects of human intelligence.
Machine Learning (ML)
Subset of AI: systems learn patterns from data rather than relying on manually coded rules.
Deep Learning (DL)
Subset of ML: neural-network-based methods that learn complex feature hierarchies automatically.
Generative AI
Subset of DL: foundation-model-based systems capable of generating text, images, audio, and more.

AI Use Cases

AI applications can be categorised into several broad areas, each reflecting how intelligent systems are being incorporated into modern digital environments.

One major area focuses on improving **customer experience**. In this space, AI is used to power contact centres, conversational assistants, personalised recommendations, identity verification, and automated moderation of user-generated content. These systems rely on language, vision, and pattern-recognition capabilities to provide faster and more accurate responses to user interaction.

Another area concentrates on **augmenting human work**. AI systems are used to generate text, create images, assist with daily tasks, produce forecasts, generate software code, and prepare analytical reports. These applications reduce manual effort and support employees in performing tasks that require synthesis of information or repetitive decision-making.

A third area involves improving **business operations**. AI is used in document understanding, fraud detection, predictive maintenance, language translation, visual inspection, and the optimisation of routine processes. These applications are often built by integrating pretrained models into business workflows, enabling organisations to automate tasks that previously required significant human effort.

A final category involves the creation of **entirely new products** and services that rely on capabilities made possible by modern AI techniques. These offerings often depend on generative or adaptive models that enable new forms of interaction, content creation, or decision support that traditional software could not previously deliver.

The Three Generations of Machine Learning

This slide explains the evolution of ML through three distinct generations.

Generation 1 is described as the era of traditional machine learning. Labeled datasets must be manually created, features must be engineered, and models must be trained from scratch. Performance evaluation is performed manually, and significant effort is required at each stage.

Generation 2 introduces deep learning, driven by the availability of large datasets and enhanced compute resources. Models are re-trained using neural networks, and while accuracy improves dramatically, the resource burden remains high. Neural architectures such as CNNs and RNNs define this era.

Generation 3 is marked by transfer learning and transformers. Large pre-trained models (GPT, BERT, etc.) are made available to the public. These models are trained once at enormous cost and then reused by millions of developers. In Generation 3, model training is no longer required for most practical applications. Instead, **APIs are used to perform inference on pre-trained or fine-tuned models**, often through a pay-per-use model.

Pre-trained Models

A **pre-trained model** is a machine learning model that has already been trained once on a very large collection of data before it is used for any specific task.

In the context of language, a pre-trained language model is trained on large amounts of text. During this training, the model is repeatedly exposed to sentences and is asked to perform tasks such as predicting missing words or the next token. Over time, the model learns:

- how words usually appear together,
- how sentence structure works,
- how meaning depends on context.

By the end of this training, the model does not specialise in any one task such as sentiment analysis or translation. Instead, it gains a **general understanding of language**. These pre-trained models are then **published and shared** on platforms like the **Hugging Face Hub,** an open source platform hosting models, datasets, and code repositories.

Once a model is on the Hub, it:

- belongs to the collection of community-available models,
- can be downloaded as a model file,
- can be loaded through libraries such as transformers,
- or can be accessed via APIs provided by Hugging Face.

At this stage, the model is called a **pre-trained model** or **base model**. It is powerful, but still general-purpose.

Examples include models based on transformer architectures such as **BERT** and **GPT**, and many others hosted on Hugging Face.

Fine-tuned Models: Specialising the Base Model

A **fine-tuned model** is created when a pre-trained model is taken and then trained further on a **smaller, task-specific dataset**.

The key idea is:

- The pre-trained model already knows “how language works” in general.
- Fine-tuning teaches it “how this specific task works” in particular.

Sentiment analysis with IMDB

Example: “bert-base-uncased-sentiment”, fine-tuned for sentiment analysis, trained on the **IMDB movie reviews** dataset.

Stages of Fine Tuning the Base Model

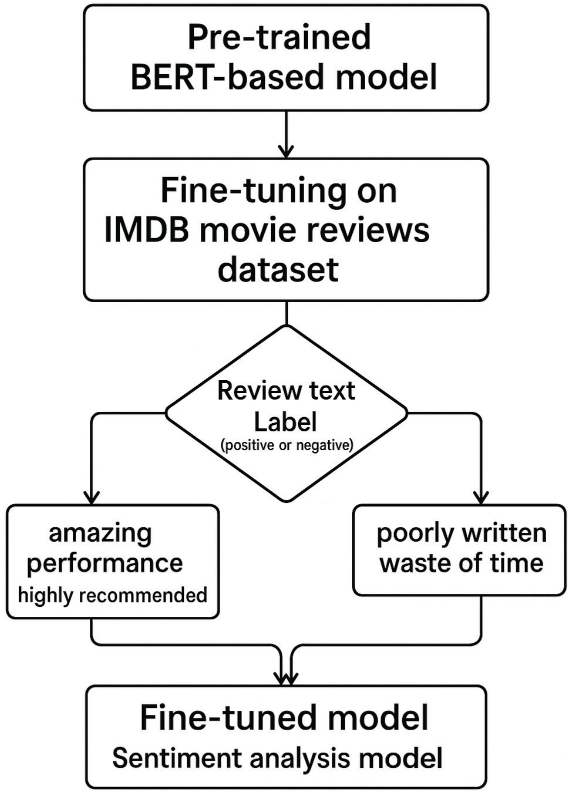
1. A general BERT-based model is first **pre-trained** on a large corpus of English text. At this stage, it learns grammar, vocabulary, and general patterns of language usage.
2. This pre-trained model is then **fine-tuned** on the IMDB movie reviews dataset, where each review is labelled as positive or negative.
3. During fine-tuning, the model is shown many pairs of the form:
   - review text
   - label (positive or negative)
4. The model gradually learns **what patterns in the text correspond to positive sentiment** (for example, phrases like “amazing performance”, “highly recommended”) and **what patterns correspond to negative sentiment** (for example, “poorly written”, “waste of time”).

At the End of Fine Tuning

By the end of fine-tuning:

- The model is no longer just a general English model.
- It becomes a **sentiment analysis model** that is particularly good at judging whether a review is positive or negative.

The model performs well on product reviews, social media posts, and similar opinionated text. It uses its general language understanding plus the specific patterns learned from IMDB. This is a fine-tuned model.


Pre-trained BERT-based model
Fine-tuning on IMDB movie reviews dataset
Review text Label (positive or negative)
amazing performance highly recommended
poorly written waste of time
Fine-tuned model Sentiment analysis model

Fine-tuning Improves Performance

The performance of a fine-tuned model is generally better than a model trained from scratch on a small dataset. The reasons are:

- A model trained from scratch on a small dataset (for example, only IMDB reviews) has to learn **both** language structure and sentiment patterns from the same limited data. This is difficult and often leads to weaker performance.
- A fine-tuned model starts in a much stronger position. It already knows language structure from its large-scale pre-training. The IMDB dataset then only needs to teach it **how sentiment is expressed**.

This combination of:

- general language capability (from pre-training), and
- task-specific understanding (from fine-tuning)

allows fine-tuned models to perform **better** and to learn **faster** than models built from scratch on the same small dataset.

Transfer Learning

Transfer learning refers to the reuse of knowledge gained from one learning process in another. In the context of transformer models, the knowledge gained during pre-training—understanding grammar, structure, vocabulary, and common language patterns—is transferred directly into the fine-tuning phase. Fine-tuning simply adjusts the model so this general knowledge becomes useful for the specialised task.

In this context:

- The **pre-training phase** teaches the model general language knowledge.
- The **fine-tuning phase** transfers that general knowledge to a new task, such as sentiment analysis or question answering.

The “transfer” is happening from:

- the general task of understanding text (pre-training),
- to the specific task (sentiment, QA, translation, etc.).

Applications of Transfer Learning

A single pre-trained model can be adapted for many diverse tasks:

- a version fine-tuned for sentiment analysis,
- another version fine-tuned for question answering,
- another for summarisation,
- and so on.

Each variant inherits the general language capability from the original pre-trained model and then learns its specific role through targeted fine-tuning.

Example: Question Answering (distilled SQuAD model)

A model such as **`Distilbert-base-uncased-distilled-squad`** is fine-tuned for **question answering**.

The process can be understood as:

- A base DistilBERT model is pre-trained as a general language model.
- It is then fine-tuned on a question–answer dataset (SQuAD-style), where each training example contains:
  - a passage of text,
  - a question,
  - and the correct answer span within the passage.
- When a new passage and question are provided, the model uses:
  - its general understanding of language,
  - plus its specialised training from SQuAD-style fine-tuning,

to select a likely answer span. Again, the knowledge learned in pre-training has been **transferred** to a more specific question-answering task.

Pre-Trained vs Fine-Tuned Models

The contrast between pre-trained and fine-tuned models is structured around three dimensions:

Training Data:
Pre-trained models are trained on very large and diverse corpora. Their learning captures general patterns. Fine-tuned models are trained on narrower datasets related to specific tasks or domains. This specialization enables improved performance in narrow use cases.

A pre-trained model serves as a general base model. A fine-tuned model is tailored for a particular application such as sentiment analysis or domain-specific classification. The distinction is important when selecting a model from platforms such as **Hugging Face**.

Efficiency:
Pre-training is computationally expensive. Fine-tuning is far more efficient because the base knowledge is already present. Fine-tuning only adjusts model parameters relevant to the specialised task.

Pre-Trained vs Fine-Tuned

Fine-tuning generally improves performance because a pre-trained model already carries broad language knowledge learned from large text collections. When this general understanding is adjusted with domain-specific examples, the model develops a sharper sense of the patterns relevant to that task. A model trained from scratch on a small dataset does not have this advantage, so it typically performs worse.

Transfer learning enables this improvement. A pre-trained model that has learned grammar, vocabulary, and common sentence structures can be adapted to a specialised task with relatively little data. A practical example is sentiment analysis using an IMDB-based fine-tuned model. The general model understands English, while the fine-tuned version learns how positive and negative opinions are typically expressed in movie reviews. The same idea applies to question answering, where a model such as a distilled SQuAD version uses its general language knowledge and refines it with question-answer pairs drawn from that dataset.

Interpretability also becomes easier when fine-tuning is used. Because the model has been adjusted to a specific domain, its mistakes tend to reflect patterns within that domain. This makes it more straightforward to understand why a sentiment model misclassified a review or why a question-answering system selected an incorrect span from a passage.

Where These Models Live and How They Are Used

Some facts about the general purpose models:

- The **Hugging Face Hub** hosts models, datasets, and code.
- Models have **model cards**, describing:
  - what they were trained on,
  - what they should be used for,
  - their limitations,
  - and their evaluation scores.

A model such as “bert-base-uncased-sentiment”:

- lives on the Hugging Face Hub,
- can be searched and viewed through its model card,
- can be used through code such as:

```
from transformers import pipeline

classifier = pipeline('sentiment-analysis', model='bert-base-uncased-sentiment')

result = classifier("I love this movie!")
```

Here, the **`pipeline`**:

- downloads or loads the fine-tuned model,
- runs the input text through the model,
- and returns the sentiment label and a confidence score.

In this process:

- The **pre-trained part** of the model handles understanding the sentence structure and language.
- The **fine-tuned part** of the model decides whether the sentiment is positive or negative.

Interpretability in Fine-tuned Models

Interpretability becomes more manageable when a model is fine-tuned on a specific domain.

Consider the sentiment analysis case again:

- The model has been fine-tuned specifically on movie reviews from IMDB.
- When it misclassifies a review, the error can often be traced back to patterns in that domain:
  - perhaps sarcasm in a movie review,
  - or a very mixed opinion with both praise and criticism.

Because the model's behaviour has been shaped directly by that dataset, its decisions can be analysed more easily in that context. The model card, which lists evaluation metrics and limitations (for example, poor performance on sarcasm or slang), further helps to interpret and trust or question the model's outputs.

Summary

1. **A pre-trained model** is trained once, at large scale, to learn general language behaviour.
2. **A fine-tuned model** begins from this strong base and specialises in a specific task using a smaller labelled dataset.
3. **Transfer learning** is what allows the general knowledge learned in pre-training to be reused efficiently during fine-tuning.
4. **Performance** improves because fine-tuning builds on a strong foundation rather than starting from scratch.
5. **Interpretability** becomes easier because the model's behaviour is tied to the patterns present in the task-specific dataset.

Questions

Choosing between training a model vs using a pre-trained API

Training a model is suitable when the task requires domain specific behaviour that cannot be met by existing models or when proprietary training data must be used to create a customised system. This approach requires labelled data, training infrastructure, and model optimisation effort.

A pre-trained API is suitable when the task matches a common pattern such as sentiment analysis, classification, translation, summarisation, or object detection. The model is already trained on large datasets and can be integrated immediately. This avoids the effort of dataset preparation, training cycles, monitoring, and infrastructure management. Pre-trained APIs also ensure consistent performance and faster development.

Questions

Q. Explaining business use cases for AI-driven systems

AI driven systems are used to enhance customer experience by providing automated agents, personalised recommendations, identity verification, and content moderation. They assist employees by generating content, producing reports, and supporting decision making. They improve business operations through document understanding, fraud detection, predictive maintenance, language translation, visual inspection, and process optimisation. AI systems also support the creation of new digital products that rely on capabilities such as text generation, summarisation, and intelligent search.

Q. Describing efficiency advantages of transformer-based approaches

Transformer models process input by attending to all parts of a sequence simultaneously. This allows long range relationships to be captured without sequential dependency. As a result, they achieve higher accuracy in tasks such as translation, question answering, text generation, and summarisation. Because the self attention mechanism scales effectively, transformers can be pre-trained on very large corpora and reused across many tasks. This reduces the need for repeated training and provides faster adaptation through fine-tuning.

Q. Reasoning about why APIs exist in modern ML pipelines

APIs provide access to pre-built intelligence without requiring model training. Modern ML pipelines rely on APIs because they encapsulate complex models behind stable interfaces. This allows applications to perform tasks such as classification, translation, vision processing, or summarisation by sending requests to an endpoint. It reduces development time, eliminates the need for specialised infrastructure, and ensures consistent performance. APIs also enable scalable usage because the service provider manages hardware, updates, monitoring, and optimisation of the underlying model.

Q. Discussing transfer learning or model lifecycles

Transfer learning allows knowledge learned during large scale pre-training to be reused for new tasks. A pre-trained model captures general language or visual patterns. Fine-tuning adjusts this model using a smaller, task specific dataset, producing higher accuracy than training from scratch.

The model lifecycle typically includes pre-training, fine-tuning, deployment, inference, monitoring, and periodic updates. Pre-training is resource intensive and performed once. Fine-tuning adapts the model to specific tasks. Deployment exposes the model through APIs or pipelines. Monitoring ensures reliability, and updates refine the model as data or requirements evolve.

Q.2. You are working as an AI Solutions Architect for a legal firm that wants to deploy an AI chatbot to assist employees in answering questions from internal legal contracts, compliance documents, and HR policies. Describe two approaches you could use to implement this solution using APIs. For each approach, mention the APIs or services you would consider and explain any assumptions involved. Code is not required. [6 marks]

Two Approaches for Implementing the AI Chatbot Using APIs

A legal firm requires a chatbot capable of answering employee questions based on internal legal contracts, compliance documents, and HR policies. This can be achieved using API-based AI services without training models from scratch. Two suitable approaches are described below.

Approach 1: Using Cloud-based Cognitive Services APIs

Cognitive services provide ready made language capabilities through REST APIs. These services can process text, extract intent, identify entities, summarise content, and answer queries. A legal chatbot can be implemented by integrating the firm’s documents with language understanding, search, and question answering APIs.

Language Services can analyse queries and identify intent and relevant terms. Document Intelligence or text analysis APIs can extract key clauses, obligations, and policy statements from uploaded contracts or policies. A question answering API can then match the employee’s question with relevant passages and produce a concise answer. Speech Services may also be used if voice input is required.

Assumption:
The firm is willing to store its legal documents in the secured environment of the cloud provider’s cognitive services, and these services support the languages, formats, and compliance requirements of the organisation.

Approach 2: Using Hugging Face Transformer Models Through APIs

Hugging Face provides pre-trained and fine-tuned transformer models accessible through API endpoints. These models include tasks such as question answering, summarisation, text classification, and translation. A legal chatbot can be created by selecting a transformer model fine-tuned for question answering and using it through the Hugging Face Inference API.

The internal documents can be converted into passages. When an employee asks a question, the chatbot sends the question and the relevant text segment to the API. The model identifies the answer span just as in standard question answering tasks. Summarisation models can also be used to simplify long policy text, and text classification models can help route the question to the correct document category.

Assumption:
The firm stores its legal and compliance documents in a secure internal system and uses a retrieval layer to send only the required text snippets to the Hugging Face API, ensuring no full documents leave the firm’s environment.

Conclusion

Both approaches avoid training models from scratch and rely on pre-trained, production-ready APIs. The first approach uses cloud cognitive services with built-in document analysis and language models, while the second uses transformer-based models hosted on Hugging Face. Both approaches provide scalable and accurate solutions for answering queries from legal and policy documents.

Q.3. A user sends the following prompt to a chatbot – "It’s not a bad suggestion, though I’m not exactly thrilled about it."

a. Write the base Python code to analyze the sentiment of this message. Use any sentiment analysis API or model of your choice. [2 marks] b. Based on the sentiment result, what would be an appropriate chatbot response? Justify your answer. [2 marks]

Python code for sentiment analysis

A simple sentiment analysis can be performed using a pre-trained transformer model available through Hugging Face:

```python
from transformers import pipeline

classifier = pipeline("sentiment-analysis")

text = "It’s not a bad suggestion, though I’m not exactly thrilled about it."

result = classifier(text)

print(result)
```

This code loads a standard sentiment analysis model and returns a label such as positive or negative with a confidence score.

Appropriate chatbot response with justification

The message expresses mixed sentiment. The phrase “not a bad suggestion” indicates mild positivity, while “not exactly thrilled” weakens the enthusiasm. This usually results in a sentiment classification that is slightly positive or neutral.

An appropriate chatbot response would acknowledge the user’s hesitation and offer support, for example:

Response:
"I understand that the suggestion seems acceptable but not exciting. Let me know what specific concerns you have, and I can help refine it."

Justification:
The response reflects the mild, reserved sentiment detected by the model. It recognises the user’s partial agreement and addresses the lack of enthusiasm by inviting clarification.

Cognitive Services

Cognitive services refer to cloud-hosted artificial intelligence capabilities that allow applications to perform tasks that resemble human cognitive functions. These tasks include:

- understanding language,
- recognising objects in images,
- extracting meaning from speech,
- and making recommendations.

The central idea behind cognitive services is that powerful AI models can be used without requiring any expertise in training or maintaining machine learning systems. Instead, the intelligence is accessed through simple API calls.

These services are provided by major cloud platforms. They offer pre-built models that have already been trained using large datasets. Because the heavy training work has been completed in advance, developers can incorporate sophisticated AI behaviour into their applications immediately. The process usually involves sending data to a cloud endpoint and receiving the processed output in a structured format.

Nature of Cognitive Services

Cognitive services are designed to replicate abilities such as understanding, perception, and decision-making. They operate through cloud infrastructure, which allows them:

- to handle large volumes of requests,
- scale automatically,
- and deliver consistent performance.

This makes them suitable for production applications where reliability and availability are important.

One significant advantage of cognitive services is the reduction in development effort. Since the models are already trained, the user does not need to assemble datasets or configure training pipelines. The services expose their functionality through standard interfaces such as HTTP REST APIs or through SDKs for common programming languages. This enables integration into web applications, mobile applications, and enterprise systems.

Azure Cognitive Services

Azure provides a suite of AI services that can be accessed directly through the cloud. These services include pretrained models for language understanding, speech processing, vision tasks, and decision-making. The entire collection is built and maintained by the Azure AI and Research team. Each service can be accessed through REST endpoints, allowing developers to send input such as text, audio, or images and receive structured AI-generated output.

Common examples include text analysis, speech to text, text to speech, translation, object detection, and content moderation. By using these services, developers extend their applications with complex AI behaviour without performing any training or constructing machine learning infrastructure.

Amazon AI Services

AWS offers a broad collection of fully managed artificial intelligence services that can be invoked directly from the cloud. These services cover language, speech, vision, analytics, and personalisation tasks, all backed by pretrained models maintained by Amazon's AI and Machine Learning teams. Each service exposes a simple API or REST endpoint, allowing developers to submit text, images, audio, or video and receive structured, machine-generated results.

Typical examples include:

- natural language processing through **Amazon Comprehend**,
- speech recognition and synthesis through **Amazon Transcribe** and Amazon Polly,
- image and video analysis through **Amazon Rekognition**,
- translation via **Amazon Translate**,
- and content moderation using built-in **Rekognition and Comprehend** features.

By adopting these services, developers add advanced AI capabilities to their applications without building, training, or hosting any machine learning models or infrastructure themselves.

Google Cloud Vertex AI (Gemini Models)

Google Cloud provides a suite of advanced AI services powered by its Gemini family of foundation models. These services can be accessed directly through **Vertex AI**, allowing developers to use pre-trained large language models, multimodal models, and specialised vision, speech, and translation systems. The entire platform is built and maintained by **Google DeepMind** and Google Cloud's AI engineering teams. Each model can be invoked through REST or gRPC endpoints, where developers send inputs such as text, images, audio, or video and receive structured model-generated output.

Common use cases include text analysis, summarisation, translation, code generation, speech processing, image understanding, and multimodal reasoning. By relying on these models through Vertex AI's managed interface, developers can enhance their applications with sophisticated AI capabilities without training their own models or setting up any machine learning infrastructure.

Categories of Cognitive Services

Cloud-based cognitive services are often grouped by the type of cognitive ability they provide.

Language services process and analyse text. They identify entities, classify text, detect sentiment, summarise content, and extract intent from written material.

Speech services convert speech to text, convert text to speech, translate speech, and recognise speakers.

Vision services analyse images and video. They recognise objects, identify faces, classify scenes, read text present in images, and process visual data for downstream tasks.

Decision services assist with making informed choices by applying trained models for anomaly detection, recommendation, and classification in structured data.

OpenAI-based services provide access to powerful transformer models that can generate text, analyse content, summarise information, or answer questions. These models are offered through REST APIs and require no local training.

These categories offer a wide range of ready-made functions that cover common requirements in modern AI applications.

Benefits of Using Cognitive Services

Cognitive services offer several important advantages.

Complex AI behaviour can be added with minimal coding effort because the services use predefined and pre-trained algorithms.

Integration is straightforward since the services are delivered through REST interfaces, which can be called from almost any application environment.

The services are designed to be usable by individuals with varying levels of AI knowledge, meaning both experienced developers and new practitioners can incorporate intelligence into applications without facing complex training processes.

These services are hosted in the cloud, which means the underlying hardware, scaling, and maintenance are handled automatically. This is particularly helpful when applications must serve many users concurrently, process large volumes of text or images, or operate continuously.

Cognitive Services in the Broader AI Workflow

The availability of cloud-hosted cognitive services has changed the way AI applications are built. Instead of constructing models from scratch, developers rely on pretrained models and APIs that encapsulate powerful deep learning algorithms. This approach aligns with the larger shift toward transformer-based systems and foundation models, where extensive pre-training is performed once and the resulting capability is shared through public interfaces.

By using language, speech, vision, and decision services, a wide variety of applications can be created. These may include virtual assistants, content analysis systems, image classification tools, customer support agents, translation services, or automated document processing pipelines.

Hugging Face

Hugging Face is an open source platform that serves as a central hub for transformer models, datasets, and code repositories. It plays a major role in modern natural language processing because it brings together pre-trained models and fine-tuned models created by the community. These models can be downloaded, reused, and integrated into applications with minimal effort. The platform supports text, image, and audio tasks, making it useful for a wide variety of AI applications.

The Hub operates similarly to a version-controlled repository. Every model is stored with its files and documentation, and developers can track changes or contribute improvements. Alongside models, the platform hosts datasets and code, which creates an ecosystem where tasks, models, and data can be connected easily.

Tasks in Hugging Face

A task represents a specific problem an AI model is designed to solve. Hugging Face organises models by tasks so that developers can quickly locate models that suit their needs. Tasks describe the purpose of the model and the structure of the input and output.

For example,

- **Text classification** involves assigning labels to text segments. Sentiment analysis is one instance, where text is labelled as positive or negative.
- **Named Entity Recognition** identifies entities such as names, locations, and organisations within text.
- **Text generation** creates new text based on a prompt.
- **Machine translation** converts text from one language to another.
- **Question answering** identifies an answer within a passage of text.
- **Summarisation** produces a shorter version of a longer input while keeping the essential meaning.

These tasks illustrate how Hugging Face categorises practical applications.

There are also tasks outside language processing.

- **Image classification** assigns categories to images.
- **Object detection** identifies objects within an image.
- **Speech recognition** converts spoken audio into text.

All of these tasks are represented on the Hub and supported by pre-trained and fine-tuned models.

Models in Hugging Face

A model on Hugging Face is the **machine learning component** used to perform a task. Models may be general-purpose or specifically adapted to particular problems.

A pre-trained model is trained once on a large amount of data to learn broad language patterns or visual features. Such a model is suitable as a starting point for many tasks because it already understands how language or images behave in general.

A fine-tuned model is created when a pre-trained model is adapted further using a smaller, task-specific dataset. This produces better performance on that particular task than a model trained from scratch on a limited dataset.

Transformer-based models are widely used on Hugging Face. Examples include:

- **BERT**, which is effective for tasks involving understanding text, and GPT, which performs text generation.
- **T5** is a text-to-text model that can be adapted for translation, summarisation, and classification.
- **DistilBERT** is a smaller and faster version of BERT.
- Other variants such as **RoBERTa**, **XLNet**, and **ALBERT** refine transformer architecture in different ways.

The Hub stores many pre-trained and fine-tuned versions of these models.

Datasets in Hugging Face

Datasets provide the material used to train, fine-tune, or evaluate models. Hugging Face includes a wide range of datasets that support language, vision, and audio tasks. These datasets can be loaded directly through the Hugging Face library.

Examples include:

- SQuAD for question answering,
- GLUE for evaluating language understanding,
- IMDB for sentiment analysis,
- COCO for image classification and object detection,
- and LibriSpeech for speech recognition.

Many of these datasets are used to showcase the capabilities of pre-trained and fine-tuned models. Custom datasets can also be uploaded to the Hub, which allows specialised tasks to be supported by shared datasets.

The availability of datasets in one place simplifies experimentation, model building, and evaluation.

How Tasks, Models, and Datasets Interact

There are mainly three components which are as follows:

- Tasks define the type of problem to be solved.
- Models perform these tasks.
- Datasets provide the training or evaluation examples needed to teach the model.

These three components form a cycle. A developer selects a task they want to solve, chooses an appropriate model from the Hub, and uses a dataset to train, fine-tune, or test the model. This structure makes it easy to build practical AI workflows using reusable components.

Example Workflow

- An workflow begins with selecting a task such as sentiment analysis.
- A suitable model is then chosen from the Hugging Face Hub, for example a BERT-based model that has already been fine-tuned for this purpose.
- A dataset such as IMDB may be used to fine-tune or evaluate the model.
- Finally, the model is used to perform inference on new text, which produces a predicted sentiment.

This sequence provides a clear path from identifying the problem to applying the model.

Model Cards

Each model on Hugging Face includes a model card. A model card is a detailed file that documents the model. It explains what the model is designed to do, how it was trained, the data used during fine-tuning, performance metrics, and known limitations.

For instance, a sentiment model fine-tuned on the IMDB dataset may have a model card that describes its purpose, its accuracy, its F1 score, and its limitations. These limitations may include challenges with languages not included in training or difficulties with informal or sarcastic text.

Model cards help users understand where a model performs well and where it might struggle, which promotes responsible and informed usage.

Basics of Natural Language Processing

Natural Language Processing is the field that enables machines to interpret, analyse, and generate human language. It draws from computer science, linguistics, and artificial intelligence. Human communication takes place through words, sentences, and meanings. For a machine to work with this form of communication, it must process language in a form it can understand.

People interact with computers through both text and speech. A user may type a query such as a location, a fact, or a keyword into a search box. A user may also speak to a voice assistant and ask for music or information. In both cases, the system must interpret the input correctly. The machine must identify what the user said or wrote, recognise the structure of the message, and determine the intention behind it.

Understanding Language

When text is analysed, the system must identify several layers of meaning.

- **Syntax** refers to the structure of the sentence.
- **Semantics** refers to the meaning of the words in that structure.
- **Context** influences how meaning is interpreted.
- **Sentiment** expresses whether the text conveys a positive or negative feeling.
- **Intent** describes what the user wants to achieve.

To interpret these signals, NLP systems rely on a sequence of computational steps. These steps are often grouped under **Natural Language Understanding**, where the goal is to extract meaning from the text.

Syntactic Processing

Syntactic processing focuses on the structural elements of language. It includes operations such as:

- **tokenising** text into words
- reducing words to their root forms through **stemming** or **lemmatisation**
- identifying the role of each word through part of speech tagging
- segmenting text into manageable units

These steps allow the system to recognise how the words in a sentence are organised.

Semantic Processing

Semantic processing deals with meaning.

It focuses on identifying entities such as names, locations, and organisations through Named Entity Recognition.

It also involves resolving meaning when words have multiple interpretations, a task known as word sense disambiguation. Both actions assist the system in extracting information from text and determining what the text is about.

Natural Language Understanding

The combination of syntactic and semantic processing produces **Natural Language Understanding**. This stage allows the system to analyse intent, identify subjects and objects in the sentence, recognise categories, detect mood or sentiment, and extract relevant information. As a result, the system becomes capable of responding appropriately to what the user expresses.

Natural Language Generation

After understanding the input, an application may need to generate a human readable response. This is achieved through **Natural Language Generation**. Several model families have been used for this purpose.

- **Markov chains** create text based on word probabilities.
- **Recurrent neural networks** use sequential processing of input data.
- **LSTM networks** handle longer patterns in sentences.
- **Transformer** models generate fluent and contextually consistent text.

Natural Language Generation is responsible for creating meaningful sentences that a user can understand.

Example Pipeline for NLP

In many applications, the steps appear in a pipeline.

- Text is received.
- It is then tokenised and processed syntactically.
- Semantic information is extracted.
- Named entities and relations are identified.
- Intent is detected. Sentiment may be determined.
- Finally, the system either performs an action or produces a suitable text response.

This flow demonstrates how NLP transforms raw human language into structured, meaningful information.

The following diagram illustrates this flow in a visual manner.

Tools for NLP and NLU

Several libraries are available for performing NLP tasks. These include libraries such as the:

- Stanford CoreNLP parser in Python,
- NLTK for foundational tasks,
- SpaCy for industrial NLP workflows,
- and Apache OpenNLP for Java based text processing.

These tools handle tokenisation, tagging, parsing, and the many steps that contribute to Natural Language Understanding.

NLP Components and Pipelines

Natural Language Processing systems are often organised into two broad functional areas.

The first area involves understanding human language, and the second involves generating language that is meaningful and readable.

These two areas are known as **Natural Language Understanding** and **Natural Language Generation**. Together, they form the core of how machines process and respond to human communication.

Natural Language Understanding

Natural Language Understanding focuses on converting unstructured human language into structured representations that a machine can interpret.

When text or speech is received, the system must determine what the input contains, what the user intends to express, and what information is being conveyed.

The understanding process usually begins with **identifying** the basic units of the text.

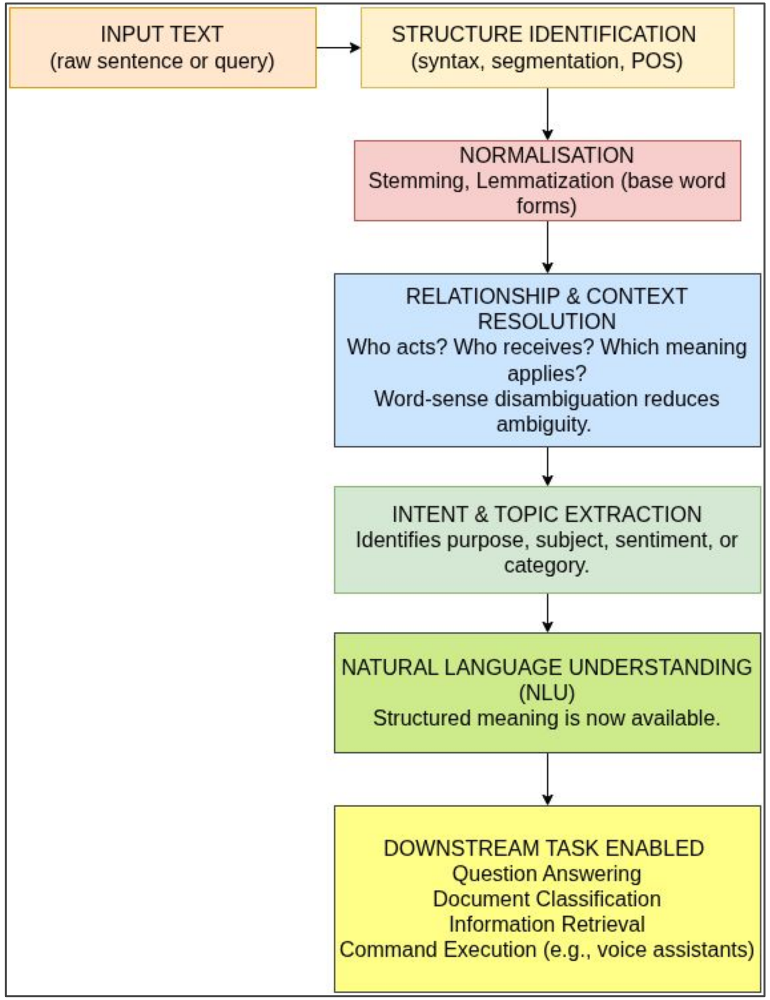
- Words are separated from each other.
- Sentences may be segmented.
- Each word is then examined so that its form and function can be identified.
- Part of speech tagging assigns grammatical roles such as noun, verb, or adjective.

This provides an initial structure for the rest of the analysis.

Once the structure is **identified**, the system moves toward **meaning**.

- Words are reduced to their base forms when required(Normalisation).
- Named Entity Recognition identifies names, places, organisations, or other entities in the text (NER).
- Relationships between words may be examined to determine who is performing an action or who is receiving it.
- Ambiguity is reduced by determining which meaning of a word applies in the given context.
- At this stage, the system is able to identify intent, detect topics, classify the text, or determine sentiment.

Natural Language Understanding transforms raw text into structured knowledge. This enables downstream tasks such as answering a question, retrieving information, classifying a document, or controlling an application through natural language commands.


INPUT TEXT
(raw sentence or query)
STRUCTURE IDENTIFICATION
(syntax, segmentation, POS)
NORMALISATION
Stemming, Lemmatization (base word forms)
RELATIONSHIP & CONTEXT RESOLUTION
Who acts? Who receives? Which meaning applies?
Word-sense disambiguation reduces ambiguity.
INTENT & TOPIC EXTRACTION
Identifies purpose, subject, sentiment, or category.
NATURAL LANGUAGE UNDERSTANDING (NLU)
Structured meaning is now available.
DOWNSTREAM TASK ENABLED
Question Answering
Document Classification
Information Retrieval
Command Execution (e.g., voice assistants)

Natural Language Generation

Natural Language Generation performs the opposite function. After a system has analysed information or made a decision, it may need to express the result in human language. Natural Language Generation creates clear and coherent sentences to convey that information.

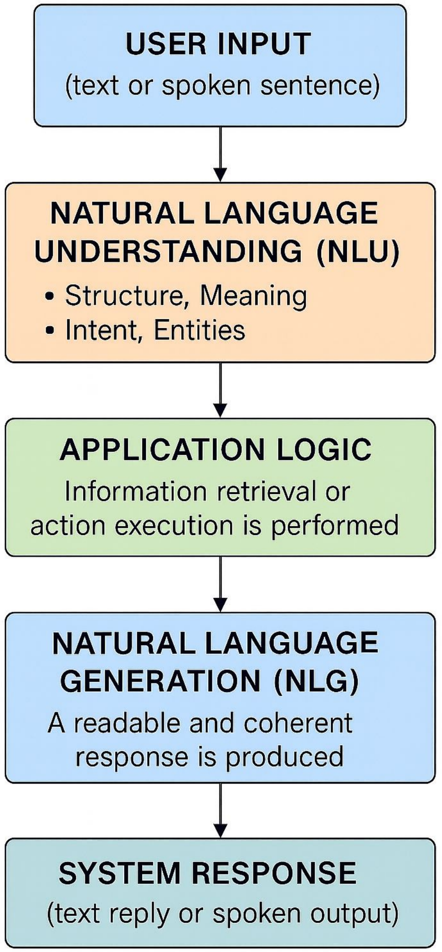
- Several model families have been used to perform text generation.
- Early models such as Markov chains generate text based on the probability of one word following another.
- Recurrent neural networks generate sequences by processing one element at a time.
- LSTM networks improve this process by retaining longer term patterns.
- Transformer models generate text by attending to relationships across the entire sequence, producing fluent and contextually consistent output.

Natural Language Generation may produce short, direct messages or longer explanations. It may summarise a large document into a compact form. It may paraphrase a sentence. It may generate text in response to a question. The output is presented in a form that users can read and understand naturally.

The Flow from NLU to NLG

Many NLP applications combine both understanding and generation. The system begins by receiving input, which may be text or speech. Natural Language Understanding extracts structure, meaning, intent, and any relevant entities. Once the meaning is recognised, the application performs the required action. The final step produces a response through Natural Language Generation.

Consider a scenario where a user asks a virtual assistant a question. The question is first processed through Natural Language Understanding to identify what is being asked. The system retrieves the information or performs the action. It then constructs a readable reply through Natural Language Generation. This chain of steps forms a typical NLP pipeline.


USER INPUT
(text or spoken sentence)
NATURAL LANGUAGE UNDERSTANDING (NLU)
• Structure, Meaning
• Intent, Entities
APPLICATION LOGIC
Information retrieval or action execution is performed
NATURAL LANGUAGE GENERATION (NLG)
A readable and coherent response is produced
SYSTEM RESPONSE
(text reply or spoken output)

Putting it all Together

NLU and NLG operate together to support a wide variety of applications.

- **Chatbots** interpret user queries and produce conversational responses.
- **Question answering systems** extract relevant answers from text and present them in readable form. Translation systems understand the structure and meaning of text in one language and generate a corresponding version in another language.
- **Sentiment systems understand** the attitude expressed in text and generate output that reflects the detected emotion.

Transformer Applications in NLP

Transformer based models have become central to modern Natural Language Processing because they handle context more effectively than earlier approaches. They process all parts of a sentence simultaneously, which allows them to capture long range relationships between words and interpret subtle variations in meaning. As a result, transformer models are used in many core NLP tasks.

These tasks cover a wide range of real world applications, such as answering questions, detecting spam, identifying sentiment, translating languages, generating text, summarising documents, and paraphrasing content. Each task relies on the model's ability to understand and manipulate natural language in a structured way.

Applications of Transformers (NLP)

Question Answering

Question answering systems identify answers to questions within a given passage or context. A user provides a question and the supporting text. The transformer model analyses both, determines how the question relates to the passage, and selects the portion that answers it. This capability is used in virtual assistants and knowledge retrieval systems.

Spam Detection

Spam detection identifies unwanted or harmful messages in communication systems such as email or messaging platforms. Transformer models examine the content of a message, recognise suspicious patterns, and classify the message accordingly. The model's understanding of text structure, intent, and phrasing allows it to distinguish between legitimate and unwanted messages.

Sentiment Analysis

Sentiment analysis, also known as opinion mining, evaluates whether a piece of text expresses a positive, negative, or neutral feeling. Transformer models identify the tone and emotional cues present in the text. These cues may include descriptive phrases, expressions of approval, complaints, or other markers of opinion. Sentiment analysis is used in product reviews, customer feedback, and social media monitoring.

Machine Translation

Machine translation converts text or speech from one natural language to another. Transformer models are particularly effective in this task because they process sequences while attending to the meaning of each word in context. As a result, the translated text is more fluent and captures the original meaning accurately. This has become a standard approach in multilingual applications.

Text Generation

Text generation produces new text based on user input. A prompt is given, and the transformer model continues the text or creates new sentences that follow logically from the prompt. The generated output aligns with the structure and style of the input. This ability is used in creative writing support, chat systems, and content creation tools.

Text Summarisation, Paraphrasing, and Error Correction

Transformers are also used in summarisation. A long passage can be condensed into a shorter version while preserving its key meaning. This is helpful when handling lengthy documents or articles. Paraphrasing transforms a sentence into an alternative form that conveys the same meaning. Grammatical error correction identifies errors in text and produces a corrected version. These applications appear in writing assistance tools.

Computer Vision APIs

Computer Vision APIs provide ready made capabilities for analysing images and videos. These APIs allow applications to:

- detect objects,
- recognise faces,
- classify images,
- or extract features without the need to train models.

The intelligence is accessed through high level interfaces, which return structured output such as predicted labels, bounding boxes, or descriptive information about the visual content.

Cloud platforms expose computer vision features as services that can be called using REST interfaces or SDKs. These capabilities include object detection, face recognition, image classification, feature extraction, image restoration, object tracking, and scene reconstruction. Such tasks form the foundation of many modern applications that rely on visual analysis.

Key Computer Vision Tasks

Object detection identifies and localises objects within an image or video. Face recognition identifies individual faces and is used in authentication or security systems. Feature extraction identifies patterns that distinguish one object from another. Image classification recognises the primary category of an image. Image restoration improves clarity or reconstructs missing portions of an image. Object tracking follows movement across video frames. Scene reconstruction produces a three dimensional structure using multiple images.

Each of these abilities is made available through Computer Vision APIs that accept an image and return analysis results.

Example of a Real Computer Vision API

A practical API example appears in the image classification demonstration using a Hugging Face transformer model. The API can load a model for classification and process an image directly through a URL. The following code illustrates a working Computer Vision API call:

```
from transformers import pipeline

# Load the image classification pipeline

image_classifier = pipeline("image-classification",model="facebook/deit-base-distilled-patch16-224")

# Use a new image URL

url = "https://upload.wikimedia.org/wikipedia/commons/9/9a/Pug_600.jpg"

# Perform image classification using the URL directly

result = image_classifier(url)

# Print the result

print(result)
```

The model processes the image hosted at the given URL and returns predictions indicating the most likely classes.