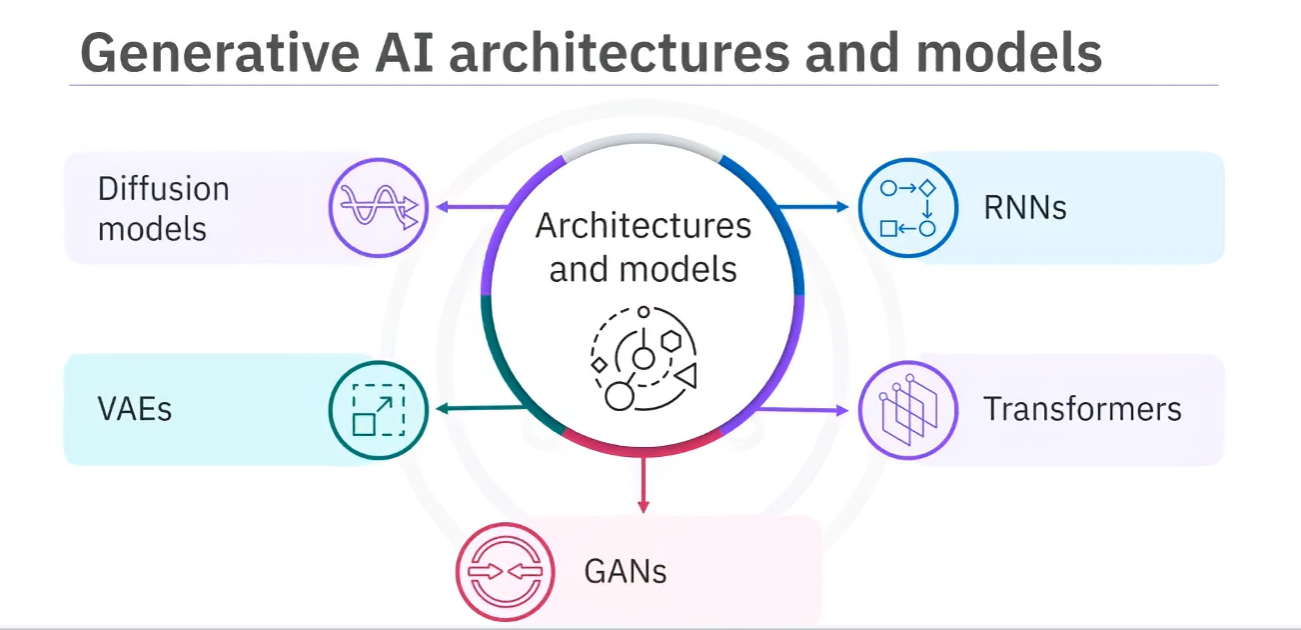
Generative AI

# Generative AI Architectures and Models

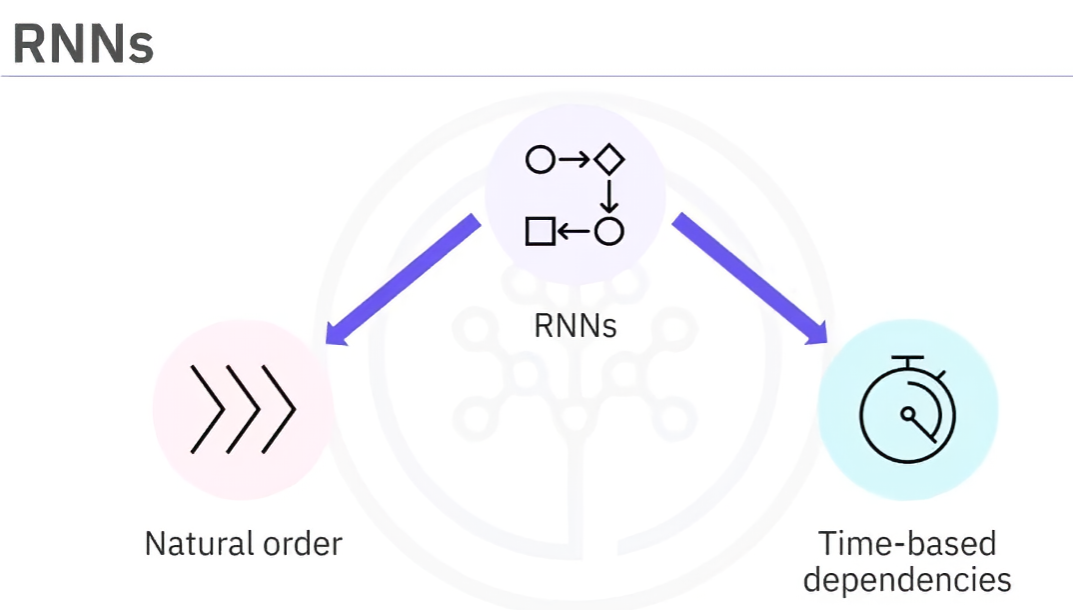
**1. Introduction**

* Overview of generative AI architectures used in text, speech, and image generation.
* Importance of choosing the right model for specific tasks, like personalized video generation.

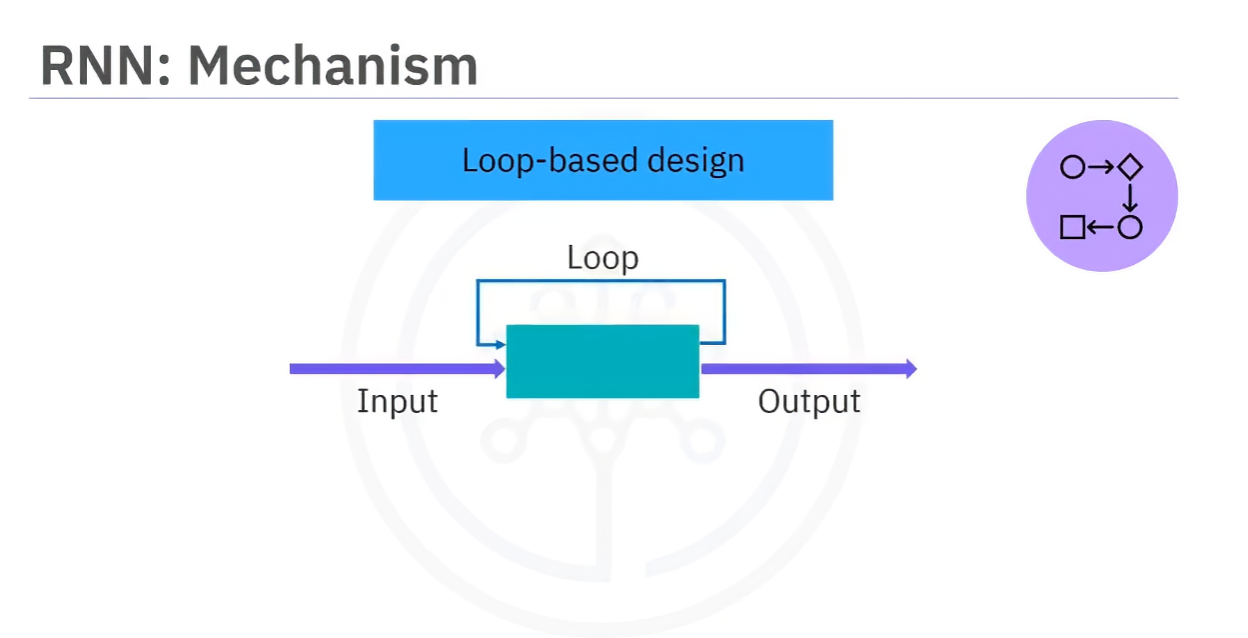
**2. Common Generative AI Architectures & Models**



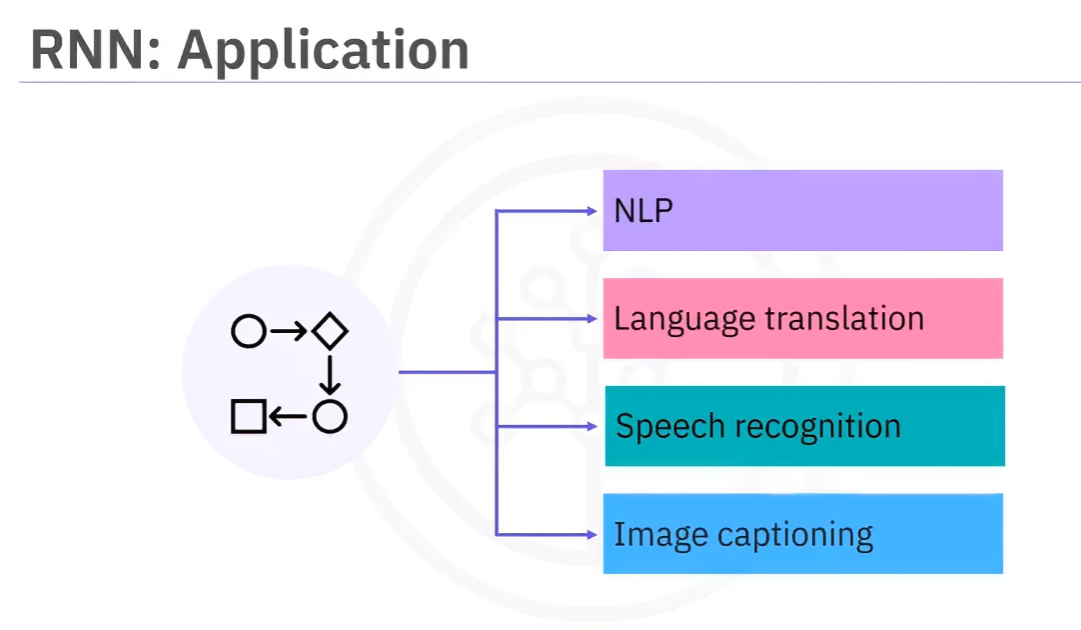
* **Recurrent Neural Networks (RNNs)**



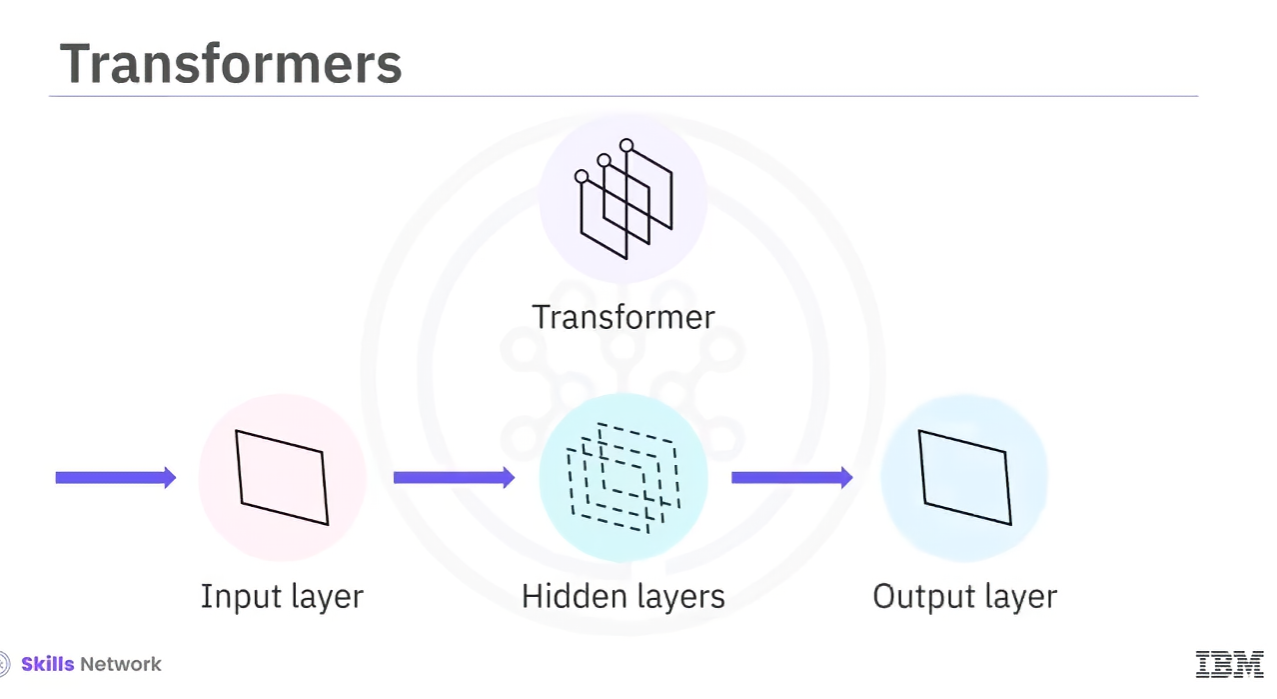
* + Used for sequential or time-series data (e.g., language modeling, speech recognition).
  + Loop-based design enables memory of previous inputs for context.

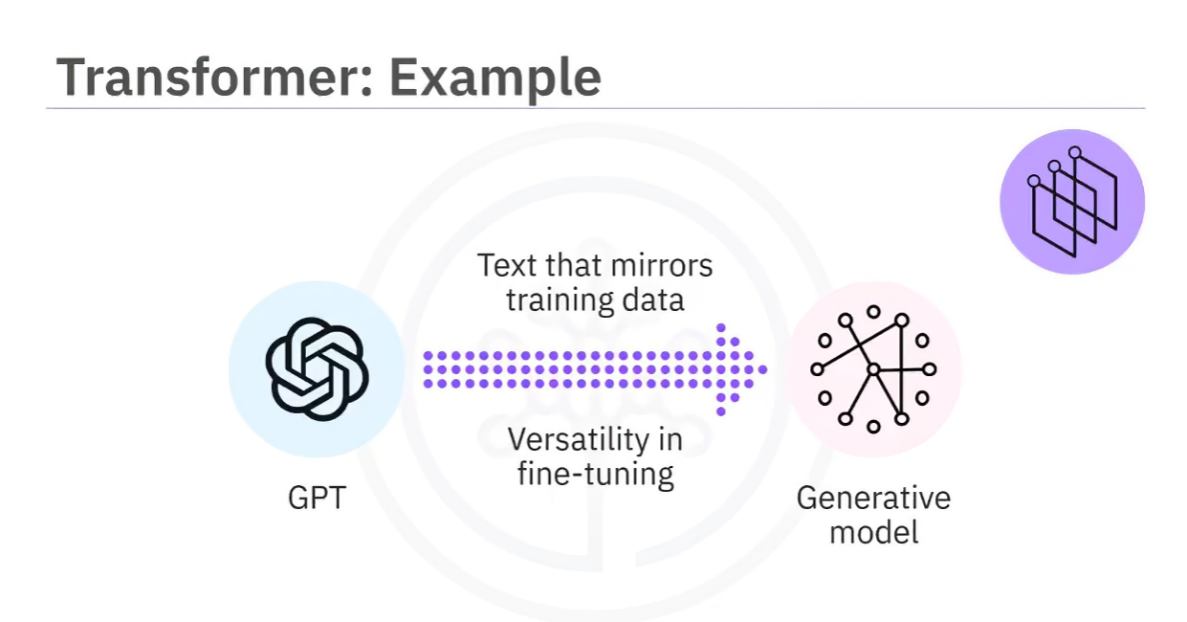
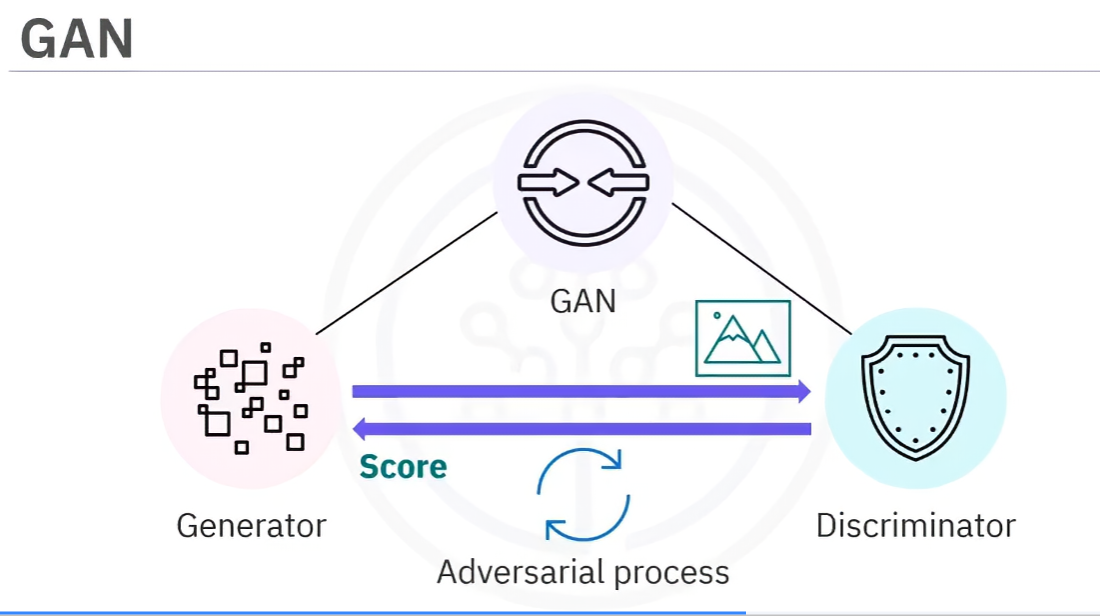


* + Fine-tuning involves adjusting weights for specific tasks.



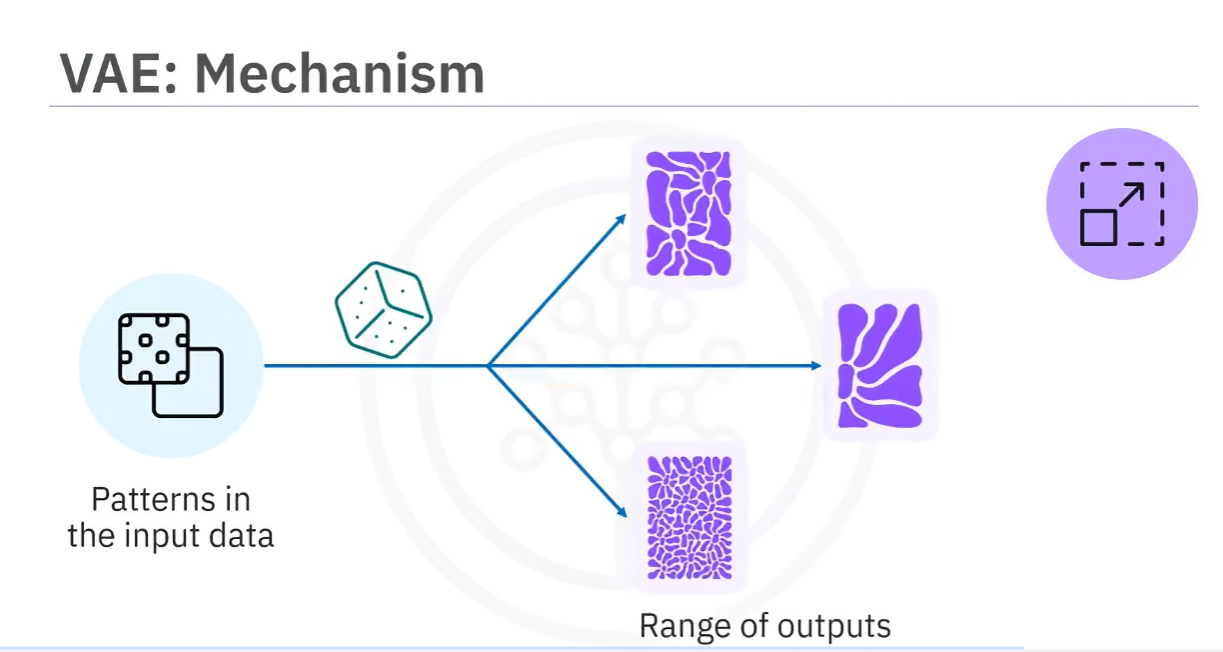
* **Transformers**



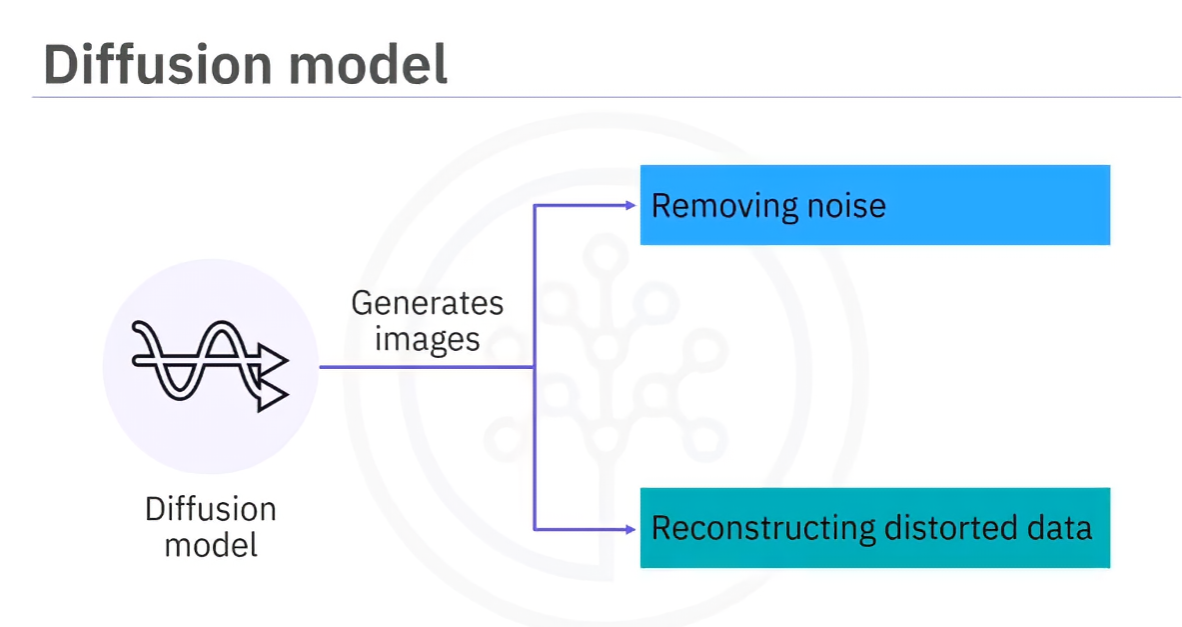
* + Deep learning models with a **self-attention mechanism** for focusing on key input segments.
  + Information flows through multiple layers, allowing efficient parallel training.
  + **Fine-tuning** typically involves adjusting only output layers while keeping core layers fixed.
  + Example: **GPT (Generative Pretrained Transformer)** – excels at text generation.
* **Generative Adversarial Networks (GANs)**
  + Consist of two components:
    - **Generator** – creates synthetic data.
    - **Discriminator** – evaluates authenticity by comparing with real data.
  + Used in **image & video generation** through an adversarial process.
* **Variational Autoencoders (VAEs)**



* + Operate on an **encoder-decoder framework** to learn latent data representations.
  + Represent data as probability distributions, capturing patterns for generating new samples.



* + Applied in **art, creative design, and feature extraction**.
* **Diffusion Models**



* + Probabilistic generative models trained to **remove noise** from data.
  + Used for **image generation and restoration** (e.g., restoring old or distorted images).

**3. Differences in Training Approaches**

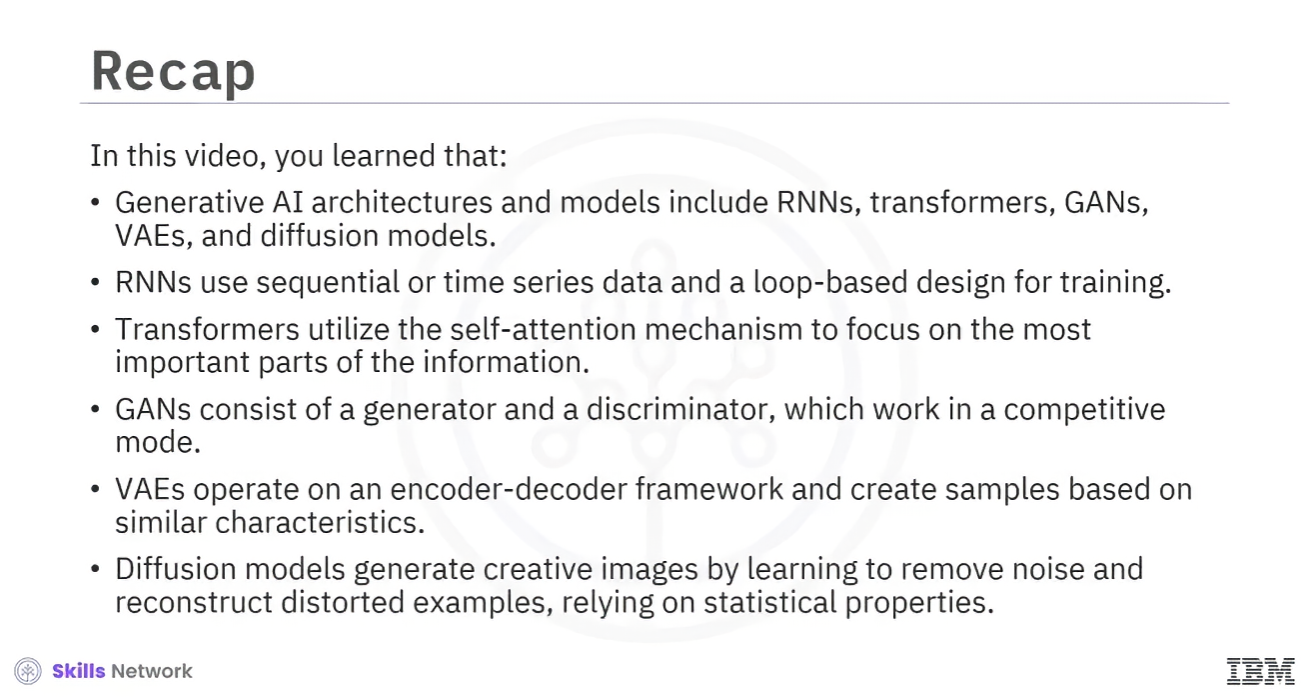
* **RNNs**: Loop-based memory structure for sequential data.
* **Transformers**: Self-attention mechanism for efficient parallel training.
* **GANs**: Competitive learning process between generator and discriminator.
* **VAEs**: Encoder-decoder architecture capturing latent distributions.
* **Diffusion Models**: Statistical approach for noise removal and reconstruction.

**4. Relationship Between Generative AI & Reinforcement Learning (RL)**

* Traditional **RL**: AI agents interact with environments to maximize rewards.
* Generative AI models **use RL techniques** for optimization and fine-tuning (e.g., RLHF in ChatGPT).

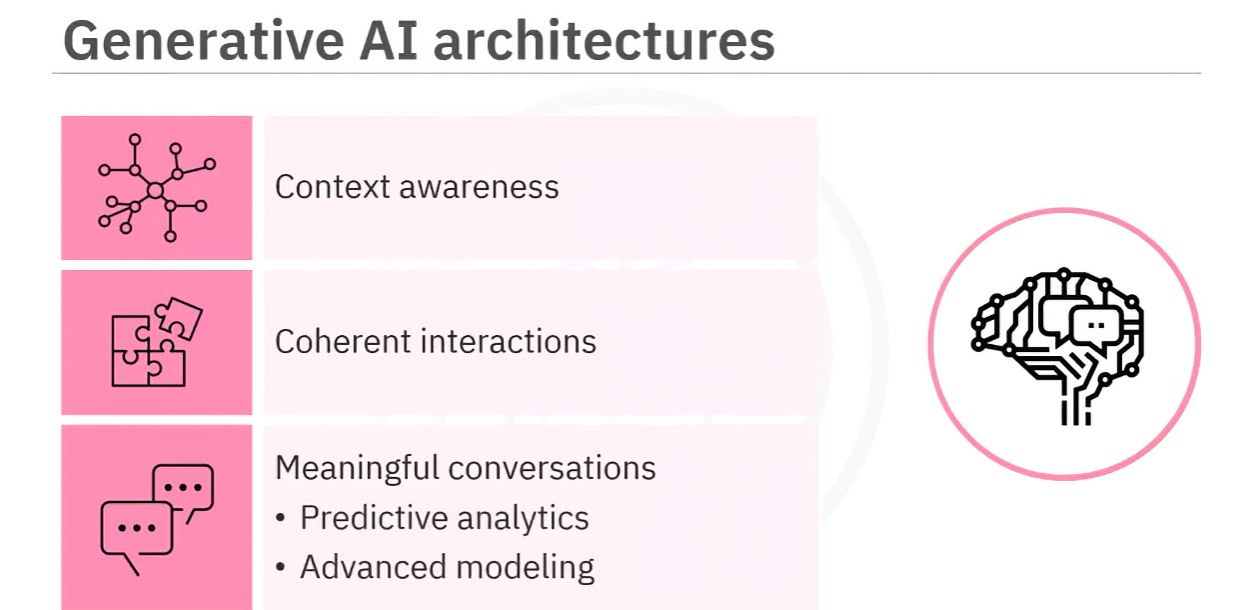
**5. Recap**

* **RNNs, transformers, GANs, VAEs, and diffusion models** are key architectures in generative AI.
* **Each model has unique strengths**:
  + **RNNs**: Sequential processing.
  + **Transformers**: Self-attention for text/speech tasks.
  + **GANs**: Adversarial training for realistic image/video synthesis.
  + **VAEs**: Probabilistic approach for new sample generation.
  + **Diffusion models**: Noise removal for high-quality image generation.
* **Reinforcement learning is used to enhance generative AI model performance.**



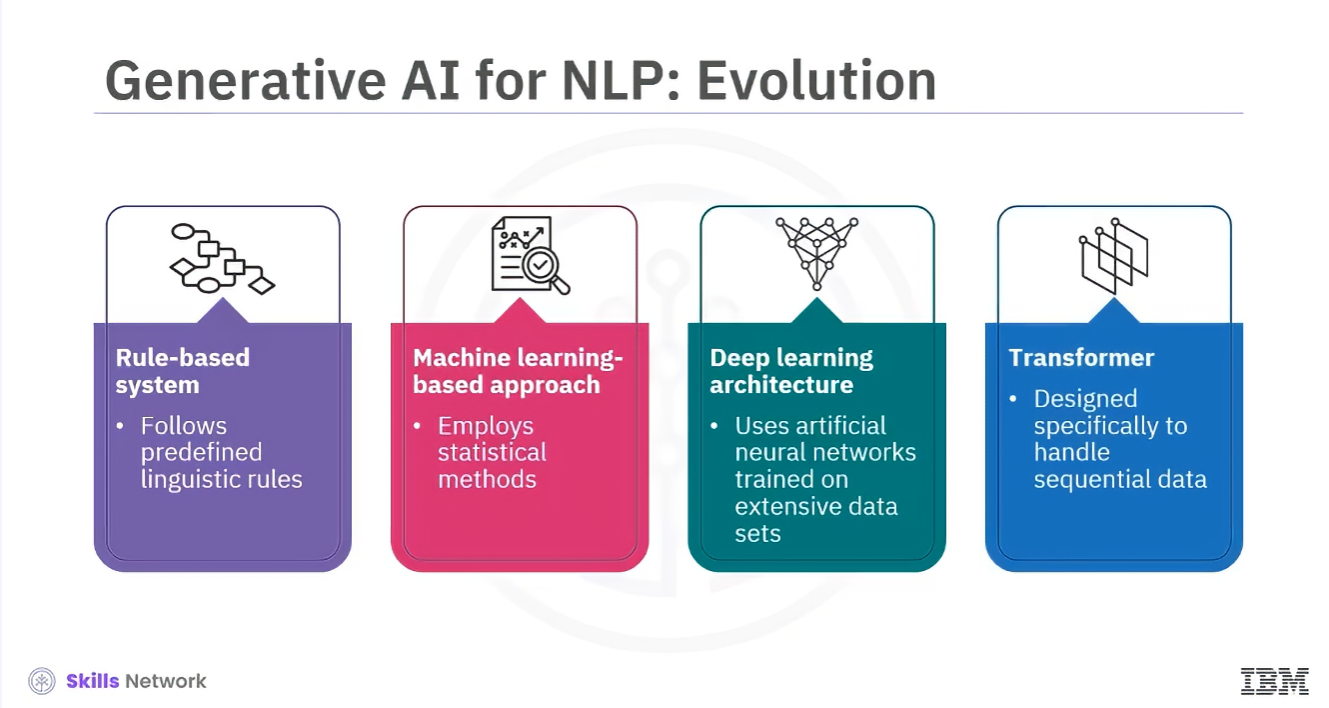
## Generative AI for NLP

**1. Introduction**



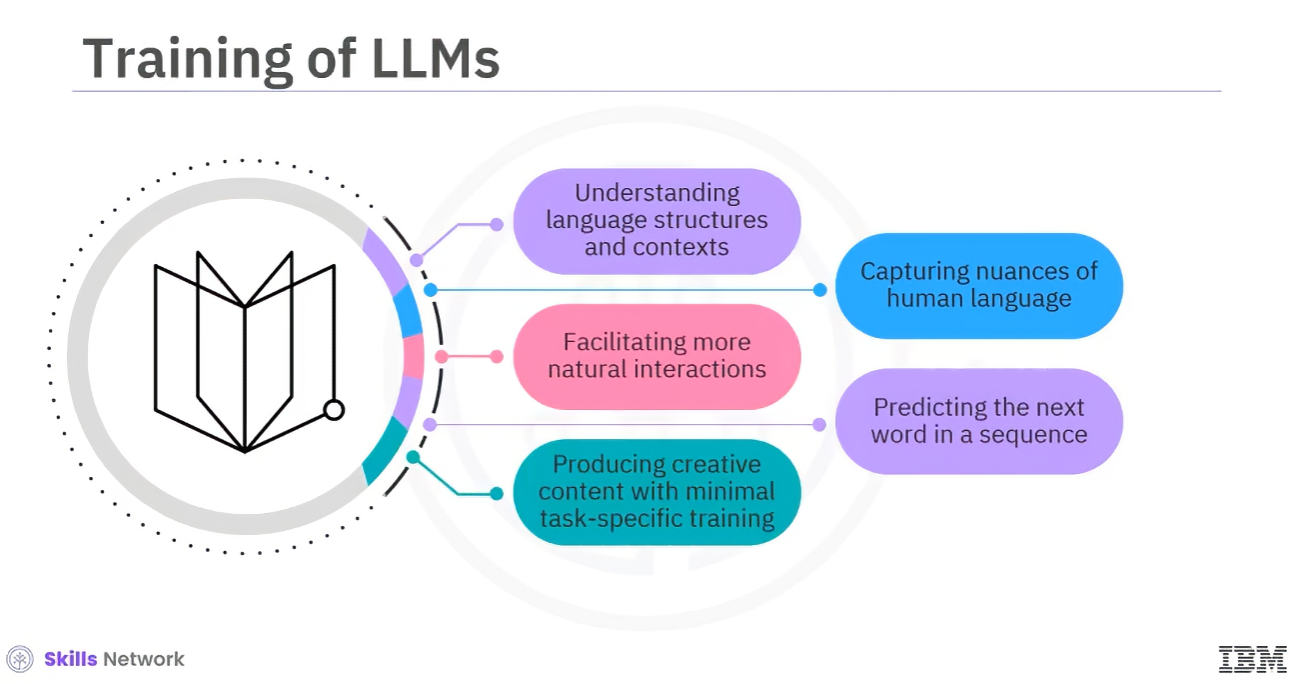
* Generative AI enables machines to understand and generate human-like language.
* Helps develop applications like chatbots and virtual assistants.
* Enhances language processing through context awareness and predictive analytics.

**2. Evolution of Generative AI for NLP**



* **Rule-based systems**: Follow predefined linguistic rules but lack flexibility.
* **Machine learning approaches**: Use statistical methods to learn from large datasets.
* **Deep learning**: Trains artificial neural networks for better language interpretation.
* **Transformers**: Latest advancement, excelling in context understanding and dependencies.

**3. Applications of Generative AI in NLP**



* **Machine translation**: Improves accuracy and context awareness.
* **Chatbots & virtual assistants**: Enhances conversation naturalness and personalization.
* **Sentiment analysis**: Captures subtle language expressions for deeper insights.
* **Text summarization**: Extracts key information for precise summaries.

**4. Large Language Models (LLMs)**

* **Foundation models trained on massive datasets (petabytes of data).**
* **Contain billions of parameters optimized for specific tasks.**
* **Capable of text generation, translation, and content creation.**
* **Examples:**
  + **GPT**: Text generation (chatbots, content creation).
  + **BERT**: Understanding word context (sentiment analysis, Q&A).
  + **BART & T5**: Encoder-decoder models for multiple NLP tasks.

**5. GPT vs. ChatGPT**

* **GPT**: General text generation, trained mainly using supervised learning.
* **ChatGPT**: Focuses on conversations, fine-tuned using Reinforcement Learning from Human Feedback (RLHF).

**6. Importance & Challenges**

* LLMs contribute to **natural language understanding and generation advancements**.
* Models can be **fine-tuned for specific industries** (e.g., retail product categorization).
* Risks: **Potential misinformation, biases, and societal impact**.

**7. Recap**

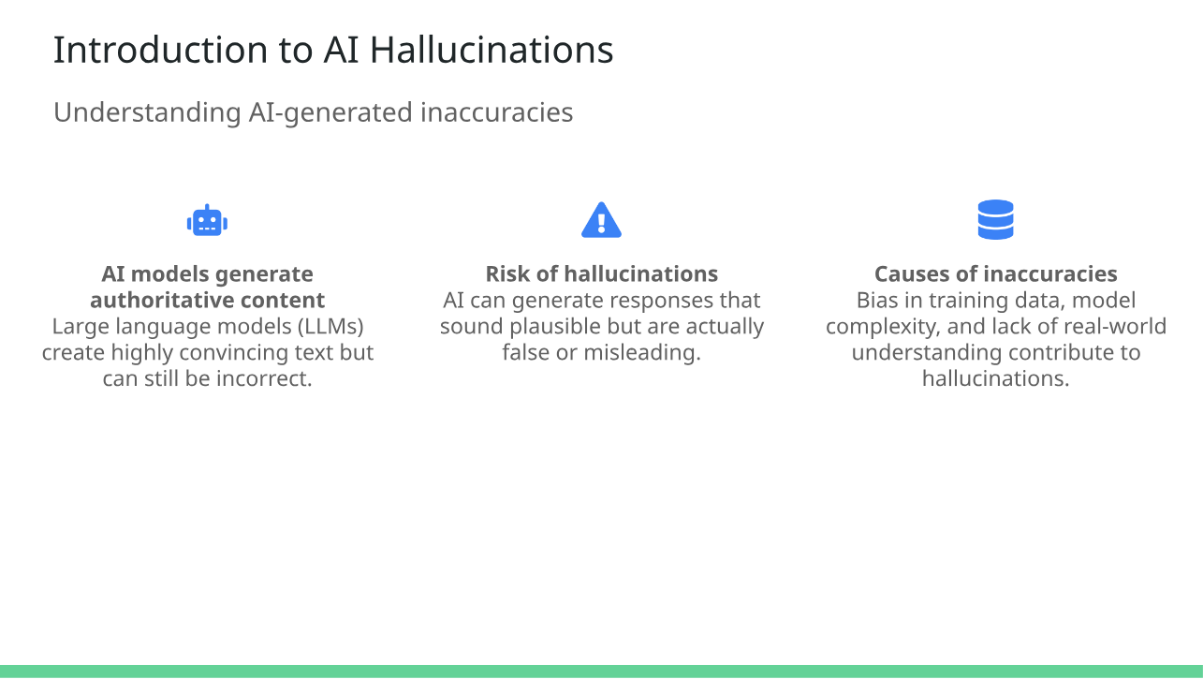
* **Generative AI has evolved from rule-based systems to transformers.**
* **LLMs like GPT, BERT, BART, and T5 play key roles in NLP advancements.**
* **Applications include machine translation, chatbots, sentiment analysis, and summarization.**
* **Fine-tuning allows customization but requires careful handling of biases and accuracy.**

## Basics of AI Hallucinations

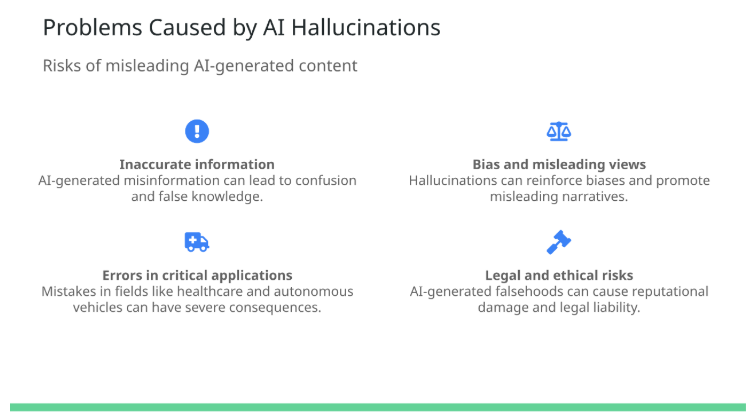
**2. Introduction**

1. Large language models (LLMs) generate authoritative text but may produce inaccurate or biased content.
2. These inaccuracies can result from AI hallucinations.

**3. AI Hallucinations**



1. Occurs when an AI model generates output that appears accurate but is unrealistic, inaccurate, irrelevant, or nonsensical.
2. Similar to human hallucinations.
3. Example:
   * ChatGPT falsely claimed an Australian mayor was guilty of bribery, though he had actually reported the issue.
4. Causes:
   * Biases in training data.
   * Limited training.
   * Model complexity.
   * Lack of human oversight.
   * Outputs not based on learned patterns.
5. **Problems Caused by AI Hallucinations**



1. **Inaccurate information generation.**
2. **Creation of biased or misleading views.**
3. **Incorrect input in sensitive applications** (e.g., autonomous vehicles, medical diagnosis).
4. **Potential legal or ethical consequences** (e.g., misrepresentation in legal documents).

**5. Methods for Mitigating AI Hallucinations**

1. Train models on high-quality, unbiased data.
2. Avoid manipulation of input data.
3. Continuously evaluate and improve models.
4. Fine-tune pre-trained LLMs using domain-specific data.

**6. Preventing Problems from AI Hallucinations**

1. **Acknowledge limitations:** AI models predict words based on patterns, not actual understanding.
2. **Ensure human oversight:** Regular fact-checking and testing.
3. **Provide additional context:** Better input improves response accuracy.

**7. Summary**

1. **AI hallucinations** occur when AI-generated output appears correct but is inaccurate or nonsensical.
2. **Problems include:**
   * Inaccurate information.
   * Biased views.
   * Incorrect input for critical applications.
3. **Prevention methods:**
   * Extensive high-quality training.
   * Avoiding input manipulation.
   * Continuous model improvement.
   * Domain-specific fine-tuning.
   * Human oversight.
   * Providing context in prompts.

# Overview of Libraries and Tools in Generative AI for NLP

**Objective**  
After reading this, you will be able to describe key features and the significance of libraries and tools used in generative AI for NLP.

**Introduction**

* Generative AI applications for NLP require a deep understanding of linguistic nuances.
* Various libraries and tools make these applications more accessible and efficient.
* Key libraries and tools include PyTorch, TensorFlow, Hugging Face, LangChain, and Pydantic.

**PyTorch**

* Open-source deep learning framework developed by Facebook (Meta).
* Features dynamic computation graphs (Autograd) for flexibility.
* Rich ecosystem with tools like **torchtext** for NLP.
* Used in research and development for neural network models in NLP.

**TensorFlow**

* Open-source machine learning and deep learning framework developed by Google.
* Scalable architecture for transitioning from research to production.
* **TensorFlow Extended (TFX)** supports production-ready ML pipelines.
* **Keras integration** provides a user-friendly API for deep learning.
* Used for NLP tasks such as sentiment analysis and text classification.

**Hugging Face**

* Platform with an open-source library offering pretrained models.
* **Extensive Model Hub** with models for translation, question-answering, etc.
* **Transformers Library** simplifies usage of pretrained models for NLP.
* **Datasets Library** provides large-scale datasets for evaluation.
* **Tokenizers Library** optimizes tokenization for NLP models.
* Used for NLP applications such as named entity recognition and text summarization.

**LangChain**

* Open-source framework for developing AI applications with LLMs.
* **Advanced prompt engineering** tools for refining model responses.
* **Seamless integration** with popular models like GPT.
* Used for building chatbots and analytical tools.

**Pydantic**

* Python library for data validation and parsing.
* Ensures **data integrity** through robust validation.
* Supports **efficient settings management** for scalable applications.
* Used in NLP pipelines to validate and manage large datasets.

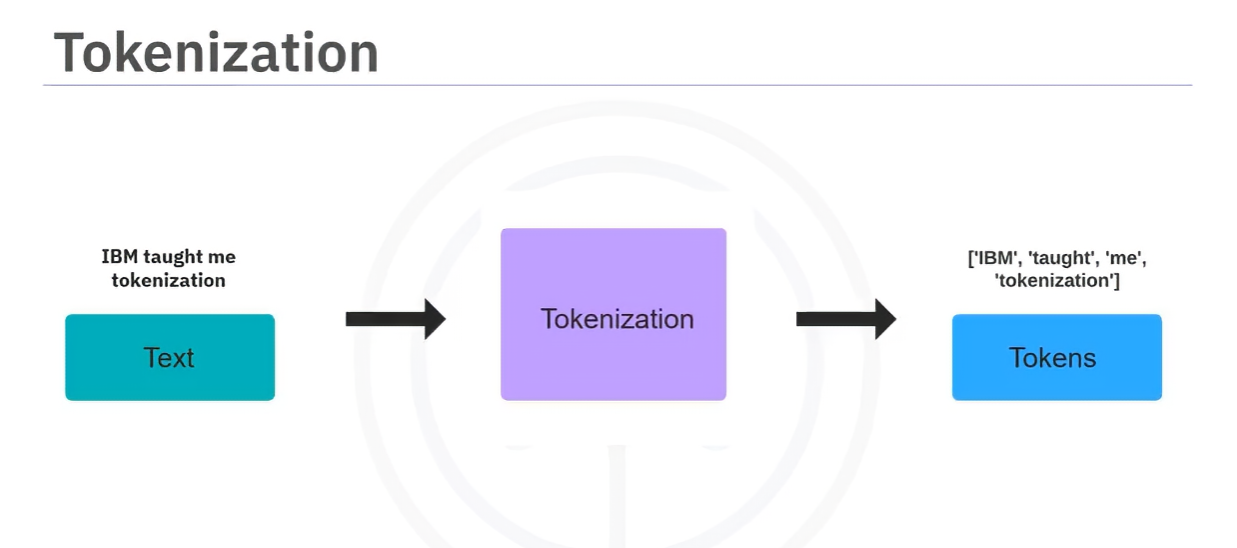
**Summary**

* Various libraries and tools help develop NLP applications using generative AI.
* **PyTorch**: Flexible deep learning framework with dynamic computation graphs.
* **TensorFlow**: Scalable framework with Keras integration for deep learning.
* **Hugging Face**: Platform offering pretrained models and NLP tools like Transformers.
* **LangChain**: Framework for building AI applications using LLMs with prompt engineering.
* **Pydantic**: Ensures data validation and integrity in NLP applications.

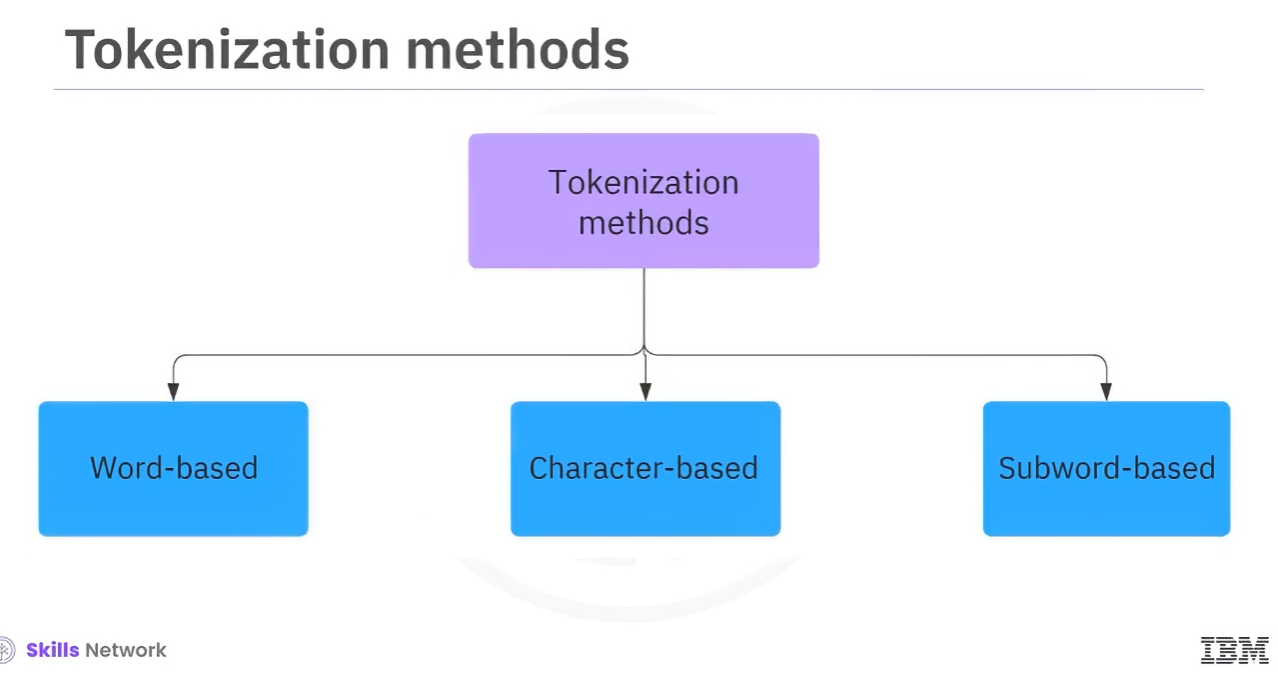
# Preparing Data

## Tokenization Process for AI Models

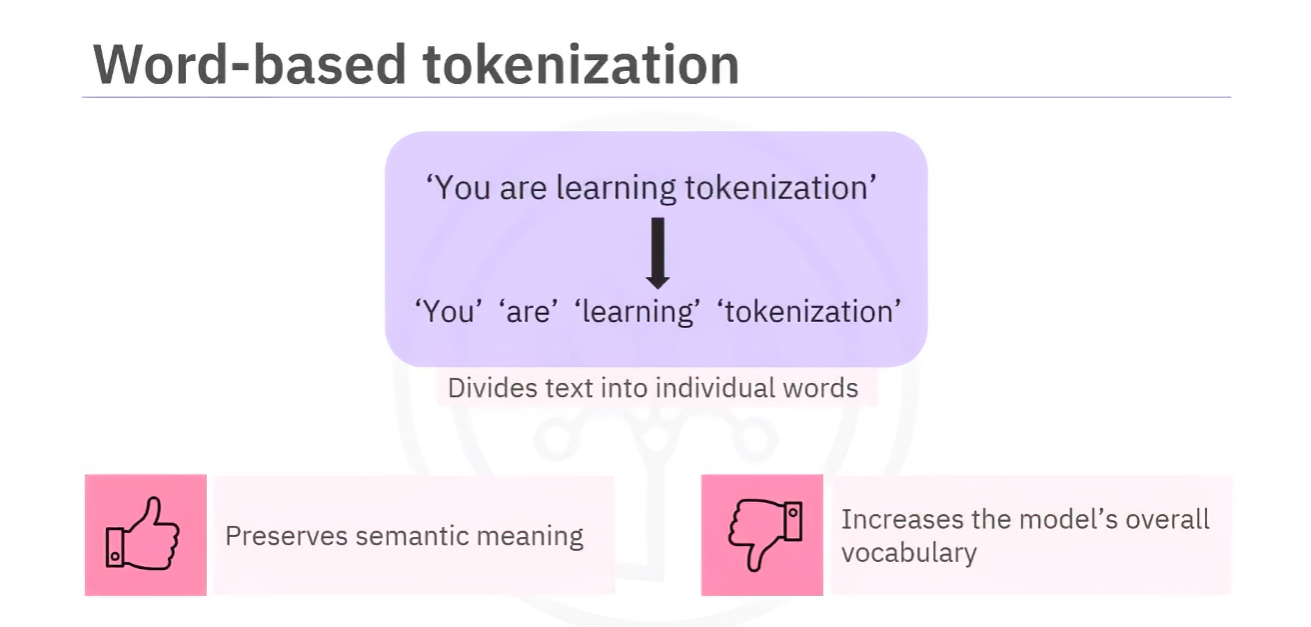
1. **Introduction to Tokenization**:
   * Tokenization breaks down text into smaller pieces, called tokens, to help AI models understand the text.
   * Example: Sentence "IBM taught me tokenization" becomes tokens: **IBM**, **taught**, **me**, **tokenization**.

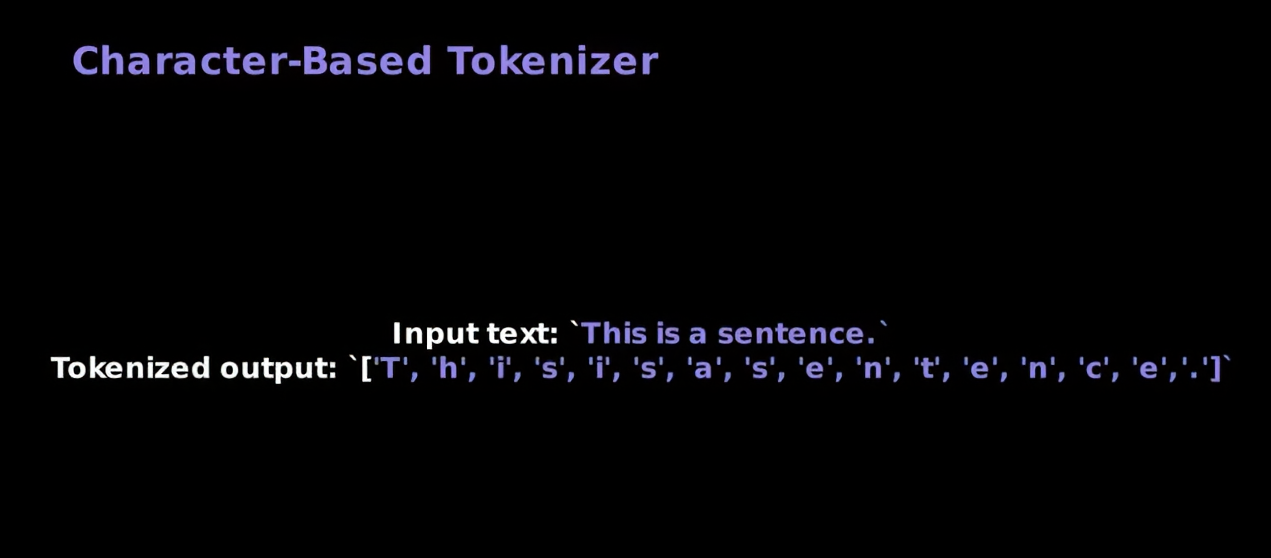


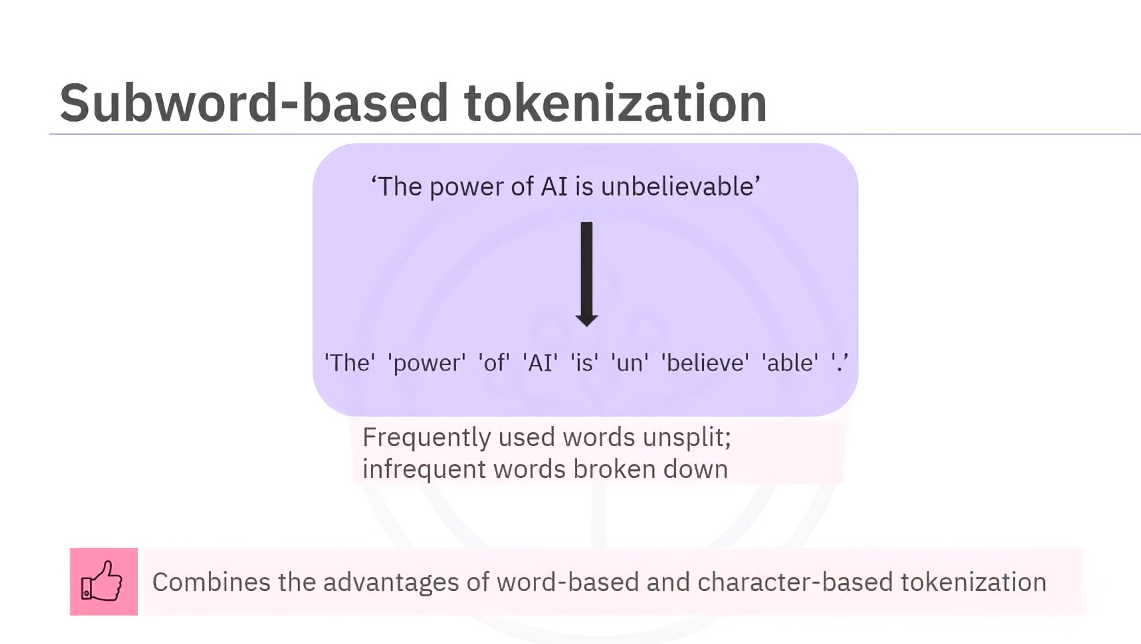
1. **Types of Tokenization**:



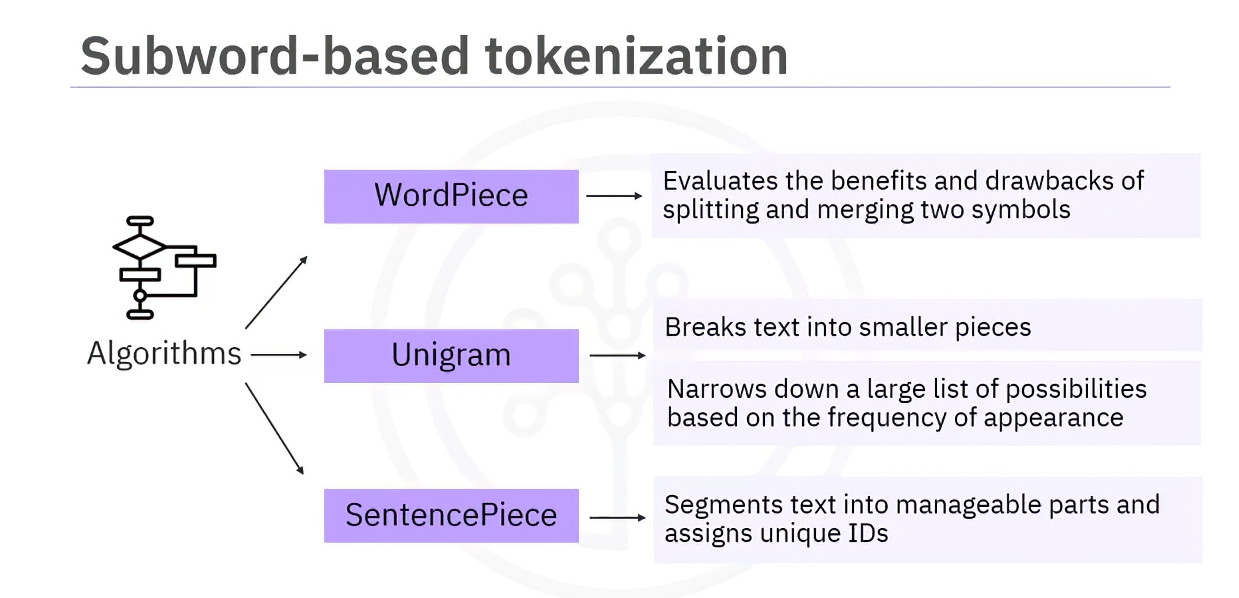
* + **Word-based Tokenization**:



* + - Splits text into individual words, each considered a token.
    - **Advantages**: Preserves semantic meaning.
    - **Disadvantages**: Increases model vocabulary size, can lead to issues with variations of words (e.g., "unicorn" vs. "unicorns").
  + **Character-based Tokenization**:
    - * 
    - Splits text into individual characters.
    - **Advantages**: Small vocabulary size.
    - **Disadvantages**: Characters may not convey the same meaning as words, increases computational load.
  + **Subword-based Tokenization**:



* + - Combines word-based and character-based methods.
    - Frequently used words remain unsplit, while rare words are split into meaningful sub-words.
    - **Advantages**: Efficient for handling infrequent words and preserving semantic meaning.
    - Examples of algorithms:



* + - * **WordPiece**: Merges and splits symbols based on value.
      * **Unigram**: Narrows down token possibilities iteratively.
      * **SentencePiece**: Segments text into manageable pieces and assigns unique IDs.

1. **Tokenizers**:
   * Tools like **NLTK** and **spaCy** tokenize text.
   * These tokenizers handle word-based tokenization but may split similar words differently.
2. **PyTorch Implementation**:
   * Use **torchtext** to tokenize sentences into words or sub-words.
   * build\_vocab\_from\_iterator function creates a vocabulary and maps tokens to indices.
   * Example code:
     + Tokenize sentences and convert tokens to indices.
     + Special token **UNK** used for unknown words.
     + Vocabulary (vocab.get\_stoi) provides a dictionary mapping tokens to indices.
3. **Adding Special Tokens**:
   * Special tokens like **BOS** (beginning of sentence) and **EOS** (end of sentence) can be added.
   * Tokenized sentences may be padded with **pad tokens** to ensure uniform sentence length.
4. **Key Takeaways**:
   * Tokenization is essential for breaking text into meaningful parts for AI models.
   * Different tokenization methods serve different purposes and have unique trade-offs.
   * Special tokens and padding ensure consistent input formatting for machine learning models.

