Project Context: Smart Inventory Management

The goal is building a system that automatically tracks products in retail stores. Instead of manually counting items, the system updates everything in real-time and sends alerts for low stock or expired products.

Python Fundamentals I've Learned

Type Conversion (Casting)

When I need to change data types:

- `int()` converts to whole numbers
- `float()` gives decimal numbers
- `str()` turns things into text

Quick example: `float(5)` becomes `5.0`

Writing Functions

Basic pattern I use:

```python
def my_function():
    # my code here
```

String Formatting with f-strings

This is my favorite way to put variables in strings - super clean:

```python
name = "Mannat"
print(f"Hello {name}")
```

Key Libraries

NumPy - For fast mathematical operations

Pandas - My go-to for working with data tables (DataFrames)

- `df.head()` - peek at first rows

- `df.describe()` - get statistics

Matplotlib - Creating visualizations

- Line charts: `plt.plot()`
- Bar charts: `plt.bar()`
- Pie charts: `plt.pie()`
- Scatter plots: `plt.scatter()`

Always remember to add `plt.title()`, `plt.xlabel()`, `plt.ylabel()`, and `plt.show()`

Understanding AI Models

AI models learn patterns from data and make predictions or generate content.

Major Players

- **Google** → Gemini models
- **OpenAI** → GPT series
- **Groq** → Llama models

What's an LLM?

Large Language Models are trained on massive text datasets to understand and create human-like text.

Examples I'm familiar with:

- GPT-4o, GPT-3.5 (OpenAI)
- Gemini, Gemma (Google)
- Llama 3, Llama 3.1 (Meta)

Chat Completion APIs

How conversational AI works - you send message history, it responds with the next message. This powers ChatGPT, Gemini, Claude, etc.

Database Knowledge

PostgreSQL Basics

Open-source database that organizes data in tables (rows & columns). Uses SQL for queries.

PostgreSQL vs MySQL:

- PostgreSQL: Better for complex queries, supports advanced features
- MySQL: Faster for simple read operations

Vector Databases

Regular databases can't search by meaning - only exact matches. Vector DBs let you:

- Find similar documents
- Match similar images
- Get recommendations based on similarity

Machine Learning Types

1. Supervised Learning

Training with labeled examples (input + correct answer)

- Price prediction
- Spam detection
- Face recognition

2. Unsupervised Learning

Finding patterns without labels

- Customer segmentation
- Detecting unusual behavior
- Recommendations

3. Reinforcement Learning

Learning through rewards and penalties (trial and error)

4. Semi-Supervised Learning

Mix of labeled and unlabeled data (useful when labeling is expensive)

API Concepts

What's an API?

Acts as a messenger between software applications. One app requests something, API delivers it.

Example: Weather app requests temperature data via weather API

HTTP Methods

GET - Retrieve data (no body needed) **POST** - Send/create data (includes payload)

Important Terms

- **Payload**: The actual data being sent
- **Endpoint**: Specific URL path for API requests (e.g., `/api/users`)

Flask vs FastAPI

Flask - Simple, beginner-friendly, good for smaller projects

```python
from flask import Flask, jsonify
app = Flask(__name__)

@app.route("/hello")
def hello():
    return jsonify({"message": "Hello World"})
```

FastAPI - Modern, high-performance, built-in validation

```python
from fastapi import FastAPI
app = FastAPI()

@app.get("/hello")
def hello():
    return {"message": "Hello World"}
```

Python Environment Setup

Virtual Environment Commands

Create: `python -m venv venv` Activate (Windows): `venv\Scripts\activate` Deactivate: `deactivate`

Managing Dependencies

Create requirements file: `requirements.txt` Install packages: `pip install -r requirements.txt`

Git Workflow

Essential Commands

- `git add .` - Stage all changes
- `git commit -m "message"` - Save changes
- `git push` - Send to remote
- `git pull` - Get latest changes
- `git branch` - View branches
- `git checkout branch_name` or `git switch branch_name` - Switch branches
- `git checkout -b new_branch` - Create and switch to new branch
- `git fetch --all` - Download changes without merging

Git vs GitHub

Git - Version control tool (works offline) **GitHub -** Online platform for hosting Git repositories

Check version: `git --version`

Repository = Storage for project files + complete change history

---

AI Agents Explained

Agentic AI Workflow

Unlike simple chatbots, these agents can plan and take actions:

1. **Goal** - Receive task (e.g., "Book flight Delhi to Jaipur")
2. **Observe** - Check environment/data
3. **Plan** - Decide sequence of actions
4. **Execute** - Perform actions
5. **Monitor** - Check results, adjust if needed

Agent Types

Reactive Agent - Responds immediately to current situation (no memory) Example: Robot vacuum changing direction when hitting obstacles

Goal-Driven Agent - Plans actions to achieve specific goals

Multi-Agent System - Multiple agents working together

How Agents Work

Perception → Thinking → Planning → Acting → Monitoring

AutoGen Framework

- **User Proxy Agent**: Represents the user
- **Assistant Agent**: Does the actual work

Working with OpenAI API

Installation

```bash
pip install openai
```

Basic Usage Pattern

```python
from openai import OpenAI

client = OpenAI()

resp = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": "Write a two-line poem about chai."}
    ],
    max_tokens=120
)
```

Breaking it down:

- `client = OpenAI()` - Creates connection to OpenAI
- `model` - Which AI model to use
- `messages` - Conversation history
- `max_tokens` - Response length limit

Google's Gemini API

```
bash

pip install google-generativeai
```

```
python

import google.generativeai as genai

genai.configure(api_key="YOUR_API_KEY")
model = genai.GenerativeModel("gemini-1.5-flash")
response = model.generate_content("Write a poem about coffee.")
print(response.text)
```

How LLMs Process Text

1. **Text → Tokens**: Breaking text into small pieces (words, parts of words)
2. **Tokens → Embeddings**: Converting to number vectors that capture meaning
3. **Transformer Layers**: Analyzing relationships between tokens
4. **Attention Mechanism**: Focusing on important connections
5. **Output Generation**: Predicting next tokens, converting back to text

Prompt Engineering

Crafting the right input to get desired output from AI models

Rate Limit - Max requests per minute **Quota** - Max total usage per day/month

---

Machine Learning Algorithms

Logistic Regression

Binary classification (Yes/No, Spam/Not Spam)

- Uses sigmoid function to get probability
- If probability > 0.5 → class 1, else class 0

```python
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
```

Decision Tree

Makes predictions like a flowchart with if-else questions

```python
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
```

Random Forest

Combines many decision trees, uses voting for classification

```python
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
```

K-Nearest Neighbors (KNN)

Predicts based on K closest training examples

- Calculate distances to all points
- Pick K nearest neighbors
- Classification: majority vote
- Regression: average value

K-Means Clustering

Unsupervised algorithm that groups data into K clusters

- Each cluster has a centroid
- Minimizes distance from points to centroids

Linear Regression

Predicts continuous values using formula: $Y = mx + c$

- m = slope
- c = intercept

SQL Essentials

IN / NOT IN

```sql
-- Match multiple values
SELECT name FROM students
WHERE city IN ('Delhi', 'Mumbai', 'Chennai');

-- Exclude multiple values
SELECT name FROM students
WHERE city NOT IN ('Delhi', 'Mumbai');
```

JOIN Types

INNER JOIN - Only matching rows from both tables

```sql
SELECT students.name, subjects.subject_name
FROM students
INNER JOIN subjects ON students.std_id = subjects.std_id;
```

LEFT JOIN - All from left table + matches from right

```sql
SELECT students.name, subjects.subject_name
FROM students
LEFT JOIN subjects ON students.std_id = subjects.std_id;
```

RIGHT JOIN - All from right table + matches from left

FULL JOIN - Everything from both tables

GROUP BY

Groups rows with same values

```sql
SELECT city, COUNT(*)
FROM students
GROUP BY city;
```

HAVING

Filters groups (used after GROUP BY)

```sql
SELECT city, COUNT(*)
FROM students
GROUP BY city
HAVING COUNT(*) > 5;
```

Remember: WHERE filters rows, HAVING filters groups

Database Design - Normalization

Process of organizing data to reduce redundancy and improve consistency.

First Normal Form (1NF)

- No repeating columns
- Only single values per cell
- Must have primary key

Second Normal Form (2NF)

- Must be in 1NF
- No partial dependencies
- Non-key attributes depend on entire primary key

Third Normal Form (3NF)

- Must be in 2NF
- No transitive dependencies

- Every column depends only on primary key

Normal Form	What It Fixes	Key Rule
1NF	Multi-valued data	Single values only
2NF	Partial dependency	Depend on whole key
3NF	Transitive dependency	Only depend on primary key

These notes cover the core concepts I'm learning for building intelligent retail inventory systems with modern tech stack.

Git and GitHub

Git is a version control software that helps developers track changes in their source code. It keeps a complete history of every modification made to a project, including who made the change and when. Git works locally on a computer, so developers can use it without an internet connection. It helps manage multiple versions of a project and allows developers to revert back to earlier versions if something goes wrong.

GitHub is an online platform that hosts Git repositories. It allows developers to store their code on the internet and collaborate with others. GitHub provides many additional features such as issue tracking, pull requests, team collaboration, project boards, and automated workflows using GitHub Actions. While Git manages the code, GitHub helps in sharing and collaboration.

The command git --version is used to check the installed version of Git on a computer.

Git Repository

A Git repository is a storage location where Git stores all the files of a project along with the complete history of changes. It remembers every file, every update, every version, and information about who made each change. Repositories can be local (on your computer) or remote (on platforms like GitHub).

Agentic AI

Agentic AI refers to an artificial intelligence system that can act independently to achieve goals. Unlike simple AI systems that only respond to user queries, agentic AI can observe its environment, plan actions, execute tasks, and adjust its behavior based on feedback. It behaves like a digital agent capable of decision-making.

Agentic AI Workflow

The first step is Goal Definition, where the AI is given a task to achieve. For example, booking a flight or managing inventory.

Next is Environment Perception, where the AI collects information from its environment such as databases, APIs, sensors, or system states. This data helps the AI understand the current situation.

In the Planning stage, the AI analyzes the collected data and decides the best sequence of actions to reach the goal. It may compare multiple options and select the most efficient one.

During Action or Execution, the AI performs the planned steps. This may include sending emails, placing orders, running programs, or interacting with other systems.

Finally, in Monitoring and Feedback, the AI checks whether its action was successful. If the goal is not achieved, it modifies its plan and tries again.

Agents in Inventory Management

An inventory agent can automatically check stock levels, predict when products will go out of stock, send alerts to managers, generate purchase orders, and update demand forecasts regularly. This reduces human effort and improves efficiency.

Types of AI Agents

A Reactive Agent responds instantly to current input without using memory or past experiences. It does not plan ahead. A robot vacuum that changes direction when it hits an obstacle is a good example.

A Goal-Driven Agent works toward a specific goal. It can plan its actions and choose the best path to achieve the goal instead of reacting blindly.

A Multi-Agent System consists of multiple independent agents working together. These agents can communicate, cooperate, or compete to achieve shared or individual goals. Such systems are used in simulations, robotics, and distributed AI applications.

AutoGen Agents

AutoGen frameworks often use multiple agents. A User Proxy Agent represents the user and communicates the user's goals to other agents. An Assistant Agent performs the actual work by planning, reasoning, and executing tasks using tools and APIs.

OpenAI API in Python

The statement from openai import OpenAI imports the OpenAI class from the OpenAI Python library. The line client = OpenAI() creates a client object that connects your Python program to OpenAI's servers. This client is used to send requests and receive responses from AI models.

When using chat completion code, messages are sent to an AI model along with instructions. The system message defines the behavior of the AI, the user message contains the question, and max_tokens controls the length of the response.

Gemini API

The command pip install google-generativeai installs Google's AI library. The genai.configure() function sets the API key. A generative model such as Gemini is then created and used to generate text responses from user prompts.

LLM Text Processing Workflow

Large Language Models do not understand raw text directly. The input text is first broken into small units called tokens. These tokens may be full words, parts of words, or punctuation marks.

Each token is then converted into a numerical representation called an embedding. Embeddings capture the meaning of tokens in mathematical form.

The embeddings pass through transformer layers, where relationships between all tokens are analyzed. The attention mechanism helps the model focus on important words and understand context.

Finally, the model generates output tokens one by one, which are converted back into readable text.

Prompt Engineering

Prompt engineering is the practice of carefully designing the input text given to an AI model so that it produces accurate and useful output. A well-written prompt clearly explains what is required and may include instructions, context, and examples.

Machine Learning Algorithms

Logistic Regression is used for binary classification problems such as spam detection or fraud detection. It uses a sigmoid function to convert values into probabilities.

A Decision Tree makes decisions using a tree-like structure of questions and answers. It is easy to understand and interpret.

A Random Forest combines many decision trees to improve accuracy and reduce overfitting. The final output is based on majority voting or averaging.

K-Nearest Neighbors (KNN) predicts output based on the closest data points in the dataset. It uses distance measures such as Euclidean distance.

K-Means Clustering is an unsupervised algorithm that groups data into clusters based on similarity.

Linear Regression predicts continuous values using a linear relationship between input and output variables.

SQL Concepts

The IN clause is used to match a value against multiple possible values, while NOT IN excludes specific values.

Joins are used to combine rows from multiple tables based on a common column. Different types of joins return different combinations of matched and unmatched records.

GROUP BY groups rows with the same values, and HAVING is used to filter grouped data.

YOLO (You Only Look Once)

YOLO is a real-time object detection algorithm that detects all objects in an image in a single pass. It divides the image into a grid and predicts object class, bounding box, and confidence score for each grid cell. Because it processes the entire image at once, YOLO is extremely fast and suitable for real-time applications such as self-driving cars, surveillance, and robotics.

Python Core Concepts

Python provides many built-in methods for strings, lists, dictionaries, and sets that simplify data manipulation. Python supports both mutable and immutable objects. It uses indentation to define code blocks instead of braces.

Advanced Python concepts include decorators, closures, variable-length arguments, exception handling, memory management, and object-oriented programming features like inheritance, polymorphism, and abstraction.

Software Development Life Cycle (SDLC)

SDLC is a structured approach to software development. It includes requirement analysis, design, development, testing, deployment, and maintenance.

The Waterfall Model follows a linear sequence of phases and is suitable for projects with fixed requirements.

The Agile Model follows an iterative approach with frequent feedback and continuous improvement, making it suitable for changing requirements.

Algorithms

Sorting algorithms such as Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, and Quick Sort are used to arrange data in a specific order. They differ in performance, efficiency, and use cases.

A* (A-Star) Search Algorithm

A* is an **informed search algorithm** used to find the **shortest path** between a start node and

a goal node. It is widely used in **AI, path-finding, and graph traversal** problems.

Working of A* Algorithm

1. Add the **start node** to the OPEN list.

2. Select the node with **minimum f(n)** from OPEN.

3. If it is the **goal node**, stop.

4. Move the node to the **CLOSED list**.

5. Expand the node and calculate **g, h, and f** for its neighbors.

6. Add unexplored neighbors to OPEN.

7. Repeat until goal is found or OPEN is empty.

Heuristic Function (h(n))

A heuristic must be:

• **Admissible** → never overestimates the true cost

• **Consistent** → satisfies triangle inequality

1. Bubble SortIdea

Repeatedly compares adjacent elements and swaps them if they are in the wrong order.

Largest element moves to the end in each pass.

Algorithm

1. Repeat for all elements:

2. Compare adjacent elements.

3. Swap if left > right.

4. Stop when no swaps occur.

Example

Array: 5 1 4 2

Pass 1 → 1 4 2 5

Pass 2 → 1 2 4 5

2. Selection Sort

Idea

Selects the smallest element and places it at the beginning.

Algorithm

1. Find the minimum element.

2. Swap with the first unsorted position.

3. Repeat for remaining array.

Example

64 25 12 22 11

After 1st pass → 11 25 12 22 64

Complexity

• Best / Average / Worst: **$O(n^2)$**

3. Insertion Sort

IdeaBuilds sorted list one element at a time by inserting elements in the correct position.

Algorithm

1. Take next element.

2. Shift larger elements to the right.

3. Insert element at correct position.

Example

8 3 5 2

Stepwise → 3 8 5 2 → 3 5 8 2 → 2 3 5 8

Complexity

• Best: **O(n)**

• Average: **O(n²)**

• Worst: **O(n²)**

4. Merge Sort

Idea

Divide-and-conquer algorithm. Splits array into halves, sorts them, and merges.

Algorithm

1. Divide array into two halves.

2. Recursively sort halves.

3. Merge sorted halves.

Example

38 27 43 3

Split → [38 27] [43 3]

Merge → [3 27 38 43]

Complexity

• Best / Average / Worst: **O(n log n)**

5. Quick SortIdea

Divide-and-conquer algorithm using a **pivot**. Elements smaller than pivot go left, larger go

right.

Algorithm

1. Choose a pivot.

2. Partition array.

3. Recursively apply quick sort.

Example

10 80 30 90 40

Pivot = 40 → [10 30] 40 [80 90]

Complexity

• Best: **O(n log n)**

Different Frameworks under Agentic AI

Agentic AI refers to AI systems that can **perceive, reason, plan, act, and learn autonomously**

to achieve goals. These agents often operate in dynamic environments with feedback loops.

Below are the **major frameworks / approaches used in Agentic AI**, explained simply (exam

friendly):

1. Reactive Agent Framework

• Agents act **only on current perceptions**

• No memory of past states

Example:

Rule-based systems, reflex agents

Use case:

Simple games, obstacle avoidance

2. Deliberative Agent Framework

• Agents **plan before acting**

• Maintain an internal model of the environment

Techniques:• Search algorithms

• Planning (STRIPS, A*)

Use case:

Robotics, route planning

3. Hybrid Agent Framework

• Combines **reactive + deliberative** agents

• Fast reactions + intelligent planning

Use case:

Autonomous vehicles, smart robots

4. Reinforcement Learning–Based Agents

• Agents learn from **reward and punishment**

• Improve decisions over time

Algorithms:

• Q-Learning

• DQN

• Policy Gradient

Use case:

Games, robotics, recommendation systems

5. Multi-Agent Systems (MAS)

• Multiple agents **interact or cooperate**

• Agents may be cooperative or competitive

Examples:

• Swarm intelligence

• Distributed AI

Use case:

Traffic control, stock trading bots

Support Vector Machine (SVM)

Support Vector Machine is a supervised learning algorithm used mainly for classification and regression tasks. It works by finding the best separating hyperplane that divides the data into different classes while maximizing the margin between them. The data points that lie closest to the decision boundary are known as support vectors, and they play a crucial role in defining the final separation line. SVM can work on both linearly and non-linearly separable data. For non-linear problems, it uses the kernel trick to transform input data into a higher-dimensional space where it becomes separable. Common kernel functions include linear, polynomial, RBF (Gaussian), and sigmoid kernels. SVM performs well in high-dimensional feature spaces and is widely

used in applications like text classification, face recognition, bioinformatics, and pattern detection.

Naive Bayes Classifier

Naive Bayes is a probabilistic machine learning algorithm based on Bayes' Theorem. It predicts class membership by calculating the probability of a class given a set of features. The model assumes that all features are independent of one another, which simplifies the computation process, even though real-world data may not strictly follow this assumption. There are different types of Naive Bayes models such as Gaussian Naive Bayes for continuous features, Multinomial Naive Bayes for text and frequency data, and Bernoulli Naive Bayes for binary input variables. Naive Bayes is widely used in spam filtering, sentiment analysis, document classification, recommendation systems, and medical diagnosis because of its simplicity, fast computation, and suitability for large datasets.

Principal Component Analysis (PCA)

Principal Component Analysis is a dimensionality reduction technique used to reduce the number of input features while retaining the most significant information in a dataset. It transforms the original features into a new set of variables called principal components, which capture maximum variance in descending order. The process involves standardizing the data, computing the covariance matrix, extracting eigenvalues and eigenvectors, and selecting principal components based on the amount of variance they contribute. PCA helps in eliminating redundant and correlated features, improves computation efficiency, and enhances visualization in high-dimensional datasets. It is commonly applied in areas like image compression, feature extraction, biological data analysis, and machine learning preprocessing.

Bias–Variance Trade-Off

The bias–variance trade-off describes the balance a model must achieve between learning patterns from training data and maintaining generalizability on unseen data. Bias refers to the error introduced by oversimplifying a model, which may cause underfitting if the model fails to capture important relationships. Variance refers to the model's sensitivity to fluctuations in the training dataset, which may result in overfitting when the model memorizes noise or unnecessary patterns. The ideal model maintains a balance where both bias and variance remain low. Techniques such as cross-

validation, proper model selection, regularization, and sufficient training data help in maintaining this balance and improving model performance.

Model Evaluation Metrics

Model evaluation metrics are used to assess the performance of classification models. Accuracy measures the proportion of correctly predicted values out of total predictions. Precision indicates how many predicted positive values are actually positive, while recall measures the model's ability to correctly identify actual positive values. The F1-score provides a balance between precision and recall, making it useful for imbalanced datasets. A confusion matrix summarizes model performance in terms of true positives, true negatives, false positives, and false negatives. ROC and AUC curves evaluate how well the model distinguishes between different classes by analyzing sensitivity and specificity across various thresholds.

Train–Test–Validation and Cross-Validation

In machine learning, a dataset is typically divided into training, validation, and testing sets to ensure proper learning and performance evaluation. The training set is used to teach the model, the validation set helps in fine-tuning parameters and selecting the best model configuration, and the test set evaluates the final performance on unseen data. Cross-validation, especially K-Fold Cross-Validation, further improves model reliability by splitting the dataset into multiple parts and training the model repeatedly on different combinations. This process ensures that the model does not depend heavily on a single data split and helps in achieving more stable and generalized results.

Overfitting and Underfitting

Overfitting occurs when a model learns the noise and unnecessary patterns in training data instead of learning the actual pattern, leading to poor performance on new data. This usually happens when the model is too complex or trained for too long on limited data. Underfitting, on the other hand, happens when a model is too simple and fails to capture important relationships in the data, resulting in poor performance on both training and testing datasets. To address overfitting, techniques such as regularization, dropout, proper feature selection, and using more training data are applied. To avoid underfitting, more relevant features may be added, and the model complexity can be increased appropriately.

Regularization

Regularization is a technique used to control model complexity and prevent overfitting by adding a penalty term to the loss function. L1 regularization, also known as Lasso, encourages sparsity in the model by reducing some feature weights to zero, which also helps in feature selection. L2 regularization, also known as Ridge, penalizes large weights by distributing them more evenly across features, improving stability. Elastic Net combines both L1 and L2 regularization to balance sparsity and stability. Regularization allows models to generalize better and perform more consistently on unseen datasets.

Artificial Neural Networks (ANN)

Artificial Neural Networks are computational models inspired by the functioning of the human brain. They consist of an input layer, one or more hidden layers, and an output layer. Data flows through interconnected neurons where each connection contains weights and biases that determine the strength of information being passed. Activation functions such as ReLU, sigmoid, and tanh introduce non-linearity to help the network learn complex patterns. During training, the network adjusts weights using backpropagation to minimize error. ANNs are widely used in pattern recognition, image and speech analysis, classification tasks, natural language processing, and predictive modeling.

Convolutional Neural Networks (CNN)

Convolutional Neural Networks are specialized neural networks primarily used for image processing and visual recognition tasks. They consist of convolution layers that extract spatial features from images, pooling layers that reduce dimensionality and preserve important information, and fully connected layers that perform final classification or prediction. CNNs can automatically learn patterns such as edges, shapes, textures, and objects from images without manual feature extraction. They are extensively used in applications such as object detection, face recognition, medical imaging, autonomous driving systems, and real-time computer vision tasks.

RNN, LSTM, and GRU

Recurrent Neural Networks are designed to process sequential and time-dependent data such as text, speech, and time-series signals. They retain information from previous steps using hidden states, allowing the model to understand context. However, traditional RNNs suffer from vanishing gradient problems when dealing with

long sequences. Long Short-Term Memory (LSTM) networks address this issue by using memory cells and gating mechanisms that control information flow, enabling them to retain long-term dependencies. Gated Recurrent Units (GRU) simplify the LSTM structure while providing similar performance with faster computation. These models are widely used in applications like language modeling, speech recognition, translation, text prediction, and financial forecasting.

Transformers

Transformers are modern deep learning architectures that rely on the self-attention mechanism instead of sequential processing. Unlike RNNs, they process entire sequences in parallel, making them faster and more efficient for large-scale language tasks. The self-attention mechanism allows the model to focus on different parts of a sentence based on contextual relevance, enabling better understanding of long-range dependencies. Transformers are the foundation of many advanced language models used today, including chatbots, translation systems, summarization tools, and generative AI applications.

NLP Basics

Natural Language Processing involves techniques used to process and analyze human language data. Core steps include tokenization, where text is broken into words or subwords; stopword removal, which removes commonly used words that add little meaning; and normalization methods such as stemming and lemmatization that convert words to their root forms. Text features can be represented using TF-IDF values or word embeddings that capture semantic meaning in numerical form. These representations are used in tasks such as text classification, sentiment analysis, speech-to-text, chat systems, and information retrieval.

Advanced SQL Concepts

Advanced SQL concepts focus on efficient database organization and structured querying. Database design uses different types of keys such as primary keys, foreign keys, and candidate keys to uniquely identify and relate records. Normalization organizes data across tables to reduce redundancy and maintain consistency, typically following 1NF, 2NF, and 3NF rules. Advanced querying techniques include subqueries, joins, and views, which enable retrieval of complex relationships across tables. Database transactions ensure reliability through commit and rollback operations, and

the ACID properties (Atomicity, Consistency, Isolation, Durability) ensure data integrity during transaction processing.

Git Advanced Workflow

Advanced Git workflow involves creating and managing branches, integrating changes using merge or rebase, and collaborating through pull requests and code reviews on shared repositories. Branching allows developers to work on features independently without affecting the main project. Repositories may include CI/CD automation using GitHub Actions to run tests, deploy software, or perform development workflows. This structured workflow supports version control, teamwork, and collaborative software development practices.

Q-Learning in Reinforcement Learning

Q-Learning is a widely used **model-free reinforcement learning algorithm** that enables an agent to learn optimal actions by interacting with its environment. Instead of relying on a predefined model, the agent improves its behavior through **trial and error**, evaluating the outcomes of its actions.

Think of a system that looks at an apple but incorrectly identifies it as a mango. When it receives feedback — "Wrong, it's an apple" — it learns from the mistake. The next time it sees the apple, it responds correctly. This feedback-driven learning is similar to how Q-Learning operates.

Core Concept

Q-Learning builds a **Q-table** containing Q-values, which estimate how beneficial it is to take a particular action in a specific state. These values are refined as the agent gathers more experience and feedback.

Key Components

1. Q-Values (Action-Values)

Q-values represent the predicted future reward for performing an action in a given state. They are updated over time using the **Temporal Difference (TD) learning rule**.

---

2. Rewards and Episodes

The agent transitions between states by performing actions and receiving rewards. An episode ends when a **terminal state** is reached.

---

3. Temporal Difference (TD) Update Rule

Q-values are updated using:

$$Q(S,A) \leftarrow Q(S,A) + \alpha(R + \gamma Q(S',A') - Q(S,A))$$

Where:

- **S** → current state
- **A** → action taken
- **S′** → next state
- **A′** → best action in next state
- **R** → reward received
- **γ (gamma)** → discount factor for future rewards
- **α (alpha)** → learning rate

---

4. ε-Greedy Policy (Exploration vs Exploitation)

To balance learning and performance:

- **Exploitation (1 – ε probability):** choose the action with the highest Q-value
- **Exploration (ε probability):** choose a random action to discover new possibilities

This helps the agent gradually improve its strategy.

How Q-Learning Works (Step-by-Step)

1. **Agent starts in an initial state (S)**
2. **Chooses an action (A)** using the ε-greedy policy
3. **Environment returns**
   - next state (S′)
   - reward (R)
4. **Q-table is updated** using the TD rule

5. **Policy improves** and the loop repeats across many episodes

   Eventually, the agent learns an **optimal policy** that maximizes reward.

Methods for Computing Q-Values

1. Temporal Difference Learning

Learns directly from experience by comparing current and previous estimates.

2. Bellman Equation

$$Q(s,a) = R(s,a) + \gamma \max_{a} Q(s',a)$$

It recursively estimates the best expected reward for each state-action pair.

What is a Q-Table?

The Q-table acts as memory where:

- **Rows** = states
- **Columns** = actions
- **Entries** = Q-values

As learning progresses, the table reflects the **best actions for each state**.

Implementation Example

A simple Q-Learning setup in a 4×4 grid environment.

Step 1 — Define Environment

```
import numpy as np

import matplotlib.pyplot as plt


n_states = 16

n_actions = 4

goal_state = 15

Q_table = np.zeros((n_states, n_actions))
```

Step 2 — Hyperparameters

learning_rate = 0.8

discount_factor = 0.95

exploration_prob = 0.2

epochs = 1000

Step 3 — Action Mapping

- 0 = Left
- 1 = Right
- 2 = Up
- 3 = Down

Search Algorithms in AI

Depth-First Search (DFS)

DFS explores a graph by moving **as deep as possible along one branch** before backtracking.

Characteristics:

- Uses a **stack**
- Not guaranteed to find the shortest path
- Supports **backtracking search**, which saves memory

Depth-Limited Search (DLS)

A variation of DFS that sets a **maximum search depth** to avoid infinite exploration.

Steps:

1. Start with a depth limit
2. Explore nodes
3. Stop expanding when depth exceeds limit
4. Backtrack if goal not found

Uniform Cost Search (UCS)

An optimal search algorithm that expands the node with the **lowest cumulative path cost**, similar to Dijkstra's algorithm.

Key features:

- Uses a **priority queue**
- Guarantees **minimum-cost path**
- Stops when the goal node is expanded

Bidirectional Search

Runs two searches simultaneously:

- forward from start node
- backward from goal node

Search terminates when both meet.

Benefits:

- Faster than single-direction search
- Best used when start and goal are well-defined

Beam Search

A heuristic search that expands only the **top-W most promising nodes** at each level.

Properties:

- Memory-efficient
- Not always optimal (may discard good paths)
- Useful in constrained environments

Gaussian Mixture Model (GMM)

A GMM assumes data comes from multiple Gaussian distributions, each with its own:

- mean (μ_k)
- covariance (Σ_k)

- mixing weight (π_k)

  Parameters are learned using the **Expectation-Maximization (EM) algorithm**.

Backpropagation in Neural Networks

Backpropagation is used to train neural networks by propagating errors backward and updating weights using gradients.

Why it matters:

- Efficient weight optimization
- Supports deep networks
- Automates learning via gradient descent

  Two phases:

1. **Forward pass** → compute predictions
2. **Backward pass** → compute error + update weights

   Loss is often measured using **Mean Squared Error (MSE)**.

   Activation functions (ReLU, Sigmoid, Softmax) play a crucial role in gradient calculation and learning.