IIIT Dharwad Summer Internship Report

Domain: AGENTIC AI

Name: KUMAVATH SAICHARITHA

Institution Name: NIT Raipur

Internship Period: 12/5/25 -12/6/25

Mentor: Dr.MALAY KUMAR Sir, IIIT Dharwad

INTRODUCTION:

The domain of Agentic AI focuses on building autonomous systems that demonstrate purposeful, goal-directed behaviors, capable of interacting with environments and adapting over time. During my summer internship at IIIT Dharwad, I worked on two projects that explore real-world applications/articles of Agentic AI:

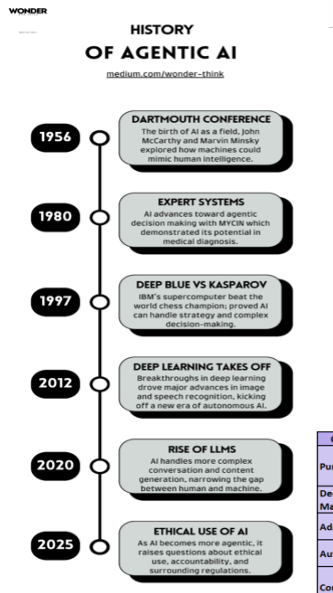
1. An autonomous AI planning system for complex decision-making
2. A cybersecurity assistant that uses large language models for vulnerability assessment

Project 1: Agentic AI – Autonomous Intelligence for Complex Tasks(ppt)

OBJECTIVE:

To explore the structure and behavior of autonomous intelligent agents capable of making decisions in dynamic, partially observable environments.

Slide 1:



"Agentic" comes from the word *agent*, which in AI refers to an entity that can **observe its environment, make decisions, and act** in that environment.

DEFINITION:

- **Agentic AI** refers to **artificial intelligence systems that can act autonomously** to pursue goals, make decisions, and take actions—often over extended periods—**without needing constant human oversight**.
- **Agentic AI** systems are designed to have a degree of **initiative, planning, and goal-seeking behavior**.
- These systems can **break down high-level goals into sub-tasks**, adapt their actions, and **interact with tools, humans, or other AIs** to achieve their objectives.

Examples:

An **AI personal assistant** that can plan a trip: book flights, find hotels, schedule meetings, and update plans if delays happen.
AutoGPT: experimental frameworks that allow AI to autonomously create and complete tasks toward a given goal.

ROLE IN AI ECOSYTEM: Agentic AI plays a **distinctive and essential role** in the broader AI landscape, occupying a **middle ground** between:
Narrow AI (limited-task systems like chatbots or image classifiers)
Artificial General Intelligence (AGI) (human-level reasoning, still hypothetical)

Criteria	Traditional AI	Agentic AI (Autonomous AI)
Purpose	Performs specific tasks based on predefined rules or training	Operates autonomously, making independent decisions and adapting to changes
Decision-Making	Follows predefined logic or supervised learning patterns	Makes self-directed decisions based on evolving goals
Adaptability	Limited adaptability; requires retraining for new scenarios	Continuously learns, adapts, and refines its approach dynamically
Autonomy	Requires human oversight or intervention	Functions with minimal to no human intervention
Complexity	Can be simple (rule-based) or complex (deep learning)	Requires advanced cognitive architectures for reasoning and planning
Examples	Chatbots, recommendation systems, fraud detection	AI agents in self-driving cars, autonomous trading bots, robotics

Core Characteristics of Agentic AI :

- **Goal-Oriented**: Agentic AI focuses on achieving specific objectives by planning and executing multiple steps independently.
- **Autonomous**: It operates without constant human supervision, making decisions on its own.
- **Adaptive**: The system can adjust to new information or changing environments in real time.
- **Reasoning Capable**: It uses logical reasoning to break down tasks and solve problems effectively.
- **Proactive and Interactive**: Agentic AI takes initiative and can interact with tools, systems, or humans to fulfill complex tasks.

Slide 2:

METHODOLOGIES IN AGENTIC AI DEVELOPMENT:

Step 1: Choose an Architectural Approach
Select a design based on task complexity and agent interaction:
- **Multi-Agent Systems (MAS):** Divide tasks among agents for collaboration or competition.
- **Hierarchical Reinforcement Learning (HRL):** Use high-level agents for planning, low-level for execution.
- **Goal-Oriented Modular Architectures:** Use modular components specialized for sub-tasks, enhancing scalability.

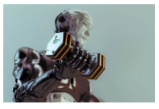
Step 2: Select Learning Paradigms
Pick learning methods based on task type:
- **Supervised Learning:** For labeled data and prediction tasks.
- **Unsupervised Learning:** For pattern discovery in unlabeled data.
- **Reinforcement Learning:** For goal-oriented behavior via rewards.

Step 3: Integrate Advanced Methodologies
Equip agents with essential cognitive and operational capabilities:
- **Reasoning & Planning:** To manage complex, dynamic tasks.
- **Tool Use:** To access APIs or external functions.
- **Memory Mechanisms:** For episodic and semantic recall.
- **Retrieval-Augmented Generation (RAG):** For real-time external knowledge retrieval.
- **Instruction Fine-Tuning:** For executing precise, multi-step tasks.

Step 4: Apply Training Techniques
Use robust training environments and strategies:
- **Simulation-Based Training:** Safe, scalable environments for skill acquisition.
- **Curriculum Learning:** Gradually increase task complexity.
- **Multi-Task Learning:** Enable learning across diverse tasks for better generalization.

Step 5: Conduct Evaluation
Evaluate agent performance using these key metrics:
- **Task Success Rate**
- **Adaptability**
- **Resource Efficiency**
- **Long-Term Goal Achievement**

TOOLS AND FRAMEWORKS:

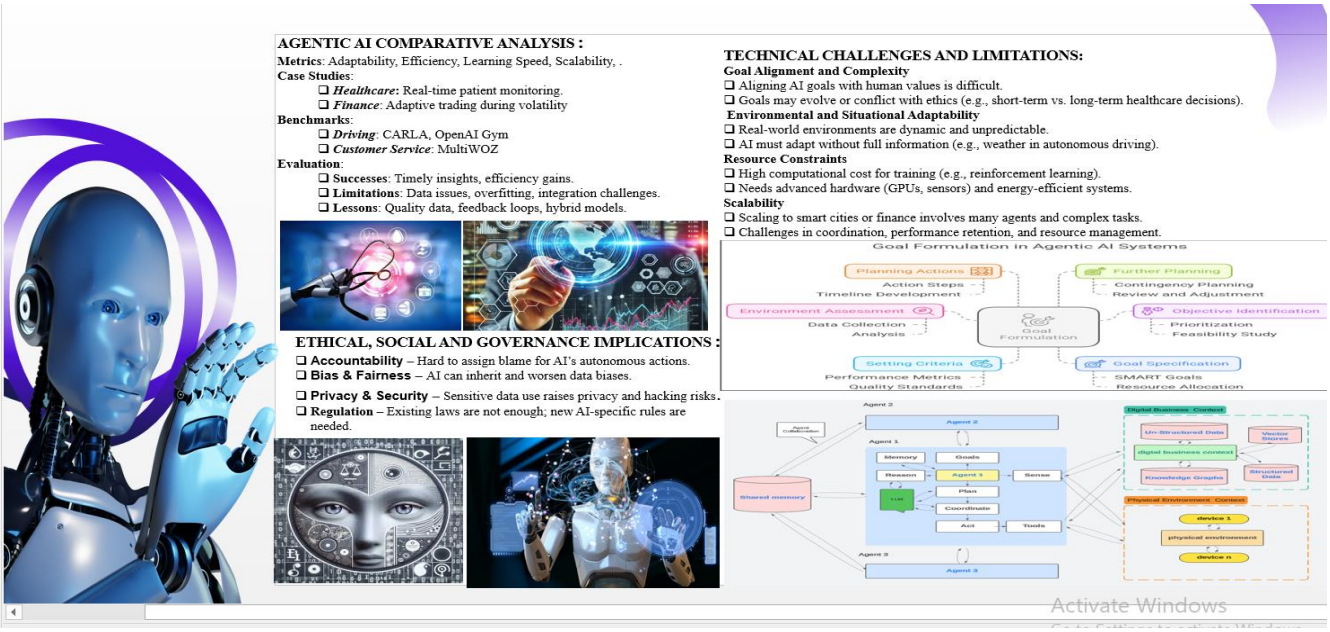



OpenAI Gym – For building and testing reinforcement learning environments.
PyMARL – For creating and testing multi-agent reinforcement learning systems.
Unity ML-Agents – For training agents in 3D game-like simulations.
TensorFlow Agents (TF-Agents) – For developing RL agents using TensorFlow.

TRAINING AND EVALUATION :

Agentic AI training involves methods like simulation-based, curriculum, and multi-task learning to help agents learn safely and progressively.
Evaluation focuses on success rate, adaptability, efficiency, and long-term performance to ensure robust intelligent agent behavior.

Slide 3:



Slide 4:

CURRENT FRAMEWORKS FOR SAFE AND ACCOUNTABLE AGENTIC AI:

A. Safety Protocols

Goal Safety – Guides AI to pursue ethical/legal goals.
Fail-safes – Stops AI when risky behavior is detected.

B. Monitoring & Control

Real-time Monitoring – Tracks AI actions for intervention.
HITL & HOTL – Human oversight during or after decisions.

C. Transparency Mechanisms

Explainable AI (XAI) – Makes AI decisions understandable.
Audit Trails – Logs decisions for review.

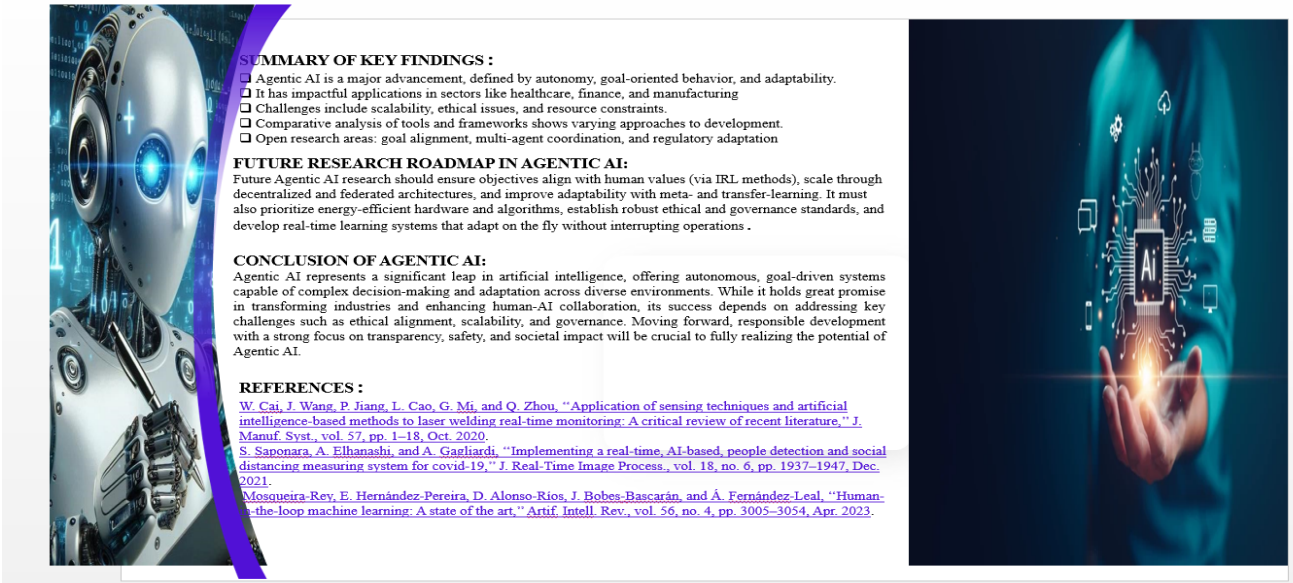
D. Governance Case Studies

Microsoft – AI principles & ethics committee.
Google – Model Cards & responsible AI use.

ETHICAL FRAMEWORKS AND GLOBAL STANDARDS:

Component	Description
Transparency and Explainability	Ensure AI decisions are interpretable and can be scrutinized by users and regulators
Accountability and Responsibility	Define and trace responsibility across stakeholders in the AI lifecycle
Privacy and Security Standards	Enforce data privacy and security protocols to protect user information
Global Governance	Develop international regulatory bodies to harmonize AI standards across borders
Dynamic Regulation	Create adaptable frameworks that evolve with technological advancements in AI

Slide 5:



KEY HIGHLIGHTS:

Created a 5-slide PowerPoint presentation that outlines:

1.Definition and principles of agentic AI

2.Role of decision-making, planning, and self-correction in agents

3.Use cases: robotics, simulation agents, digital assistants

4.Proposed a conceptual architecture for an agent that performs autonomous planning using feedback from the environment.

5.Discussed future directions such as multi-agent coordination and adaptive learning agents.

SKILLS USED:

1. Agentic AI theory
2. System architecture modeling
3. AI planning principles

Project 2: ChatNVD – Advancing Cybersecurity Vulnerability Assessment with Large Language Models(Implementation)

OBJECTIVE:

To build an AI assistant that helps in understanding and evaluating software vulnerabilities using LLMs (Large Language Models) on the National Vulnerability Database (NVD).

KEY HIGHLIGHTS:

Developed a full pipeline in Google Colab using Python:

1.Parsed CVE entries from NVD JSON files

2. Generated TF-IDF embeddings for semantic matching

3.Queried GPT-4o mini, LLaMA 3, and Gemini 1.5 Pro for CVE explanations

4.Computed accuracy and error rates for 125 generated CVE-based questions

5.Visualized results using matplotlib and seaborn (bar charts, box plots)

6.Focused on optimizing cost, time, and performance through TF-IDF-based filtering before LLM query.

7.Deployed backend via FastAPI and frontend in React, hosted on AWS EC2 instance.

SKILLS USED:

1.Python (pandas, sklearn, matplotlib)

2. LLM APIs (OpenAI, Gemini, LLaMA 3)

3.NLP (TF-IDF, embedding-based retrieval)

4.Web Development (FastAPI, React)

CODE:

Step 1:

```
# Install required packages
!pip install openai

import openai
import os
from google.colab import files
import zipfile
import gzip
import json

# Set your OpenAI API key
openai.api_key =  "sk-proj-hGeBHh79A00Ed3bvyI_glgvyq_dCT4ny--
e7XqWN_zSAim3_TI7qfXoovm7qL05azTJSJ4KEDAT3BlbkFJUnUdkFPKAjA9kLi4959I
gx7GjZ8OA5URmfU9HM3MZh4-SYLfHtl7hSraPdbhJhtzbiHQJ3ZQsA"  # Replace
with your actual API key

def upload_files():
    """Upload files to Colab environment"""
    uploaded = files.upload()
    return uploaded

def process_file(file_path):
    """Process different file types"""
```

```
    if file_path.endswith('.json'):
        with open(file_path, 'r') as f:
            return json.load(f)
    elif file_path.endswith('.zip'):
        with zipfile.ZipFile(file_path, 'r') as zip_ref:
            zip_ref.extractall('extracted_files')
            return "Extracted ZIP contents to 'extracted_files'
directory"
    elif file_path.endswith('.gz'):
        with gzip.open(file_path, 'rb') as f:
            content = f.read()
            return content.decode('utf-8')  # Assuming text content
    else:
        return "Unsupported file type"

def query_gpt4(prompt, model="gpt-4"):
    """Query GPT-4 with a prompt"""
    response = openai.ChatCompletion.create(
        model=model,
        messages=[{"role": "user", "content": prompt}]
    )
    return response.choices[0].message.content

# Main execution
if __name__ == "__main__":
    # 1. Upload files
    print("Please upload your files (JSON, ZIP, or GZ):")
    uploaded_files = upload_files()

    # 2. Process files
    for filename in uploaded_files:
        print(f"\nProcessing {filename}...")
        result = process_file(filename)
        print(f"Result: {result[:200]}...")  # Print first 200 chars

    # 3. Example GPT-4 query
    print("\nQuerying GPT-4...")
    response = query_gpt4("Explain how to analyze JSON data in
Python")
    print("GPT-4 Response:", response)
```

output:

```
Requirement already satisfied: openai in
/usr/local/lib/python3.11/dist-packages (0.28.0)
Requirement already satisfied: requests>=2.20 in
/usr/local/lib/python3.11/dist-packages (from openai) (2.32.3)
```

```
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-
packages (from openai) (4.67.1)
Requirement already satisfied: aiohttp in
/usr/local/lib/python3.11/dist-packages (from openai) (3.11.15)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.11/dist-packages (from requests>=2.20->openai)
(3.4.2)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.11/dist-packages (from requests>=2.20->openai)
(3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.11/dist-packages (from requests>=2.20->openai)
(2.4.0)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.11/dist-packages (from requests>=2.20->openai)
(2025.4.26)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in
/usr/local/lib/python3.11/dist-packages (from aiohttp->openai) (2.6.1)
Requirement already satisfied: aiosignal>=1.1.2 in
/usr/local/lib/python3.11/dist-packages (from aiohttp->openai) (1.3.2)
Requirement already satisfied: attrs>=17.3.0 in
/usr/local/lib/python3.11/dist-packages (from aiohttp->openai) (25.3.0)
Requirement already satisfied: frozenlist>=1.1.1 in
/usr/local/lib/python3.11/dist-packages (from aiohttp->openai) (1.6.0)
Requirement already satisfied: multidict<7.0,>=4.5 in
/usr/local/lib/python3.11/dist-packages (from aiohttp->openai) (6.4.4)
Requirement already satisfied: propcache>=0.2.0 in
/usr/local/lib/python3.11/dist-packages (from aiohttp->openai) (0.3.1)
Requirement already satisfied: yarl<2.0,>=1.17.0 in
/usr/local/lib/python3.11/dist-packages (from aiohttp->openai) (1.20.0)
Please upload your files (JSON, ZIP, or GZ):
```
Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
```
Saving nvdcve-1.1-2024.json (1).zip to nvdcve-1.1-2024.json (1) (1).zip

Processing nvdcve-1.1-2024.json (1) (1).zip...
Result: Extracted ZIP contents to 'extracted_files' directory...

Querying GPT-4...
GPT-4 Response: Analyzing JSON data in Python involves multiple steps.

Step 1: Import the necessary libraries

You first need to import the necessary Python libraries which are json
and pandas.

```python
import json
import pandas as pd
```

Step 2: Load your JSON data

You can load the JSON data either from a file or a JSON string. Here is
how you can load JSON data from a string.

```python
```
```

```
data = '''{
    "name" : "Mike",
    "age" : 25,
    "gender" : "male"
}'''

data = json.loads(data)
```

If you want to load JSON data from a file, you can do it like this.

```python
with open('data.json') as f:
    data = json.load(f)
```

Step 3: Analyze the JSON data

Now that you have loaded the JSON data, you can analyze it. One of the common ways to analyze JSON data is by converting it to a pandas DataFrame and then performing typical data analysis operations on it.

```python
df = pd.DataFrame(data)
```

You can then use typical pandas operations to analyze the data. For example, if you want to get the data of all males, you can do something like this.

```python
males = df[df['gender'] == 'male']
```

This will give you a new DataFrame with the data of all males. You can perform similar operations depending on what you are looking for in the data.

Remember, you need to have a solid understanding of Python's pandas library to effectively analyze the JSON data once it's converted to a DataFrame.

Step2:

```python
questions_batches = [
    ["Q1", "Q2", ..., "Q25"],  # Batch 1
    ["Q26", ..., "Q50"],       # Batch 2
    ...
]

ground_truth_batches = [
    ["A1", "A2", ..., "A25"],  # Ground truth for batch 1
    ...
]
```

```
# Responses from each model
responses = {
    "GPT 4o mini": [
        ["A1", "A2", ..., "A25"],   # Batch 1
        ...
    ],
    "Gemini 1.5 Pro": [
        ["A1", "A2", ..., "Wrong"],  # Batch 1
        ...
    ],
    "Llama 3": [
        ["Wrong", "A2", ..., "A25"],  # Batch 1
        ...
    ]
}
```

Step 3:

```
def evaluate_model_batches(responses, ground_truth_batches):
    results = {}
    for model, model_batches in responses.items():
        model_results = []
        for batch_pred, batch_true in zip(model_batches,
ground_truth_batches):
            correct = sum(p.strip().lower() == t.strip().lower() for
p, t in zip(batch_pred, batch_true))
            wrong = len(batch_true) - correct
            model_results.append([correct, wrong])
        results[model] = model_results
    return results
```

step 4:

```
results = evaluate_model_batches(responses, ground_truth_batches)
```

step 5:

```
# Sample ground truth and responses
ground_truth_batches = [
    ["a", "b", "c", "d", "e"],
    ["a", "b", "c", "d", "e"],
    ["a", "b", "c", "d", "e"],
    ["a", "b", "c", "d", "e"],
    ["a", "b", "c", "d", "e"]
]
```

```
responses = {
    "GPT 4o mini": [
        ["a", "b", "c", "d", "e"], ["a", "b", "c", "d", "e"],
        ["a", "b", "c", "d", "e"], ["a", "b", "c", "d", "e"], ["a",
"b", "c", "d", "e"]
    ],
    "Gemini 1.5 Pro": [
        ["a", "b", "x", "d", "e"], ["a", "b", "x", "x", "e"],
        ["a", "x", "x", "d", "e"], ["a", "b", "x", "x", "e"], ["x",
"x", "x", "x", "x"]
    ],
    "Llama 3": [
        ["a", "x", "x", "d", "e"], ["a", "b", "c", "d", "e"],
        ["a", "b", "c", "d", "e"], ["a", "b", "c", "d", "e"], ["x",
"x", "x", "x", "x"]
    ]
}

# Evaluation function
def evaluate_model_batches(responses, ground_truth_batches):
    results = {}
    for model, model_batches in responses.items():
        model_results = []
        for batch_pred, batch_true in zip(model_batches,
ground_truth_batches):
            correct = sum(str(p).strip().lower() ==
str(t).strip().lower() for p, t in zip(batch_pred, batch_true))
            wrong = len(batch_true) - correct
            model_results.append([correct, wrong])
        results[model] = model_results
    return results

# Evaluate
results = evaluate_model_batches(responses, ground_truth_batches)
print(results)
```

output:

```
{'GPT 4o mini': [[5, 0], [5, 0], [5, 0], [5, 0], [5, 0]], 'Gemini 1.5
Pro': [[4, 1], [3, 2], [3, 2], [3, 2], [0, 5]], 'Llama 3': [[3, 2], [5,
0], [5, 0], [5, 0], [0, 5]]}
```

Step 6:

```
# Convert to accuracy and error rate
def compute_metrics(results):
```

```
    accuracy_error = {}
    for model, batches in results.items():
        batch_metrics = []
        for correct, wrong in batches:
            total = correct + wrong
            acc = correct / total
            err = wrong / total
            batch_metrics.append((acc, err))
        accuracy_error[model] = batch_metrics
    return accuracy_error

metrics = compute_metrics(results)
```

step 7:

```
import matplotlib.pyplot as plt
import numpy as np

# Aggregate total accuracy/error
combined_accuracy = []
combined_error = []
models = list(metrics.keys())

for model in models:
    acc = np.mean([a for a, _ in metrics[model]])
    err = np.mean([e for _, e in metrics[model]])
    combined_accuracy.append(acc)
    combined_error.append(err)

# Bar plot
x = np.arange(len(models))
width = 0.35

fig, ax = plt.subplots()
bars1 = ax.bar(x - width/2, combined_accuracy, width,
label='Accuracy')
bars2 = ax.bar(x + width/2, combined_error, width, label='Error
Rate')

ax.set_ylabel('Values')
ax.set_title('Combined Accuracy and Error Rate of all batches for 3
LLMs')
ax.set_xticks(x)
ax.set_xticklabels(models)
```

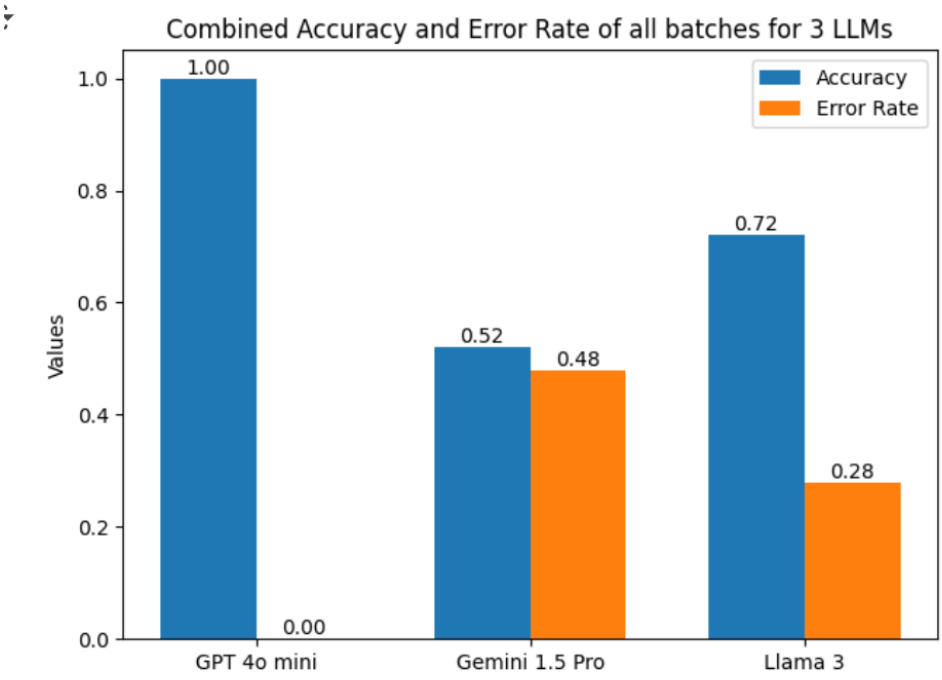
```
ax.legend()

# Add text labels
for bar in bars1:
    height = bar.get_height()
    ax.text(bar.get_x() + bar.get_width()/2.0, height,
f'{height:.2f}', ha='center', va='bottom')

for bar in bars2:
    height = bar.get_height()
    ax.text(bar.get_x() + bar.get_width()/2.0, height,
f'{height:.2f}', ha='center', va='bottom')

plt.ylim(0, 1.05)
plt.tight_layout()
plt.show()
```

output:



Step 8:

```
# Plot batch-wise accuracy/error for each model
fig, axes = plt.subplots(3, 5, figsize=(20, 8))
```

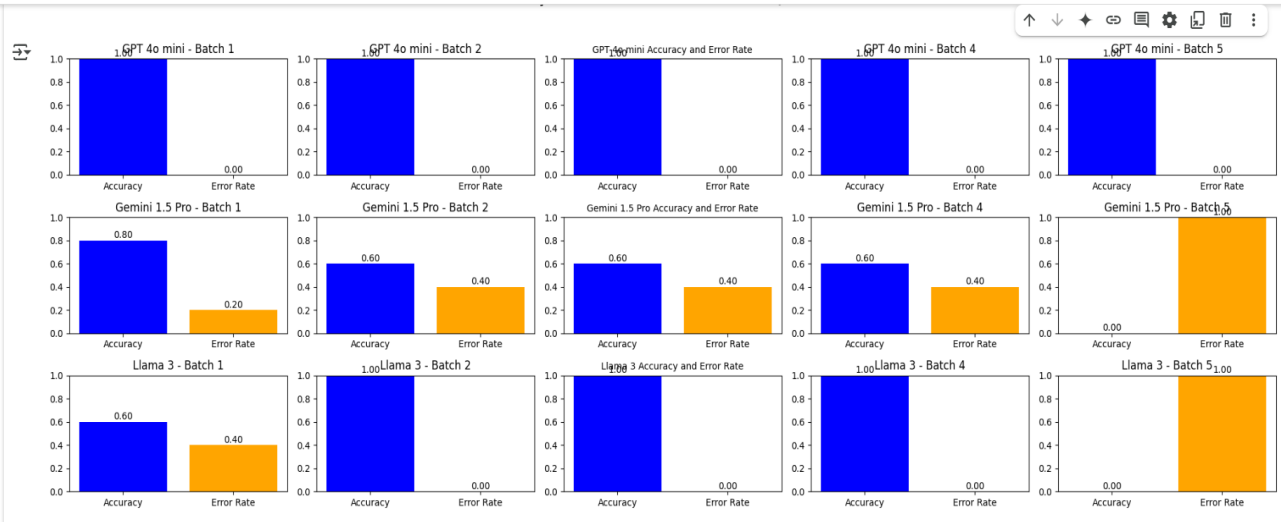
```
fig.suptitle("Accuracy and Error Rate for 5 Batches of Questions",
fontsize=16)

for i, model in enumerate(models):
    for j in range(5):
        acc, err = metrics[model][j]
        ax = axes[i, j]
        ax.bar(['Accuracy', 'Error Rate'], [acc, err],
color=['blue', 'orange'])
        ax.set_ylim(0, 1)
        ax.set_title(f"{model} - Batch {j+1}")
        for k, v in enumerate([acc, err]):
            ax.text(k, v + 0.02, f"{v:.2f}", ha='center')

# Set model labels as figure captions
axes[0, 2].set_title("GPT 4o mini Accuracy and Error Rate",
fontsize=10)
axes[1, 2].set_title("Gemini 1.5 Pro Accuracy and Error Rate",
fontsize=10)
axes[2, 2].set_title("Llama 3 Accuracy and Error Rate", fontsize=10)

plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```

output:



OUTCOMES:

1.Understood the core concepts and architectures behind agentic systems.

2.Gained hands-on experience integrating LLMs for cybersecurity applications.

3. Learned practical skills in NLP, backend APIs, frontend development, and cloud deployment.
4.Evaluated the performance of LLMs in real-world security scenarios.

CONCLUSION:

This internship significantly contributed to my understanding of Agentic AI, especially in the context of autonomy and cybersecurity assistance. The combination of theoretical exploration and practical implementation has strengthened my interest in the field and has equipped me with tools for future research and development.

REFERENCES:

ARTICLES:

- Agentic AI – Autonomous Intelligence for Complex Tasks
- ChatNVD – Advancing Cybersecurity Vulnerability Assessment with Large Language Models