

In [1]:

```
# Colab Setup
!pip install -q \
    langchain \
    langchain-community \
    langchain-huggingface \
    langgraph \
    chromadb \
    sentence-transformers \
    transformers \
    accelerate \
    pypdf \
    fastapi \
    uvicorn \
    requests
```

```
[2K   [90m-----[0m [32m52.0/52.0 kB[0m [31m3.0 MB/s[0m eta [36m0:00:00[0m
[2K   [90m-----[0m [32m2.5/2.5 MB[0m [31m48.6 MB/s[0m eta [36m0:00:00[0m
[2K   [90m-----[0m [32m21.1/21.1 MB[0m [31m78.2 MB/s[0m eta [36m0:00:00[0m
[2K   [90m-----[0m [32m329.1/329.1 kB[0m [31m17.6 MB/s[0m eta [36m0:00:00[0m
[2K   [90m-----[0m [32m64.7/64.7 kB[0m [31m3.8 MB/s[0m eta [36m0:00:00[0m
[2K   [90m-----[0m [32m278.2/278.2 kB[0m [31m18.4 MB/s[0m eta [36m0:00:00[0m
[2K   [90m-----[0m [32m2.0/2.0 MB[0m [31m53.9 MB/s[0m eta [36m0:00:00[0m
[2K   [90m-----[0m [32m1.0/1.0 MB[0m [31m41.6 MB/s[0m eta [36m0:00:00[0m
[2K   [90m-----[0m [32m17.4/17.4 MB[0m [31m79.1 MB/s[0m eta [36m0:00:00[0m
[2K   [90m-----[0m [32m72.5/72.5 kB[0m [31m5.3 MB/s[0m eta [36m0:00:00[0m
[2K   [90m-----[0m [32m132.6/132.6 kB[0m [31m6.1 MB/s[0m eta [36m0:00:00[0m
[2K   [90m-----[0m [32m66.4/66.4 kB[0m [31m4.0 MB/s[0m eta [36m0:00:00[0m
[2K   [90m-----[0m [32m220.0/220.0 kB[0m [31m18.3 MB/s[0m eta [36m0:00:00[0m
[2K   [90m-----[0m [32m105.4/105.4 kB[0m [31m7.9 MB/s[0m eta [36m0:00:00[0m
[2K   [90m-----[0m [32m71.6/71.6 kB[0m [31m7.4 MB/s[0m eta [36m0:00:00[0m
[2K   [90m-----[0m [32m60.6/60.6 kB[0m [31m4.8 MB/s[0m eta [36m0:00:00[0m
[2K   [90m-----[0m [32m51.0/51.0 kB[0m [31m3.1 MB/s[0m eta [36m0:00:00[0m
[2K   [90m-----[0m [32m46.0/46.0 kB[0m [31m2.9 MB/s[0m eta [36m0:00:00[0m
[2K   [90m-----[0m [32m86.8/86.8 kB[0m [31m8.5 MB/s[0m eta [36m0:00:00[0m
[?25h[31mERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behavior
google-colab 1.0.0 requires requests==2.32.4, but you have requests 2.32.5 which is incompatible.
google-adk 1.21.0 requires opentelemetry-api<=1.37.0,>=1.37.0, but you have opentelemetry-api 1.39.1 which is incompatible.
google-adk 1.21.0 requires opentelemetry-sdk<=1.37.0,>=1.37.0, but you have opentelemetry-sdk 1.39.1 which is incompatible.
opentelemetry-exporter-gcp-logging 1.11.0a0 requires opentelemetry-sdk<1.39.0,>=1.35.0, but you have opentelemetry-sdk 1.39.1 wh
opentelemetry-exporter-otlp-proto-http 1.37.0 requires opentelemetry-exporter-otlp-proto-common==1.37.0, but you have openteleme
opentelemetry-exporter-otlp-proto-http 1.37.0 requires opentelemetry-proto==1.37.0, but you have opentelemetry-proto 1.39.1 whic
opentelemetry-exporter-otlp-proto-http 1.37.0 requires opentelemetry-sdk==1.37.0, but you have opentelemetry-sdk 1.39.1 which is
[0m
```

In [2]:

```
!pip install pdfplumber
```

```
Collecting pdfplumber
  Downloading pdfplumber-0.11.9-py3-none-any.whl.metadata (43 kB)
[?25l   [90m-----[0m [32m0.0/43.6 kB[0m [31m?/[0m eta [36m-:--:--[0m[2K   [90m-----[0m
[?25hCollecting pdfminer.six==20251230 (from pdfplumber)
  Downloading pdfminer_six-20251230-py3-none-any.whl.metadata (4.3 kB)
Requirement already satisfied: Pillow>=9.1 in /usr/local/lib/python3.12/dist-packages (from pdfplumber) (11.3.0)
Collecting pypdfium2>=4.18.0 (from pdfplumber)
  Downloading pypdfium2-5.3.0-py3-none-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (67 kB)
[2K   [90m-----[0m [32m67.8/67.8 kB[0m [31m3.8 MB/s[0m eta [36m0:00:00[0m
[?25hRequirement already satisfied: charset-normalizer>=2.0.0 in /usr/local/lib/python3.12/dist-packages (from pdfminer.six==20251230) (3.4.0)
Requirement already satisfied: cryptography>=36.0.0 in /usr/local/lib/python3.12/dist-packages (from pdfminer.six==20251230->pdfminer.six) (42.0.8)
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.12/dist-packages (from cryptography>=36.0.0->pdfminer.six==20251230) (1.17.1)
Requirement already satisfied: pycparser in /usr/local/lib/python3.12/dist-packages (from cffi>=1.12->cryptography>=36.0.0->pdfminer.six==20251230) (2.0)
Downloading pdfplumber-0.11.9-py3-none-any.whl (60 kB)
[2K   [90m-----[0m [32m60.0/60.0 kB[0m [31m4.1 MB/s[0m eta [36m0:00:00[0m
[?25hDownloading pdfminer_six-20251230-py3-none-any.whl (6.6 MB)
[2K   [90m-----[0m [32m6.6/6.6 MB[0m [31m62.4 MB/s[0m eta [36m0:00:00[0m
[?25hDownloading pypdfium2-5.3.0-py3-none-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (3.0 MB)
[2K   [90m-----[0m [32m3.0/3.0 MB[0m [31m37.0 MB/s[0m eta [36m0:00:00[0m
[?25hInstalling collected packages: pypdfium2, pdfminer.six, pdfplumber
Successfully installed pdfminer.six-20251230 pdfplumber-0.11.9 pypdfium2-5.3.0
```

Data Preprocessing

In [3]:

```
import pdfplumber
from pathlib import Path

PDF_PATH = "/content/EAST WEST UNIVERSITY DISCIPLINARY.pdf"

def extract_pdf_text(pdf_path):
    pages = []
    with pdfplumber.open(pdf_path) as pdf:
        for i, page in enumerate(pdf.pages):
            text = page.extract_text()
            if text:
                pages.append({
```

```

        "page_number": i + 1,
        "text": text.strip()
    })
    return pages

raw_pages = extract_pdf_text(PDF_PATH)

print(f"Extracted {len(raw_pages)} pages")

```

Extracted 18 pages

```

In [5]: import re

def clean_text(text):
    text = re.sub(r"Page\s+\d+\s+of\s+\d+", "", text)
    text = re.sub(r"Tuesday,\s+\w+\s+\d{1,2},\s+\d{4}", "", text)
    text = re.sub(r"\n{2,}", "\n", text)
    return text.strip()

for page in raw_pages:
    page["text"] = clean_text(page["text"])

```

```

In [6]: full_text = "\n\n".join(page["text"] for page in raw_pages)

```

```

In [7]: SECTION_PATTERN = re.compile(
    r"\n(?:\d+\.\s+[A-Z][A-Z\s\-\&]+)",
    re.MULTILINE
)

raw_sections = SECTION_PATTERN.split("\n" + full_text)

```

```

In [9]: def parse_section(section_text):
    lines = section_text.strip().split("\n", 1)

    header = lines[0].strip()
    body = lines[1].strip() if len(lines) > 1 else ""

    match = re.match(r"(\d+)\.\s+(.*)", header)
    if not match:
        return None

    section_number = match.group(1)
    section_title = match.group(2)

    return {
        "section_number": section_number,
        "section_title": section_title,
        "content": body
    }

sections = []
for sec in raw_sections:
    parsed = parse_section(sec)
    if parsed:
        sections.append(parsed)

print(f"Parsed {len(sections)} sections")

```

Parsed 22 sections

```

In [10]: CLAUSE_PATTERN = re.compile(r"(?=\d+\.\d+(?:\.\d+)*)"

def split_clauses(section):
    # Safely get content
    text = section.get("content", "").strip()

    # If no body text, return section with empty clauses
    if not text:

```

```

        section["clauses"] = []
        section.pop("content", None)
        return section

chunks = CLAUSE_PATTERN.split(text)

clauses = []
for chunk in chunks:
    chunk = chunk.strip()
    if not chunk:
        continue

    match = re.match(r"(\d+\.\d+(?:\.\d+)*)\s+(.*)", chunk, re.DOTALL)
    if match:
        clauses.append({
            "clause_id": match.group(1),
            "text": match.group(2).strip()
        })

section["clauses"] = clauses
section.pop("content", None)
return section

```

```
In [11]: structured_sections = [split_clauses(sec) for sec in sections]
```

```
In [12]: import json

final_json = {
    "metadata": {
        "title": "The East West University Disciplinary Code for Students, 2011",
        "source": Path(PDF_PATH).name,
        "total_pages": len(raw_pages)
    },
    "sections": structured_sections
}

with open("ewu_disciplinary_code.json", "w", encoding="utf-8") as f:
    json.dump(final_json, f, indent=2, ensure_ascii=False)

print("JSON saved as ewu_disciplinary_code.json")

```

JSON saved as ewu_disciplinary_code.json

Chunking and Embedding

```
In [13]: from langchain_core.documents import Document
import json

with open("ewu_disciplinary_code.json", "r", encoding="utf-8") as f:
    ewu_data = json.load(f)

documents = []

for section in ewu_data["sections"]:
    section_no = section["section_number"]
    section_title = section["section_title"]

    for clause in section.get("clauses", []):
        documents.append(
            Document(
                page_content=clause["text"],
                metadata={
                    "section": section_no,
                    "section_title": section_title,
                    "clause_id": clause["clause_id"],
                    "source": "EWU Disciplinary Code"
                }
            )
        )

```

```
)

print(f"Total documents created: {len(documents)}")
```

Total documents created: 140

```
In [14]: from langchain_text_splitters import RecursiveCharacterTextSplitter

text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=400,
    chunk_overlap=80
)

chunks = text_splitter.split_documents(documents)

print("Total chunks:", len(chunks))
```

Total chunks: 159

```
In [15]: from langchain_huggingface.embeddings import HuggingFaceEmbeddings

embedding_model = HuggingFaceEmbeddings(
    model_name="sentence-transformers/all-MiniLM-L6-v2"
)
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab
(https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
warnings.warn(
```

```
modules.json: 0%|          | 0.00/349 [00:00<?, ?B/s]
config_sentence_transformers.json: 0%|          | 0.00/116 [00:00<?, ?B/s]
README.md: 0.00B [00:00, ?B/s]
sentence_bert_config.json: 0%|          | 0.00/53.0 [00:00<?, ?B/s]
config.json: 0%|          | 0.00/612 [00:00<?, ?B/s]
model.safetensors: 0%|          | 0.00/90.9M [00:00<?, ?B/s]
tokenizer_config.json: 0%|          | 0.00/350 [00:00<?, ?B/s]
vocab.txt: 0.00B [00:00, ?B/s]
tokenizer.json: 0.00B [00:00, ?B/s]
special_tokens_map.json: 0%|          | 0.00/112 [00:00<?, ?B/s]
config.json: 0%|          | 0.00/190 [00:00<?, ?B/s]
```

```
In [16]: # ChromaDB Vector Store
from langchain_community.vectorstores import Chroma

vectorstore = Chroma.from_documents(
    documents=chunks,
    embedding=embedding_model,
    persist_directory="./chroma_ewu_disciplinary"
)

vectorstore.persist()
print("ChromaDB for EWU Disciplinary Code created.")
```

ChromaDB for EWU Disciplinary Code created.

```
/tmp/ipython-input-3412040844.py:10: LangChainDeprecationWarning: Since Chroma 0.4.x the manual persistence method is
no longer supported as docs are automatically persisted.
vectorstore.persist()
```

Load Model

```
In [17]: # Login to hugging face (if use licenced model, need to authenticate the token)
from huggingface_hub import login
login()
# hf_FVFIyHzCHQPxDwXtuuZWYkuGsnECqJxtn -> Token
```

```
VBox(children=(HTML(value='<center> <img\nsrc=https://huggingface.co/front/assets/huggingface_logo-noborder.svg...
```

```
In [18]: # Load Meta-Llama-3-8B
from transformers import AutoTokenizer, AutoModelForCausalLM, pipeline
import torch

model_id = "google/gemma-2b-it"
```

```
tokenizer = AutoTokenizer.from_pretrained(model_id, use_auth_token=True)
model = AutoModelForCausalLM.from_pretrained(
    model_id,
    device_map="auto",
    torch_dtype=torch.float16,
    use_auth_token=True
)

llm_pipeline = pipeline(
    "text-generation",
    model=model,
    tokenizer=tokenizer,
    max_new_tokens=256,
    do_sample=False,
    pad_token_id=tokenizer.eos_token_id
)
```

/usr/local/lib/python3.12/dist-packages/transformers/models/auto/tokenization_auto.py:1041: FutureWarning: The `use_auth_token` argument is deprecated and will be removed in v5 of Transformers. Please use `token` instead.
warnings.warn(

```
tokenizer_config.json: 0%|          | 0.00/34.2k [00:00<?, ?B/s]
tokenizer.model: 0%|          | 0.00/4.24M [00:00<?, ?B/s]
tokenizer.json: 0%|          | 0.00/17.5M [00:00<?, ?B/s]
special_tokens_map.json: 0%|          | 0.00/636 [00:00<?, ?B/s]
```

/usr/local/lib/python3.12/dist-packages/transformers/models/auto/auto_factory.py:492: FutureWarning: The `use_auth_token` argument is deprecated and will be removed in v5 of Transformers. Please use `token` instead.
warnings.warn(

```
config.json: 0%|          | 0.00/627 [00:00<?, ?B/s]
```

`torch_dtype` is deprecated! Use `dtype` instead!

```
model.safetensors.index.json: 0%|          | 0.00/13.5k [00:00<?, ?B/s]
Fetching 2 files: 0%|          | 0/2 [00:00<?, ?it/s]
model-00002-of-00002.safetensors: 0%|          | 0.00/67.1M [00:00<?, ?B/s]
model-00001-of-00002.safetensors: 0%|          | 0.00/4.95G [00:00<?, ?B/s]
Loading checkpoint shards: 0%|          | 0/2 [00:00<?, ?it/s]
generation_config.json: 0%|          | 0.00/137 [00:00<?, ?B/s]
```

Device set to use cuda:0
The following generation flags are not valid and may be ignored: ['temperature']. Set `TRANSFORMERS_VERBOSITY=info` for more details.

```
In [19]: # RAG Answer Generator
def rag_answer(question, context):
    prompt = f"""
    You are answering questions about the East West University Disciplinary Code.

    Answer STRICTLY using the context.
    If the answer is not explicitly stated, say:
    "I don't know based on the provided documents."

    Context:
    {context}

    Question:
    {question}

    Answer:
    """
    result = llm_pipeline(prompt)
    return result[0]["generated_text"].split("Answer: ")[-1].strip()
```

Retriever

```
In [20]: def retrieve_docs(state):
    query = state["query"]
    docs = vectorstore.similarity_search(query, k=4)

    state["context"] = "\n\n".join(
        f"[Section {d.metadata['section']} | Clause {d.metadata['clause_id']}] \n {d.page_content}"
        for d in docs
```

```
)  
return state
```

```
In [21]: def planner(state):  
        state["action"] = "answer"  
        return state  
  
def answer_node(state):  
    answer = rag_answer(state["query"], state["context"])  
    state["final_answer"] = answer  
    return state  
  
from langgraph.graph import StateGraph  
  
graph = StateGraph(dict)  
  
graph.add_node("planner", planner)  
graph.add_node("retriever", retrieve_docs)  
graph.add_node("answer", answer_node)  
  
graph.set_entry_point("planner")  
graph.add_edge("planner", "retriever")  
graph.add_edge("retriever", "answer")  
graph.set_finish_point("answer")  
  
agent = graph.compile()
```

```
In [24]: query = "What are “unfair or illegal means” in examinations according to the Code?"  
  
result = agent.invoke({"query": query})  
  
print(result["final_answer"])
```

According to the context, unfair or illegal means include communicating or attempting to communicate verbally or otherwise any in

Testing Queries

Test your system with this queries:

1. What actions are considered student misconduct under the EWU Disciplinary Code?
2. What are "unfair or illegal means" in examinations according to the Code?
3. What punishments can be awarded if a student is found guilty of misconduct?
4. Who are the members of the Disciplinary Committee and what is its authority?
5. What is the procedure after a complaint is lodged against a student?
6. Can a student be suspended before the disciplinary hearing? If yes, under what conditions?
7. What rights does an accused student have during a disciplinary hearing?
8. How does the university handle cases related to sexual harassment?
9. Can a student appeal a disciplinary decision, and what is the appeal process?
10. What immediate powers does an invigilator have if unfair means are detected during an examination?

Task

Create a Gradio application that takes a user query as input, uses the LangGraph agent to process it, and then displays the original query, the context retrieved by the retriever node, and the final answer generated by the answer node. Finally, test the Gradio UI with various queries.

```
In [25]: !pip install -q gradio
```

```
In [26]: def rag_interface(user_query):  
        result = agent.invoke({"query": user_query})  
  
        original_query = result.get("query", "N/A")  
        retrieved_context = result.get("context", "No context retrieved.")  
        final_answer = result.get("final_answer", "No answer generated.")  
  
        formatted_output = f"""  
Original Query: {original_query}  
  
Retrieved Context:
```

```
{retrieved_context}

Final Answer: {final_answer}
"""
    return formatted_output

print("rag_interface function defined.")
```

rag_interface function defined.

Launch Gradio Application

```
In [28]: import gradio as gr

iface = gr.Interface(
    fn=rag_interface,
    inputs=gr.Textbox(label='Enter your query:', placeholder='What actions are considered student disciplinary actions?'),
    outputs=gr.Markdown(label='RAG Agent Output'),
    title='EWU Disciplinary Agent'
)

iface.launch(share=True)
```

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: <https://79da25f0bcf72694c8.gradio.live>

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory.



No interface is running right now

Out [28]: