

Artificial Intelligence

Laporan Progres Proyek Minggu ke-2



Disusun Oleh:

Andreas Teguh Santoso Kosasih - 140810230047

Muhammad Raihan Rizky Zain - 140810230049

Atharik Putra Rajendra - 140810230077

PROGRAM STUDI S1 TEKNIK INFORMATIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS PADJADJARAN

JATINANGOR

2025

Source Code:

1. *test.py*

```
import streamlit as st

st.set_page_config(page_title="Chatbot Edukasi AI", layout="wide")

import json
import faiss
import numpy as np
import os
import re
from dotenv import load_dotenv
import google.generativeai as genai
from sentence_transformers import SentenceTransformer

# --- 0. Konfigurasi Awal & Pemuatan Variabel Lingkungan ---
load_dotenv()
GEMINI_API_KEY = os.getenv("GEMINI_API_KEY")

if not GEMINI_API_KEY:
    st.error("Variabel lingkungan GEMINI_API_KEY tidak ditemukan. Harap atur API Key Anda.")
    st.stop()

try:
    genai.configure(api_key=GEMINI_API_KEY)
    print("Konfigurasi Gemini API berhasil.")
except Exception as e:
    st.error(f"Gagal mengkonfigurasi Gemini API: {e}")
    st.stop()

# --- Variabel Global & Path Konfigurasi ---
# Sesuaikan dengan nama model embedding yang Anda gunakan di prepare_data_for_rag.py
EMBEDDING_MODEL_NAME = 'all-MiniLM-L6-v2'

CURRENT_WORKING_DIRECTORY = os.getcwd()
print(f"Direktori Kerja Saat Ini (CWD): {CURRENT_WORKING_DIRECTORY}")

BASE_DATA_DIR = os.path.join("dataset", "SistemOperasi")

print(f"BASE_DATA_DIR diatur ke (relatif terhadap CWD atau absolut): {BASE_DATA_DIR}")
print(f"Path absolut yang akan digunakan untuk BASE_DATA_DIR: {os.path.abspath(BASE_DATA_DIR)}")

FAISS_INDEX_FILEPATH = os.path.join(BASE_DATA_DIR, "vector_store.index")
TEXT_CHUNKS_FILEPATH = os.path.join(BASE_DATA_DIR, "processed_chunks_with_metadata.json")
OUTLINE_FILEPATH = os.path.join(BASE_DATA_DIR, "outline_operating_systems.txt")

# Model LLM Gemini yang akan digunakan
LLM_MODEL_NAME = "gemini-1.5-flash-latest"
```

```

# --- 1. Pemuatan Resources Utama (Menggunakan Cache Streamlit) ---
@st.cache_resource
def load_all_application_resources():
    """Memuat semua resource yang dibutuhkan aplikasi."""
    print("Memulai pemuatan resources aplikasi...")
    print(f"          Mencoba memuat FAISS index dari:
{os.path.abspath(FAISS_INDEX_FILEPATH)}")
    print(f"          Mencoba memuat Text Chunks dari:
{os.path.abspath(TEXT_CHUNKS_FILEPATH)}")
    print(f"          Mencoba memuat Outline dari:
{os.path.abspath(OUTLINE_FILEPATH)}")

    resources = {
        "faiss_index": None,
        "text_chunks_with_metadata": [],
        "parsed_outline": [],
        "query_embedding_model": None,
        "llm_model": None
    }

    all_paths_exist = True
    if not os.path.exists(FAISS_INDEX_FILEPATH):
        st.error(f"File FAISS index TIDAK DITEMUKAN di:
{os.path.abspath(FAISS_INDEX_FILEPATH)}")
        all_paths_exist = False
    if not os.path.exists(TEXT_CHUNKS_FILEPATH):
        st.error(f"File text chunks JSON TIDAK DITEMUKAN di:
{os.path.abspath(TEXT_CHUNKS_FILEPATH)}")
        all_paths_exist = False
    if not os.path.exists(OUTLINE_FILEPATH):
        st.error(f"File outline mata kuliah TIDAK DITEMUKAN di:
{os.path.abspath(OUTLINE_FILEPATH)}")
        all_paths_exist = False

    if not all_paths_exist:
        st.warning("Satu atau lebih file data penting tidak ditemukan.
Aplikasi mungkin tidak berfungsi dengan benar. Harap periksa path di
atas dan pastikan file ada.")
        return {key: None for key in resources}

    try:
        # Muat FAISS Index
        resources["faiss_index"] =
faiss.read_index(FAISS_INDEX_FILEPATH)
        print(f"FAISS index dimuat ({resources['faiss_index'].ntotal}
vektor).")

        # Muat Text Chunks dengan Metadata
        with open(TEXT_CHUNKS_FILEPATH, "r", encoding="utf-8") as f:
            resources["text_chunks_with_metadata"] = json.load(f)
            print(f"Text chunks dimuat
({len(resources['text_chunks_with_metadata'])} chunk).")

        # Muat dan Parse Outline Mata Kuliah
        parsed_outline_data = []
        with open(OUTLINE_FILEPATH, 'r', encoding='utf-8') as f:
            content = f.read()

        if content.strip().startswith("MATAKULIAH:"):

```

```

        try:
            st.session_state.nama_matakuliah =
content.splitlines()[0].split(":",1)[1].strip()
        except Exception:
            pass

        pertemuan_blocks = re.split(r'\nPERTEMUAN:', '\n' +
content.split('PERTEMUAN:', 1)[-1] if 'PERTEMUAN:' in content else '')
        for block in pertemuan_blocks:
            if not block.strip(): continue
            current_pertemuan = {}
            lines = block.strip().splitlines()
            if lines:
                # Coba parse ID dari awal baris pertama blok
                id_match_from_line_start = re.match(r'^\s*(\d+)',
lines[0])

                if id_match_from_line_start:
                    current_pertemuan['id'] =
int(id_match_from_line_start.group(1))

                # Proses sisa baris untuk KEY: VALUE
                for line_idx, line in enumerate(lines):
                    if ":" in line:
                        key, value = line.split(":", 1)
                        key_clean = key.strip().lower().replace(" ",
"_" )

                        value_clean = value.strip()

                        if key_clean == "pertemuan" and 'id' not in
current_pertemuan:
                            id_match_val = re.match(r'^\s*(\d+)',
value_clean)

                            if id_match_val:
                                current_pertemuan['id'] =
int(id_match_val.group(1))

                                elif key_clean not in current_pertemuan or
key_clean == 'judul': # Ambil judul dari baris pertama jika ada
                                    if key_clean == 'judul' and line_idx == 0
and 'id' in current_pertemuan:
                                        # Khusus untuk kasus "ID JUDUL: Isi
Judul"

                                        current_pertemuan[key_clean] =
value_clean.split(":",1)[-1].strip() if ":" in value_clean and
value_clean.startswith(str(current_pertemuan['id'])) else value_clean
                                        elif key_clean == 'judul' and
lines[0].strip().startswith(str(current_pertemuan.get('id','')) + " " +
key.strip()) :
                                            current_pertemuan[key_clean] =
value_clean

                                        elif key_clean != 'pertemuan':
                                            current_pertemuan[key_clean] =
value_clean

                                if 'id' in current_pertemuan and 'judul' in
current_pertemuan:
                                    parsed_outline_data.append(current_pertemuan)
                                else:

```

```

        print(f"Peringatan: Gagal mem-parsing ID atau Judul
untuk blok: {lines[:2]}")

        resources["parsed_outline"] = parsed_outline_data
        print(f"Outline mata kuliah dimuat
({len(resources['parsed_outline'])} pertemuan).")

        # Muat Model Embedding untuk Query
        print(f"Memuat model embedding untuk query:
{EMBEDDING_MODEL_NAME}...")
        resources["query_embedding_model"] =
SentenceTransformer(EMBEDDING_MODEL_NAME)
        print("Model embedding query berhasil dimuat.")

        # Muat Model LLM Gemini
        print(f"Memuat model LLM Gemini: {LLM_MODEL_NAME}...")
        resources["llm_model"] =
genai.GenerativeModel(model_name=LLM_MODEL_NAME)
        print("Model LLM Gemini berhasil dimuat.")

        st.success("Semua resources aplikasi berhasil dimuat!")
        return resources

    except FileNotFoundError as fnf_error:
        st.error(f"Kesalahan File Tidak Ditemukan saat memuat
resources: {fnf_error}")
        print(f"Detail FileNotFoundError saat
load_all_application_resources: {fnf_error}")
        import traceback
        traceback.print_exc()
        return {key: None for key in resources} # Kembalikan None untuk
semua resource

    except Exception as e:
        st.error(f"Terjadi kesalahan fatal saat memuat resources: {e}")
        print(f"Detail error umum saat load_all_application_resources:
{e}")
        import traceback
        traceback.print_exc()
        return {key: None for key in resources}

# Panggil fungsi untuk memuat resources
app_resources = load_all_application_resources()
faiss_search_index = app_resources.get("faiss_index") # Gunakan .get()
untuk keamanan
loaded_text_chunks_with_metadata =
app_resources.get("text_chunks_with_metadata", [])
parsed_outline = app_resources.get("parsed_outline", [])
query_embedding_model = app_resources.get("query_embedding_model")
llm_chat_model = app_resources.get("llm_model")

# --- 2. Fungsi-Fungsi Inti untuk RAG dan Kuis ---

def get_embedding_for_query(user_query_text):
    if query_embedding_model is None:
        st.warning("Model embedding query belum siap.")
        return None

```

```

try:
    return query_embedding_model.encode([user_query_text])[0]
except Exception as e:
    print(f"Error embedding query: {e}")
    st.error(f"Error saat membuat embedding untuk query: {e}")
    return None

def search_relevant_chunks(query_embedding_vector,
current_pertemuan_id=None, top_k=5):
    if faiss_search_index is None or query_embedding_vector is None or
not loaded_text_chunks_with_metadata:
        st.warning("Komponen RAG (FAISS/chunks/query embedding) belum
siap untuk pencarian.")
        return []
    try:
        query_np_array =
np.array([query_embedding_vector]).astype('float32')
        if faiss_search_index.d != query_np_array.shape[1]:
            st.error(f"Dimensi embedding query
({query_np_array.shape[1]}) tidak cocok dengan index FAISS
({faiss_search_index.d}).")
            return []

        # Ambil lebih banyak jika perlu filter, terutama jika
current_pertemuan_id tidak None
        num_to_search = top_k * 5 if current_pertemuan_id is not None
else top_k
        num_to_search = min(num_to_search, faiss_search_index.ntotal) #
Jangan cari lebih dari total vektor

        distances, global_indices =
faiss_search_index.search(query_np_array, num_to_search)

        retrieved_chunks_texts = []
        for i in global_indices[0]:
            if i != -1 and 0 <= i <
len(loaded_text_chunks_with_metadata):
                chunk_data = loaded_text_chunks_with_metadata[i]
                # Filter berdasarkan ID pertemuan JIKA
current_pertemuan_id diberikan
                if current_pertemuan_id is None or
chunk_data.get("pertemuan_id") == current_pertemuan_id:
                    retrieved_chunks_texts.append(chunk_data["chunk_text"])
                    if len(retrieved_chunks_texts) == top_k: # Jika
sudah cukup (baik dengan filter atau tanpa)
                        break
        return retrieved_chunks_texts
    except Exception as e:
        print(f"Error saat search_relevant_chunks: {e}")
        st.error(f"Error saat melakukan pencarian di FAISS: {e}")
        return []

def get_rag_answer_from_llm(user_query, context_chunks):
    if llm_chat_model is None:
        st.warning("Model LLM tidak siap.")
        return "Error: Model LLM tidak siap."

```

```

prompt_to_send = ""
if not context_chunks:
    prompt_to_send = f"Jawab pertanyaan berikut berdasarkan
pengetahuan umum Anda: \"{user_query}\""
    print("INFO: Menjawab tanpa konteks RAG karena tidak ada chunk
relevan ditemukan.")
else:
    context_string = "\n\n---\n\n".join(context_chunks)
    prompt_to_send = f""Anda adalah asisten AI edukasi yang cerdas
dan membantu.
Berdasarkan KONTEKS MATERI di bawah ini, jawablah PERTANYAAN MAHASISWA
dengan jelas dan akurat.
Fokuskan jawaban Anda HANYA pada informasi yang ada dalam KONTEKS
MATERI.
Jika informasi tidak ada dalam konteks, katakan bahwa Anda tidak dapat
menemukannya dalam materi yang disediakan.

KONTEKS MATERI:
---
{context_string}
---

PERTANYAAN MAHASISWA:
"{user_query}"

JAWABAN ANDA:
"""
    try:
        response = llm_chat_model.generate_content(prompt_to_send)
        if response.parts:
            return "".join(part.text for part in response.parts)
        elif hasattr(response, 'text') and response.text:
            return response.text
        elif hasattr(response, 'prompt_feedback') and
response.prompt_feedback:
            feedback_info = response.prompt_feedback
            print(f"Feedback dari LLM untuk prompt: {feedback_info}")
            return f"Tidak dapat menghasilkan jawaban. Feedback:
{feedback_info}"
        else:
            print(f"Respons LLM tidak memiliki 'text' atau 'parts':
{response}")
            return "Maaf, format respons dari LLM tidak dikenali."

    except Exception as e:
        print(f"Error saat get_rag_answer_from_llm: {e}")
        st.error(f"Maaf, terjadi kesalahan saat mencoba menghasilkan
jawaban: {e}")
        return "Maaf, terjadi kesalahan internal saat mencoba
menghasilkan jawaban."

def generate_mcq_from_llm(pertemuan_id, num_questions=3):
    if llm_chat_model is None:
        st.warning("Model LLM tidak siap untuk generasi soal.")
        return []

    relevant_chunks_for_quiz = [

```

```

        chunk["chunk_text"] for chunk in
loaded_text_chunks_with_metadata
        if chunk.get("pertemuan_id") == pertemuan_id
    ]
    if not relevant_chunks_for_quiz:
        st.warning(f"Tidak ada materi chunk yang ditemukan untuk
pertemuan ID {pertemuan_id} untuk membuat soal.")
        return []

    sample_context_for_quiz =
"\n\n---\n\n".join(relevant_chunks_for_quiz[:min(len(relevant_chunks_for_quiz), 5)])

    prompt_quiz_generation = f"""Anda adalah seorang ahli pembuat soal
ujian.
Berdasarkan potongan materi kuliah berikut:
---
{sample_context_for_quiz}
---
Tolong buat saya {num_questions} soal pilihan ganda yang menguji
pemahaman mahasiswa mengenai konsep-konsep utama dalam materi di atas.
Untuk setiap soal, sertakan:
1. "pertanyaan": Pertanyaan yang jelas.
2. "opsi": Sebuah dictionary berisi empat opsi jawaban (kunci: "A",
"B", "C", "D").
3. "jawaban_benar": Kunci dari opsi yang benar (misalnya, "B").
4. "penjelasan_jawaban": Penjelasan singkat mengapa jawaban tersebut
benar dan opsi lain salah.
5. "topik_terkait": Topik atau sub-bagian spesifik dari materi yang
diuji oleh soal ini.

Format output HARUS berupa list dari JSON object yang valid, seperti
ini:
[
    {{
        "pertanyaan": "Contoh pertanyaan 1...",
        "opsi": {{ "A": "Opsi A1", "B": "Opsi B1", "C": "Opsi C1", "D":
"Opsi D1" }},
        "jawaban_benar": "A",
        "penjelasan_jawaban": "Penjelasan untuk soal 1...",
        "topik_terkait": "Topik terkait soal 1"
    }},
    {{
        "pertanyaan": "Contoh pertanyaan 2...",
        "opsi": {{ "A": "Opsi A2", "B": "Opsi B2", "C": "Opsi C2", "D":
"Opsi D2" }},
        "jawaban_benar": "C",
        "penjelasan_jawaban": "Penjelasan untuk soal 2...",
        "topik_terkait": "Topik terkait soal 2"
    }}
]
Pastikan outputnya adalah JSON list yang valid dan tidak ada teks
tambahan di luar list JSON tersebut.
"""

    print(f"DEBUG: Prompt untuk generasi soal
(sebagian):\n{prompt_quiz_generation[:300]}...")
    try:
        response =

```



```

llm_chat_model.generate_content(prompt_quiz_generation)
    response_text = ""
    if response.parts:
        response_text = "".join(part.text for part in
response.parts)
    elif hasattr(response, 'text') and response.text:
        response_text = response.text

        print(f"DEBUG: Respons mentah dari LLM untuk generasi
soal:\n{response_text}")

        # Mencari blok JSON yang valid dalam respons
        # Ini lebih toleran terhadap teks tambahan sebelum atau sesudah
JSON

        json_match =
re.search(r'\[\\s*(\\[[\\s\\S]*?\\])(?:\\s*,\\s*\\[[\\s\\S]*?\\])*\\s*\\]',
response_text, re.DOTALL)

        if json_match:
            json_str = json_match.group(0) # Ambil seluruh match [ ...
]

            try:
                questions = json.loads(json_str)
                # Validasi sederhana struktur soal
                if isinstance(questions, list) and all(isinstance(q,
dict) and "pertanyaan" in q and "opsi" in q and "jawaban_benar" in q
for q in questions):
                    print(f"Berhasil mem-parsing {len(questions)} soal
dari LLM.")
                    return questions
                else:
                    st.error("Format JSON soal dari LLM tidak sesuai
ekspektasi setelah parsing.")
                    print(f"Format JSON soal tidak sesuai. Parsed:
{questions}")
                    return []
            except json.JSONDecodeError as je:
                st.error(f"Gagal mem-parsing JSON soal dari LLM: {je}")
                print(f"JSON Decode Error. String yang dicoba parse:
{json_str}")
                return []
            else:
                st.error("Tidak menemukan format JSON list yang valid dalam
respons LLM untuk soal.")
                print(f"Tidak ada JSON list valid ditemukan dalam:
{response_text}")
                # Cek jika ada feedback blocking
                if hasattr(response, 'prompt_feedback') and
response.prompt_feedback:
                    if response.prompt_feedback.block_reason:
                        st.warning(f"Generasi soal mungkin diblokir:
{response.prompt_feedback.block_reason_message}")
                    return []

        except Exception as e:
            st.error(f"Error saat men-generate soal dari LLM: {e}")
            print(f"Error detail saat generate_mcq_from_llm: {e}")
            import traceback


```

```

        traceback.print_exc()
        return []

# --- 3. Inisialisasi Session State ---
if "chat_history" not in st.session_state:
    st.session_state.chat_history = []
if "current_pertemuan_id" not in st.session_state:
    st.session_state.current_pertemuan_id = None
if "current_pertemuan_judul" not in st.session_state:
    st.session_state.current_pertemuan_judul = None
if "quiz_questions" not in st.session_state:
    st.session_state.quiz_questions = []
if "current_question_index" not in st.session_state:
    st.session_state.current_question_index = 0
if "user_answers" not in st.session_state:
    st.session_state.user_answers = {}
if "quiz_mode" not in st.session_state:
    st.session_state.quiz_mode = None
if "quiz_score" not in st.session_state:
    st.session_state.quiz_score = {"benar": 0, "salah": 0, "total_soal": 0, "topik_salah": {}}
if "pemahaman_mahasiswa" not in st.session_state:
    st.session_state.pemahaman_mahasiswa = {}
if "nama_matakuliah" not in st.session_state:
    st.session_state.nama_matakuliah = "Mata Kuliah Anda" # Default
if "last_processed_query" not in st.session_state:
    st.session_state.last_processed_query = None

# --- 4. Antarmuka Pengguna (UI) Streamlit ---

# Sidebar untuk Navigasi Pertemuan
with st.sidebar:
    st.header(f" "
            elif level_paham == "Perlu Belajar Lagi":
                display_judul += "  "

            if st.button(display_judul,
                key=f"pertemuan_{pertemuan_id}", use_container_width=True):
                if st.session_state.current_pertemuan_id != pertemuan_id:
                    st.session_state.current_pertemuan_id =

```

```

pertemuan_id

        st.session_state.current_pertemuan_judul = judul
        st.session_state.quiz_mode = None
        st.session_state.chat_history = []
        st.session_state.last_processed_query = None
        st.rerun()

    st.divider()
    if st.sidebar.button("🔄 Reset Aplikasi & Muat Ulang Resources",
type="primary", use_container_width=True):
        st.cache_resource.clear()
        keys_to_reset = ["chat_history", "current_pertemuan_id",
"current_pertemuan_judul",
                                "quiz_questions", "current_question_index",
"user_answers",
                                "quiz_mode", "quiz_score",
"pemahaman_mahasiswa", "last_processed_query", "nama_matakuliah"]
        for key in keys_to_reset:
            if key in st.session_state:
                del st.session_state[key]
        st.rerun()

# Tampilan Utama Aplikasi
if st.session_state.current_pertemuan_id is None:
    st.title("Selamat Datang di Chatbot Edukasi AI!")
    st.write("Silakan pilih pertemuan dari menu di sebelah kiri untuk
memulai.")
else:
    st.title(f"📌 Pertemuan {st.session_state.current_pertemuan_id}:
{st.session_state.current_pertemuan_judul}")

    col1, col2 = st.columns(2)
    with col1:
        if st.button("💬 Tanya Jawab Materi Ini",
use_container_width=True, disabled=(st.session_state.quiz_mode is not
None and st.session_state.quiz_mode != "quiz_results_chat_forward")):
            if st.session_state.quiz_mode != None:
                st.session_state.quiz_mode = None
                st.session_state.chat_history = []
                st.session_state.last_processed_query = None
                st.rerun()

    with col2:
        if st.button("📝 Uji Pemahaman Saya", use_container_width=True,
type="primary", disabled=(st.session_state.quiz_mode ==
"quiz_ongoing")):
            st.session_state.quiz_mode = "quiz_generating"
            st.session_state.chat_history = []
            st.session_state.last_processed_query = None
            st.rerun()

    st.divider()

    if st.session_state.quiz_mode == "quiz_generating":
        with st.spinner("Sedang membuat soal untuk Anda... Mohon tunggu
sebentar. ⌚"):
            st.session_state.quiz_questions =
generate_mcq_from_llm(st.session_state.current_pertemuan_id,
num_questions=3)

            if st.session_state.quiz_questions and

```

```

isinstance(st.session_state.quiz_questions, list) and
len(st.session_state.quiz_questions) > 0 :
    st.session_state.current_question_index = 0
    st.session_state.user_answers = {}
    st.session_state.quiz_score = {"benar": 0, "salah": 0,
"total_soal": len(st.session_state.quiz_questions), "topik_salah": {}}
    st.session_state.quiz_mode = "quiz_ongoing"
else:
    st.error("Gagal membuat soal kuis atau format soal
tidak valid. Silakan coba lagi atau pilih pertemuan lain.")
    st.session_state.quiz_mode = None
    st.rerun()

elif st.session_state.quiz_mode == "quiz_ongoing":
    st.subheader("👉 Kuis Pemahaman")
    if not st.session_state.quiz_questions or
st.session_state.current_question_index >=
len(st.session_state.quiz_questions):
        st.session_state.quiz_mode = "quiz_results"
        st.rerun()
    else:
        q_idx = st.session_state.current_question_index
        question_data = st.session_state.quiz_questions[q_idx]

        st.markdown(f"**Soal {q_idx + 1} dari
{len(st.session_state.quiz_questions)}:**")
        st.markdown(f"#### {question_data.get('pertanyaan',
'Pertanyaan tidak tersedia.')}")

        options = question_data.get("opsi", {})
        option_items = list(options.items())

        if not option_items:
            st.error("Opsi jawaban tidak tersedia untuk soal ini.")
            if st.button("Lanjut ke Soal Berikutnya (jika ada)":
                if q_idx < len(st.session_state.quiz_questions) -
1:
                    st.session_state.current_question_index +=1
                else:
                    st.session_state.quiz_mode = "quiz_results"
                    st.rerun()
            # return # Hentikan render soal ini jika tidak ada opsi

        user_choice_for_current_q =
st.session_state.user_answers.get(q_idx)

        default_radio_index = None
        if user_choice_for_current_q:
            try:
                default_radio_index = [item[0] for item in
option_items].index(user_choice_for_current_q)
            except ValueError:
                default_radio_index = None

        selected_option_key = st.radio(
            "Pilih jawaban Anda:",
            options=[item[0] for item in option_items],
            format_func=lambda key: f"{key}. {options.get(key,

```

```

'Opsi tidak valid'}}",

key=f"quiz_q_{st.session_state.current_pertemuan_id}_{q_idx}",
    index=default_radio_index
    )

    if selected_option_key:
        st.session_state.user_answers[q_idx] =
selected_option_key

        nav_cols = st.columns(3)
        with nav_cols[0]:
            if st.button("⬅ Soal Sebelumnya", disabled=(q_idx ==
0), use_container_width=True):
                st.session_state.current_question_index -= 1
                st.rerun()
            with nav_cols[1]:
                if q_idx == len(st.session_state.quiz_questions) - 1:
                    if st.button("🚩 Selesai Kuis & Lihat Hasil",
type="primary", use_container_width=True, disabled=(selected_option_key
is None and user_choice_for_current_q is None)):
                        if selected_option_key:
st.session_state.user_answers[q_idx] = selected_option_key
                            st.session_state.quiz_mode = "quiz_results"
                            st.rerun()
                        else:
                            if st.button("Soal Berikutnya ➡",
use_container_width=True, disabled=(selected_option_key is None and
user_choice_for_current_q is None)):
                                if selected_option_key:
st.session_state.user_answers[q_idx] = selected_option_key
                                    st.session_state.current_question_index += 1
                                    st.rerun()
                                with nav_cols[2]:
                                    if st.button("❌ Batalkan Kuis",
use_container_width=True):
                                        st.session_state.quiz_mode = None
                                        st.rerun()

elif st.session_state.quiz_mode == "quiz_results":
    st.subheader("📊 Hasil Uji Pemahaman Anda")
    benar, salah = 0, 0
    topik_salah_dict = {}
    total_soal_quiz = len(st.session_state.quiz_questions)
    st.session_state.quiz_score["total_soal"] = total_soal_quiz

    for idx, q_data in enumerate(st.session_state.quiz_questions):
        user_ans = st.session_state.user_answers.get(idx)
        correct_ans_key = q_data.get("jawaban_benar")
        if user_ans == correct_ans_key:
            benar += 1
        else:
            salah += 1
            topik = q_data.get("topik_terkait", "Umum")
            topik_salah_dict[topik] = topik_salah_dict.get(topik,
0) + 1

```

```

st.session_state.quiz_score["benar"] = benar
st.session_state.quiz_score["salah"] = salah
st.session_state.quiz_score["topik_salah"] = topik_salah_dict

if total_soal_quiz > 0:
    persentase_benar = (benar / total_soal_quiz) * 100
    st.metric(label="Skor Anda",
value=f"{persentase_benar:.2f}%",      delta=f"{benar} benar dari
{total_soal_quiz} soal")

    level_paham = "Perlu Belajar Lagi"
    if persentase_benar > 80: level_paham = "Sangat Paham"
    elif persentase_benar >= 60: level_paham = "Paham Sebagian"

    if level_paham == "Sangat Paham": st.success("🎉 Luar
biasa! Pemahaman Anda sangat baik.")
    elif level_paham == "Paham Sebagian": st.info("👍 Bagus!
Ada beberapa poin yang bisa ditingkatkan.")
    else: st.warning("⚠️ Anda perlu mempelajari lagi beberapa
bagian.")

st.session_state.pemahaman_mahasiswa[st.session_state.current_pertemuan
_id] = level_paham

if topik_salah_dict:
    st.markdown("#### Topik yang Perlu Diperdalam:")
    for topik, count in topik_salah_dict.items():
        st.markdown(f"- **{topik}** ({count} kesalahan)")
        if st.button(f"💬 Tanya tentang: {topik}",
key=f"learn_{st.session_state.current_pertemuan_id}_{topik.replace('
', '_')}"):
            st.session_state.chat_history = []
            st.session_state.quiz_mode =
"quiz_results_chat_forward"
            st.session_state.auto_send_prompt = f"Tolong
jelaskan lebih detail mengenai topik '{topik}' dari Pertemuan
{st.session_state.current_pertemuan_id}
({st.session_state.current_pertemuan_judul})."
            st.rerun()
            st.markdown("---")
            st.markdown("#### Detail Jawaban:")
            for idx, q_data in
enumerate(st.session_state.quiz_questions):
                with st.expander(f"Soal {idx+1}:
{q_data.get('pertanyaan', 'N/A')}"):
                    user_ans_key =
st.session_state.user_answers.get(idx)
                    user_ans_text =
q_data.get("opsi", {}).get(user_ans_key, "Tidak Dijawab")
                    correct_ans_key = q_data.get("jawaban_benar")
                    correct_ans_text =
q_data.get("opsi", {}).get(correct_ans_key, "N/A")

                    st.markdown(f"**Jawaban Anda:** {user_ans_key or
''}. {user_ans_text}")
                    st.markdown(f"**Jawaban Benar:** {correct_ans_key
or ''}. {correct_ans_text}")

```

```

        if user_ans_key == correct_ans_key: st.success("✓ Benar")
        else: st.error("✗ Salah")
            st.markdown(f"**Penjelasan:**")
            {q_data.get('penjelasan_jawaban', 'Tidak ada penjelasan.')}
            st.caption(f"Topik Terkait: ")
            {q_data.get('topik_terkait', 'Tidak diketahui')}
        else:
            st.warning("Tidak ada soal yang dijawab untuk ditampilkan hasilnya.")

    if st.button("← Kembali ke Pilihan Aksi"):
        st.session_state.quiz_mode = None
        st.rerun()

# Mode Tanya Jawab RAG (default atau setelah forwarding dari hasil kuis)
else:
    for message_entry in st.session_state.chat_history:
        with st.chat_message(message_entry["role"]):
            st.markdown(message_entry["content"])

    # Logika untuk auto-send prompt setelah kuis
    if st.session_state.quiz_mode == "quiz_results_chat_forward" and "auto_send_prompt" in st.session_state:
        auto_query = st.session_state.auto_send_prompt
        del st.session_state.auto_send_prompt
        st.session_state.quiz_mode = None

        st.session_state.chat_history.append({"role": "user", "content": auto_query})

        query_vector = get_embedding_for_query(auto_query)
        if query_vector is not None:
            relevant_chunks = search_relevant_chunks(query_vector, current_pertemuan_id=st.session_state.current_pertemuan_id)
            assistant_response = get_rag_answer_from_llm(auto_query, relevant_chunks)
        else:
            assistant_response = "Gagal memproses prompt otomatis (embedding error). "

        st.session_state.chat_history.append({"role": "assistant", "content": assistant_response})
        st.session_state.last_processed_query = auto_query
        st.rerun()

# Terima input chat dari pengguna
if user_chat_input := st.chat_input("Ketik pertanyaan Anda di sini..."):
    st.session_state.chat_history.append({"role": "user", "content": user_chat_input})

    query_vector = get_embedding_for_query(user_chat_input)
    if query_vector is not None:
        relevant_chunks = search_relevant_chunks(query_vector, current_pertemuan_id=st.session_state.current_pertemuan_id)
        assistant_response =

```

```

get_rag_answer_from_llm(user_chat_input, relevant_chunks)
    else:
        assistant_response = "Maaf, ada masalah saat memproses
embedding pertanyaan Anda."

        st.session_state.chat_history.append({"role": "assistant",
"content": assistant_response})
        st.session_state.last_processed_query = user_chat_input
        st.rerun()

if not all(app_resources.values()):
    st.error("Beberapa resource penting gagal dimuat. Aplikasi tidak
dapat berjalan. Silakan cek konsol server untuk detail dan coba 'Reset
Aplikasi'.")
    st.stop()

```

2. *prepare_data_for_rag.ipynb*

```

# %%
import json
import os
import re
import numpy as np
from sentence_transformers import SentenceTransformer
import faiss

# %%
# --- Konfigurasi ---
BASE_DIR = r"dataset/SistemOperasi" # Direktori utama mata kuliah Anda
OUTLINE_FILE = os.path.join(BASE_DIR, "outline_operating_systems.txt")
OUTPUT_JSON_CHUNKS = os.path.join(BASE_DIR,
"processed_chunks_with_metadata.json")
OUTPUT_FAISS_INDEX = os.path.join(BASE_DIR, "vector_store.index")

EMBEDDING_MODEL_NAME = 'all-MiniLM-L6-v2' # Menghasilkan vektor 384
dimensi

# Parameter untuk chunking
MAX_CHUNK_SIZE_CHARS = 1000 # Ukuran maksimal chunk sebelum dipecah
lebih lanjut
CHUNK_OVERLAP_CHARS = 150 # Jumlah karakter tumpang tindih antar
sub-chunk
# Pola regex untuk mendeteksi heading (misal: # Judul, ## Sub Judul,
dst.)
HEADING_SPLIT_PATTERN = r"^(#{1,6})\s+.*$" # Tangkap baris yang
dimulai dengan 1-6 '#' diikuti spasi dan teks

# --- Fungsi Helper ---

def parse_outline(outline_filepath):
    """
    Mem-parsing file outline untuk mendapatkan informasi setiap
pertemuan.
    Mengasumsikan format KEY: VALUE dan pemisah antar pertemuan adalah
baris kosong.
    """
    pertemuan_list = []

```



```

try:
    with open(outline_filepath, 'r', encoding='utf-8') as f:
        content = f.read()

        # Pisahkan berdasarkan blok pertemuan (diasumsikan dipisah oleh
        'PERTEMUAN:')
        # dan pastikan ada MATAKULIAH di awal
        if not content.strip().startswith("MATAKULIAH:"):
            print(f"Peringatan: Format file outline
            '{outline_filepath}' mungkin tidak sesuai (tidak ada 'MATAKULIAH:').")
            # return [] # Bisa dihentikan jika format ketat

        # Menggunakan regex untuk menangkap blok pertemuan dengan lebih
        fleksibel
        # Pola ini mencari "PERTEMUAN:" dan mengambil semua baris
        hingga "PERTEMUAN:" berikutnya atau akhir file
        pertemuan_blocks = re.split(r'\nPERTEMUAN:', '\n' +
        content.split('PERTEMUAN:', 1)[-1] if 'PERTEMUAN:' in content else '')

        for block in pertemuan_blocks:
            if not block.strip():
                continue

            current_pertemuan = {}
            lines = block.strip().splitlines()

            # Ambil ID Pertemuan dari baris pertama blok (setelah
            "PERTEMUAN:")
            if lines:
                pertemuan_id_match = re.match(r'^\s*(\d+)', lines[0])
                if pertemuan_id_match:
                    current_pertemuan['id'] =
                    int(pertemuan_id_match.group(1))
                else:
                    print(f"Peringatan: Tidak bisa parse ID Pertemuan
                    dari blok: {lines[0]}")
                    continue # Lewati blok ini jika ID tidak bisa
                    diparse

            # Proses sisa baris untuk KEY: VALUE
            for line in lines: # Mulai dari baris pertama lagi
                untuk key lain juga
                    if ":" in line:
                        key, value = line.split(":", 1)
                        key = key.strip().lower().replace(" ", "_")
                        value = value.strip()
                        if key == "file_materi" and not value: # Jika
                        FILE_MATERI kosong
                            current_pertemuan[key] = None
                        else:
                            current_pertemuan[key] = value

                        if 'id' in current_pertemuan and 'judul' in
                        current_pertemuan and 'file_materi' in current_pertemuan :
                            # Pastikan file materi tidak None atau string kosong
                            untuk dimasukkan
                                if current_pertemuan.get('file_materi'):
                                    pertemuan_list.append(current_pertemuan)

```

```

        elif current_pertemuan.get('file_materi') is None or
not current_pertemuan.get('file_materi','').strip():
            print(f"Info: Pertemuan
{current_pertemuan.get('id','N/A')}
({current_pertemuan.get('judul','Tanpa Judul')}) tidak memiliki file
materi, akan dilewati untuk RAG.")

    except FileNotFoundError:
        print(f"Error: File outline '{outline_filepath}' tidak
ditemukan.")
    except Exception as e:
        print(f"Error saat mem-parsing file outline: {e}")

    print(f"Berhasil mem-parsing {len(pertemuan_list)} pertemuan dengan
file materi dari outline.")
    return pertemuan_list

def read_material_text(material_filepath):
    """Membaca konten teks dari file."""
    try:
        with open(material_filepath, 'r', encoding='utf-8') as f:
            return f.read()
    except FileNotFoundError:
        print(f"Error: File materi '{material_filepath}' tidak
ditemukan.")
        return ""
    except Exception as e:
        print(f"Error saat membaca file materi '{material_filepath}':
{e}")
        return ""

def _split_text_block_sliding_window(text_block, pertemuan_id,
pertemuan_judul, heading, max_size, overlap):
    """
    Helper untuk memecah blok teks yang panjang menggunakan sliding
window karakter.
    """
    final_chunks = []
    text_block_stripped = text_block.strip()
    if not text_block_stripped:
        return []

    if len(text_block_stripped) <= max_size:
        final_chunks.append({
            "pertemuan_id": pertemuan_id,
            "pertemuan_judul": pertemuan_judul,
            "original_heading": heading,
            "chunk_text": text_block_stripped
        })
    else:
        start_index = 0
        doc_len = len(text_block_stripped)
        while start_index < doc_len:
            end_index = start_index + max_size
            current_slice =
text_block_stripped[start_index:min(end_index, doc_len)]

```

```

        chunk_text_final = current_slice.strip()

        if chunk_text_final:
            final_chunks.append({
                "pertemuan_id": pertemuan_id,
                "pertemuan_judul": pertemuan_judul,
                "original_heading": heading,
                "chunk_text": chunk_text_final
            })

        if min(end_index, doc_len) >= doc_len:
            break

        start_index += (max_size - overlap)
        start_index = max(0, start_index)
        if start_index >= doc_len:
            break

    return final_chunks

def chunk_material_heading_aware(text_content, pertemuan_id,
                                pertemuan_judul):
    """
    Memecah konten materi menjadi chunks, mempertimbangkan heading
    sebagai pemisah alami.
    Jika teks di bawah satu heading terlalu panjang, akan dipecah lebih
    lanjut.
    """
    processed_chunks = []
    if not text_content or not text_content.strip():
        return []

    # Pisahkan teks berdasarkan heading, sambil mempertahankan
    headingnya.
    # re.split dengan capturing group (...) akan mempertahankan
    delimiter.
    parts = re.split(HEADING_SPLIT_PATTERN, text_content,
                     flags=re.MULTILINE)

    current_heading = "Umum" # Default untuk konten sebelum heading
    pertama
    accumulated_text_for_section = ""

    for i, part in enumerate(parts):
        part_stripped = part.strip()
        if not part_stripped:
            continue

        # Cek apakah part ini adalah heading (berdasarkan pola regex)
        is_current_part_a_heading = re.match(HEADING_SPLIT_PATTERN,
        part_stripped, flags=re.MULTILINE)

        if is_current_part_a_heading:
            # Jika ada teks yang sudah terakumulasi untuk section
            SEBELUMNYA, proses dulu
            if accumulated_text_for_section.strip():
                sub_chunks =
                _split_text_block_sliding_window(accumulated_text_for_section,

```

```

pertemuan_id, pertemuan_judul, current_heading,

MAX_CHUNK_SIZE_CHARS, CHUNK_OVERLAP_CHARS)
    processed_chunks.extend(sub_chunks)

    current_heading = part_stripped # Update heading saat ini
    accumulated_text_for_section = "" # Reset akumulator teks
else:
    # Ini adalah konten di bawah heading saat ini
    accumulated_text_for_section += part_stripped + "\n" #
    Tambahkan newline agar antar paragraf tidak menyatu

# Proses sisa teks yang terakumulasi untuk section terakhir
if accumulated_text_for_section.strip():
    sub_chunks =
    _split_text_block_sliding_window(accumulated_text_for_section,
                                     pertemuan_id,
    pertemuan_judul, current_heading,

MAX_CHUNK_SIZE_CHARS, CHUNK_OVERLAP_CHARS)
    processed_chunks.extend(sub_chunks)

return processed_chunks

def get_text_embeddings(list_of_chunk_texts,
model_name=EMBEDDING_MODEL_NAME):
    """Mengubah daftar teks chunk menjadi vektor embeddings."""
    if not list_of_chunk_texts:
        print("Tidak ada teks untuk di-embed.")
        return np.array([])
    try:
        print(f"Memuat model embedding: {model_name}...")
        embedding_model = SentenceTransformer(model_name)
        print("Model embedding berhasil dimuat.")
        print(f"Memulai proses embedding untuk
{len(list_of_chunk_texts)} chunk teks...")
        embeddings = embedding_model.encode(list_of_chunk_texts,
show_progress_bar=True)
        print(f"Proses embedding selesai. Dihasilkan
{embeddings.shape[0]} embeddings dengan dimensi
{embeddings.shape[1]}.")
        return embeddings
    except Exception as e:
        print(f"Error saat membuat embeddings: {e}")
        return np.array([])

def create_and_save_faiss_index(embeddings_np_array,
index_output_path=OUTPUT_FAISS_INDEX):
    """Membuat FAISS index dan menyimpannya."""
    if embeddings_np_array.size == 0 or embeddings_np_array.ndim != 2:
        print("Array embedding kosong atau formatnya salah. FAISS index
tidak dibuat.")
        return
    dimension = embeddings_np_array.shape[1]
    try:
        print(f"Membuat FAISS index dengan dimensi {dimension}...")
        index = faiss.IndexFlatL2(dimension)

```

```

        index.add(embeddings_np_array.astype('float32'))
        faiss.write_index(index, index_output_path)
        print(f"FAISS index dengan {index.ntotal} vektor berhasil
dibuat dan disimpan ke: {index_output_path}")
    except Exception as e:
        print(f"Error saat membuat atau menyimpan FAISS index: {e}")

# --- Proses Utama Skrip ---
if __name__ == "__main__":
    print("Memulai Prosesor RAG: Persiapan Data...")

    # 1. Parse file outline mata kuliah
    print(f"\nLangkah 1: Mem-parsing outline dari '{OUTLINE_FILE}'...")
    daftar_pertemuan = parse_outline(OUTLINE_FILE)

    if not daftar_pertemuan:
        print("Tidak ada informasi pertemuan yang valid dari outline.
Proses dihentikan.")
    else:
        all_processed_chunks_with_metadata = []

        # 2. Loop setiap pertemuan untuk membaca materi dan melakukan
chunking
        print(f"\nLangkah 2: Memproses materi per pertemuan...")
        for pertemuan_info in daftar_pertemuan:
            pertemuan_id = pertemuan_info.get('id')
            judul_pertemuan = pertemuan_info.get('judul', f"Pertemuan
{pertemuan_id}")
            file_materi_rel_path = pertemuan_info.get('file_materi')

            if not file_materi_rel_path:
                print(f"Info: Pertemuan ID {pertemuan_id}
({judul_pertemuan}) tidak memiliki path file materi. Dilewati.")
                continue

            file_materi_abs_path = os.path.join(BASE_DIR,
file_materi_rel_path)
            print(f"    Memproses: Pertemuan {pertemuan_id} -
'{judul_pertemuan}' dari file '{file_materi_abs_path}'")

            materi_text = read_material_text(file_materi_abs_path)
            if materi_text:
                chunks_for_this_pertemuan =
chunk_material_heading_aware(materi_text,
                                pertemuan_id,
                                judul_pertemuan)

            all_processed_chunks_with_metadata.extend(chunks_for_this_pertemuan)
            print(f"    Dihasilkan {len(chunks_for_this_pertemuan)}
chunk untuk pertemuan ini.")
            else:
                print(f"    Tidak ada konten teks yang dibaca dari
'{file_materi_abs_path}'.")

        print(f"\nTotal chunk yang diproses dari semua pertemuan:
{len(all_processed_chunks_with_metadata)}")

        if all_processed_chunks_with_metadata:
            # Ekstrak hanya teks chunk untuk proses embedding

```

```

        list_of_chunk_texts_for_embedding = [chunk['chunk_text']]
for chunk in all_processed_chunks_with_metadata]

    # 3. Buat Embeddings
    print(f"\nLangkah 3: Membuat embeddings untuk semua chunk
teks...")

    document_embeddings =
get_text_embeddings(list_of_chunk_texts_for_embedding)

    if document_embeddings.size > 0:
        # 4. Buat dan Simpan FAISS Index
        print(f"\nLangkah 4: Membuat dan menyimpan FAISS
index...")

        create_and_save_faiss_index(document_embeddings,
OUTPUT_FAISS_INDEX)

    # 5. Simpan semua chunks beserta metadatanya ke file
JSON
    print(f"\nLangkah 5: Menyimpan semua chunk yang
diproses (dengan metadata) ke JSON...")
    try:
        with open(OUTPUT_JSON_CHUNKS, "w",
encoding="utf-8") as f:
            json.dump(all_processed_chunks_with_metadata,
f, ensure_ascii=False, indent=2)
            print(f"Semua chunk yang diproses berhasil disimpan
ke: {OUTPUT_JSON_CHUNKS}")
        except Exception as e:
            print(f"Error saat menyimpan chunks ke JSON: {e}")
        else:
            print("Pembuatan FAISS index dan penyimpanan JSON
chunks dibatalkan karena tidak ada embeddings yang valid.")
        else:
            print("Tidak ada chunk yang diproses sama sekali. Pastikan
file materi ada dan berisi teks.")

    print("\n--- Prosesor RAG: Persiapan Data Selesai ---")

```

Penjelasan Laporan

1. Pendahuluan

Diskusi ini bertujuan untuk menganalisis implementasi salah satu algoritma Artificial Intelligence (AI) berdasarkan proyek yang sudah kita buat dengan mengidentifikasi jenis algoritma AI yang diimplementasikan (regresi, klasifikasi, klasterisasi, atau forecasting) dan memahami bagaimana implementasi tersebut tercermin dalam kode serta output awal.

2. Deskripsi Proyek

Proyek ini mengimplementasikan sistem edukasi interaktif yang menyediakan *chatbot* yang menjawab pertanyaan mahasiswa berdasarkan materi kuliah yang diproses dan

retrieval, serta menghasilkan kuis otomatis untuk menguji pemahaman, menggunakan teknologi seperti Sentence Transformers untuk pencarian semantik dan FAISS untuk *retrieval* informasi yang efisien, semuanya disajikan dalam antarmuka pengguna Streamlit yang interaktif.

3. Identifikasi Algoritma AI yang Diimplementasikan

Berdasarkan analisis kode yang diberikan (terutama file *test.py* dan *prepare_data_for_rag_py.ipynb*), algoritma AI yang diimplementasikan **bukan merupakan implementasi langsung dari algoritma klasifikasi, regresi, klusterisasi, atau *forecasting* yang berdiri sendiri dalam bentuk tradisional**. Kode ini menunjukkan implementasi sistem **Retrieval-Augmented Generation (RAG)** yang memanfaatkan model *embedding* dan *Large Language Model (LLM)* untuk membangun *chatbot* edukasi. Meskipun demikian, **konsep klasifikasi merupakan prinsip operasional yang mendasar** dalam cara sistem ini berfungsi.

3.1 Peran File Kode

a. *prepare_data_for_rag_py.ipynb*

File Jupyter Notebook ini bertanggung jawab untuk **tahap persiapan data** dalam *pipeline* RAG, *notebook* ini menyiapkan data dalam format yang dibutuhkan oleh aplikasi *chatbot*.

Fungsinya meliputi:

- Mem-parsing *outline* materi kuliah.
- Membaca konten teks materi.
- Memecah teks menjadi *chunk* yang lebih kecil.
- Menghasilkan *embeddings* untuk *chunk* teks menggunakan model **SentenceTransformer**.
- Membuat dan menyimpan indeks FAISS untuk pencarian 유사.
- Menyimpan *chunk* yang diproses dan metadatanya ke dalam file JSON (*processed_chunks_with_metadata.json*).

b. *test.py*

File Python ini berisi **logika aplikasi dan antarmuka pengguna** (menggunakan Streamlit) dan menggunakan data dari *prepare_data_for_rag_py.ipynb* yang disiapkan oleh pre untuk memberikan fungsionalitas *chatbot*.

Fungsinya meliputi

- Memuat data dan model yang telah diproses (*FAISS index*, *chunk* teks JSON, model *embedding*, model LLM).
- Menerima input pertanyaan pengguna melalui antarmuka Streamlit.
- Membuat *embedding* untuk pertanyaan pengguna.
- Melakukan *retrieval chunk* teks yang relevan dari *FAISS index*.
- Menggunakan LLM untuk menghasilkan jawaban berdasarkan *chunk* teks yang diambil.
- Mengelola histori obrolan dan interaksi pengguna.
- Membuat dan menyajikan kuis (jika diimplementasikan).

4. Implementasi Klasifikasi dalam Program

a. Klasifikasi Relevansi oleh Model *Embedding*:

- **Proses:** Model `SentenceTransformer` (diinisialisasi di *test.py* dan *prepare_data_for_rag_py.ipynb*) dilatih untuk memahami makna semantik. Ketika model ini mengubah pertanyaan dan *chunk* teks menjadi vektor *embedding*, ia mengklasifikasikan teks berdasarkan maknanya. Teks dengan makna serupa akan memiliki vektor yang berdekatan dalam ruang vektor.
- **Implementasi dalam Kode:** Pemanggilan `query_embedding_model.encode([user_query_text])[0]` (di *test.py*) menghasilkan representasi vektor dari pertanyaan.
- **Tujuan:** Mengklasifikasikan *chunk* teks sebagai "relevan" atau "tidak relevan" berdasarkan kedekatan semantik dengan pertanyaan.

b. Klasifikasi Top-k oleh FAISS

- **Proses:** *Library* FAISS (diimpor di *test.py* dan *prepare_data_for_rag_py.ipynb*) membangun indeks dari vektor *embedding chunk* teks. Pencarian *nearest neighbors* (`faiss_search_index.search()` di *test.py*) mengklasifikasikan semua *chunk* teks dalam indeks berdasarkan jarak ke vektor pertanyaan dan memilih *k* teratas sebagai "paling relevan".
- **Implementasi dalam Kode:** Fungsi `search_relevant_chunks`

(di *test.py*) menggunakan FAISS untuk mengklasifikasikan dan memilih *top-k chunk*.

- **Tujuan:** Mengklasifikasikan dan memilih sejumlah kecil *chunk* teks yang paling relevan dari basis pengetahuan.

c. Klasifikasi Informan Relevan oleh LLM (dalam Generasi)

- **Proses:** Meskipun LLM (**gemini-1.5-flash-latest**) utamanya melakukan *generasi* teks, untuk menghasilkan jawaban yang koheren dan relevan berdasarkan *context chunks*, LLM harus secara internal "memahami" dan "memilah" informasi yang penting dari yang kurang penting. Proses ini melibatkan klasifikasi informasi dalam *context chunks* sebagai "relevan untuk menjawab pertanyaan" atau "tidak relevan".
- **Implementasi dalam Kode:** Fungsi **get_rag_answer_from_llm** (di *test.py*) mengirimkan *prompt* yang berisi pertanyaan dan *context chunks* ke LLM. LLM kemudian mengklasifikasikan informasi dalam *context* untuk menghasilkan jawaban.
- **Tujuan:** Mengklasifikasikan informasi dalam *context chunks* dan memprioritaskan informasi yang paling relevan untuk menghasilkan jawaban yang akurat.

d. Klasifikasi Konsep dalam Pembuatan Soal oleh LLM

- **Proses:** Dalam fungsi **generate_mcq_from_llm** (di *test.py*), LLM diminta untuk membuat soal pilihan ganda. Untuk melakukan ini, LLM harus mengklasifikasikan informasi dalam materi pelajaran menjadi konsep utama (untuk pertanyaan), jawaban yang benar, dan opsi pengecoh yang masuk akal.
- **Implementasi dalam Kode:** *Prompt* yang dikirim ke LLM menginstruksikannya untuk mengklasifikasikan informasi dan menghasilkan format soal yang diinginkan.
- **Tujuan:** Mengklasifikasikan informasi dalam materi pelajaran ke dalam berbagai peran yang diperlukan untuk membuat soal kuis yang efektif.

e. Contoh pada codingan

Line Coding yang Mendukung Analisis:

- **Import** `SentenceTransformer` :

Python

```
from sentence_transformers import SentenceTransformer # app.py & prepare_data_for_r
```

Mengaktifkan klasifikasi relevansi semantik melalui *embeddings*.

- **Inisialisasi** `query_embedding_model` :

Python

```
resources["query_embedding_model"] = SentenceTransformer(EMBEDDING_MODEL_NAME) # ap
```

Memuat model untuk klasifikasi makna pertanyaan dan dokumen.

- **Fungsi** `search_relevant_chunks` :

Python

```
def search_relevant_chunks(query_embedding_vector, current_pertemuan_id=None, top_k  
distances, global_indices = faiss_search_index.search(query_np_array, num_to_se
```

Mengimplementasikan klasifikasi *top-k chunk* relevan menggunakan FAISS.

- **Fungsi** `get_rag_answer_from_llm` :

Python

```
def get_rag_answer_from_llm(user_query, context_chunks):  
    prompt_to_send = f"... KONTEKS MATERI: ... PERTANYAAN MAHASISWA: ...""  
    response = llm_chat_model.generate_content(prompt_to_send) # app.py
```

Menunjukkan LLM melakukan klasifikasi informasi relevan dalam konteks.

- **Fungsi** `generate_mcq_from_llm` :

Python

```
def generate_mcq_from_llm(pertemuan_id, num_questions=3):  
    prompt_quiz_generation = f"... Anda adalah seorang ahli pembuat soal ujian. ...""  
    response = llm_chat_model.generate_content(prompt_quiz_generation) # app.py
```

Menunjukkan LLM mengklasifikasikan informasi untuk pembuatan soal kuis.

5. Hasil Awal

Mata Kuliah Anda

Daftar Pertemuan:

Foundations & Overview of Operating Systems

OS Components, Services, and Structure

Process Management

CPU Scheduling Algorithms

Reset Aplikasi & Muat Ulang Resources

Semua resources aplikasi berhasil dimuat!

Pertemuan 2: OS Components, Services, and Structure

Tanya Jawab Materi Ini

Uji Pemahaman Saya

🗣️ Jelaskan materi ini

📖 Materi ini membahas sistem operasi dan layanan yang disediakan. Sistem operasi menyediakan layanan untuk pengguna dan program, termasuk dukungan untuk programmer (seperti editor, debugger, compiler, dan linker), eksekusi program, operasi I/O (input/output), manipulasi sistem file, dan komunikasi antar proses. Komunikasi meliputi pembuatan koneksi virtual antara proses, pengguna, dan sistem komputer yang berbeda. Program aplikasi, seperti browser web dan email client, juga merupakan bagian dari sistem operasi.

Sistem operasi juga menyediakan system call, yang merupakan antarmuka antara program yang berjalan dan kernel. System call memungkinkan program untuk meminta layanan dari sistem operasi, seperti kontrol proses (membuat, mengakhiri, dan mengelola proses), manipulasi file (membuat, menghapus, membaca, dan menulis file), manipulasi perangkat (berinteraksi dengan perangkat keras, informasi lingkungan (mendapatkan informasi sistem), dan komunikasi (bertukar informasi antara proses).

Struktur internal sistem operasi bervariasi, termasuk struktur sederhana (seperti MS-DOS), sistem monolitik (seperti UNIX tradisional dan Linux), dan sistem berlapis (informasi detail tentang sistem berlapis tidak tersedia dalam materi ini).

Ketik pertanyaan Anda di sini...

Mata Kuliah Anda

Daftar Pertemuan:

Foundations & Overview of Operating Systems

OS Components, Services, and Structure

Process Management

CPU Scheduling Algorithms

Reset Aplikasi & Muat Ulang Resources

Semua resources aplikasi berhasil dimuat!

Pertemuan 2: OS Components, Services, and Structure

Tanya Jawab Materi Ini

Uji Pemahaman Saya

📖 Kuis Pemahaman

Soal 1 dari 3:

Manakah dari berikut ini BUKAN merupakan tanggung jawab Sistem Operasi dalam manajemen proses?

Pilih jawaban Anda:

☐ A. Membuat dan menghapus proses

☒ B. Menyerjemahkan kode sumber menjadi kode mesin

☐ C. Menunda dan melanjutkan proses

☐ D. Menangani deadlock

Soal Sebelumnya

Soal Berikutnya

Batalkan Kuis

Mata Kuliah Anda

Daftar Pertemuan:

Foundations & Overview of Operating Systems

OS Components, Services, and Structure

Process Management

CPU Scheduling Algorithms

Reset Aplikasi & Muat Ulang Resources

Hasil Uji Pemahaman Anda

Skor Anda

33.33%

📈 1 benar dari 3 soal

⚠️ Anda perlu mempelajari lagi beberapa bagian.

Topik yang Perlu Diperdalam:

System Call (1 kesalahan)

Tanya tentang: System Call

Dukungan untuk Programmer (1 kesalahan)

Tanya tentang: Dukungan untuk Programmer

Detail Jawaban:

Soal 1: Manakah dari berikut ini BUKAN merupakan tanggung jawab Sistem Operasi dalam manajemen proses?

Soal 2: Sistem panggilan (system call) berfungsi sebagai:

Soal 3: Manakah dari layanan berikut yang TIDAK termasuk dalam kategori layanan yang disediakan oleh Sistem Operasi untuk mendukung programmer?

Kembali ke Pilihan Aksi