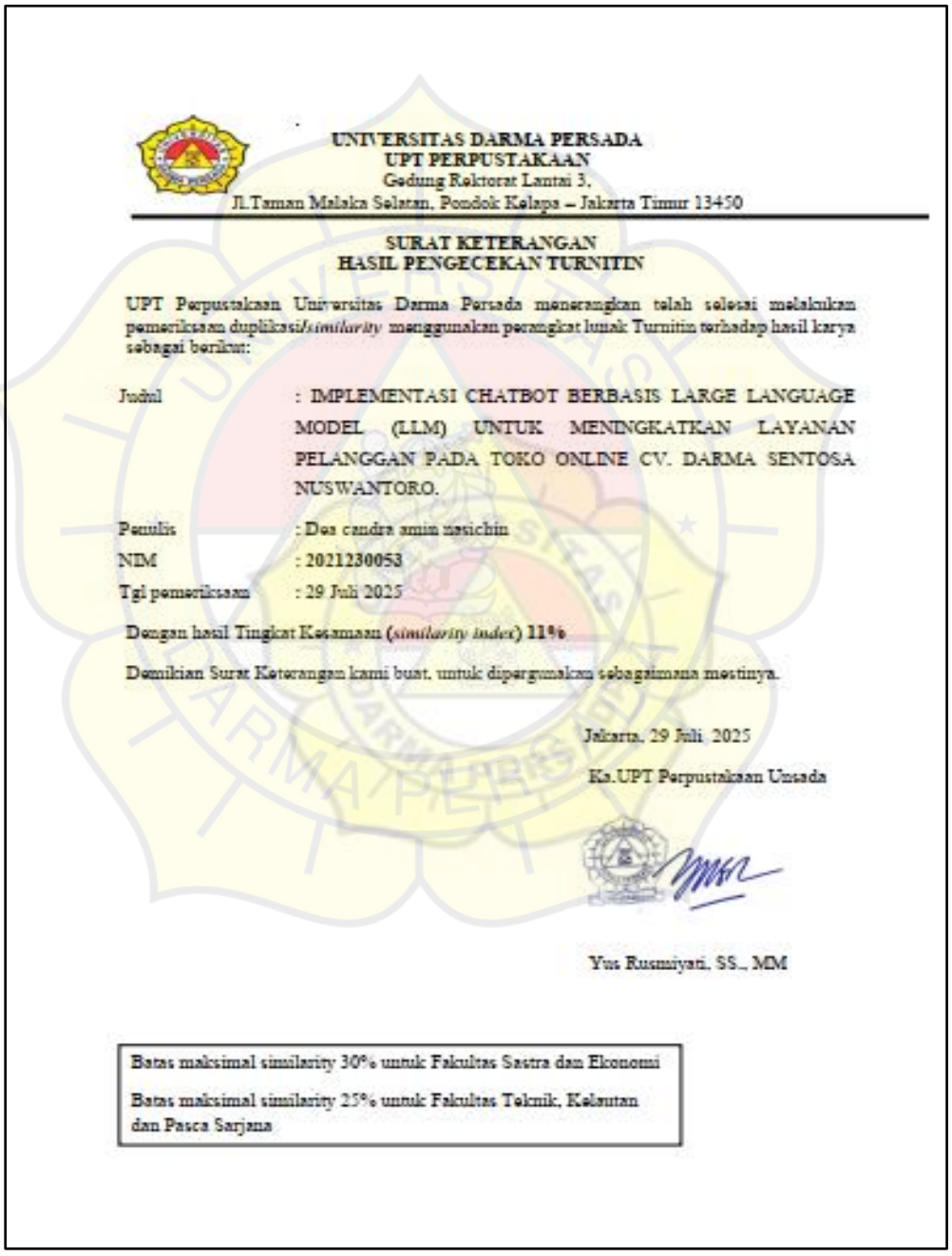
LAMPIRAN

Lampiran 1. Surat Keterangan Bebas Plagiat



UNIVERSITAS DARMA PERSADA
UPT PERPUSTAKAAN
Gedung Rektorat Lantai 3,
Jl.Taman Malaka Selatan, Pondok Kelapa – Jakarta Timur 13450

SURAT KETERANGAN
HASIL PENGECEKAN TURNITIN

UPT Perpustakaan Universitas Darma Persada menerangkan telah selesai melakukan pemeriksaan duplikasi/*similarity* menggunakan perangkat lunak Turnitin terhadap hasil karya sebagai berikut:

Judul : IMPLEMENTASI CHATBOT BERBASIS LARGE LANGUAGE MODEL (LLM) UNTUK MENINGKATKAN LAYANAN PELANGGAN PADA TOKO ONLINE CV. DARMA SENTOSA NUSWANTORO.

Penulis : Dea candra amin nasichin
NIM : 2021230053
Tgl pemeriksaan : 29 Juli 2025

Dengan hasil Tingkat Kesamaan (*similarity index*) **11%**

Demikian Surat Keterangan kami buat, untuk dipergunakan sebagaimana mestinya.

Jakarta, 29 Juli 2025

Ka.UPT Perpustakaan Unsada

Yus Rusmiyati, SS., MM

Batas maksimal similarity 30% untuk Fakultas Sastra dan Ekonomi

Batas maksimal similarity 25% untuk Fakultas Teknik, Kelautan dan Pasca Sarjana

Lampiran 2. *Source Code* (Llama Dengan RAG)

```
!pip install -U langchain langchain-chroma chromadb sentence-
transformers pypdf langchain_community

from sentence_transformers import SentenceTransformer

import os

drive_path = "/content/drive/MyDrive/projekku/model-
embedding/all-mpnet-base-v2"

os.makedirs(drive_path, exist_ok=True)

SentenceTransformer("sentence-transformers/all-mpnet-base-
v2").save(drive_path)

# versi qna lebel

import json

from collections import OrderedDict

from langchain.schema import Document

from langchain.document_loaders import PyPDFLoader

from langchain.text_splitter import
RecursiveCharacterTextSplitter

from langchain.embeddings import HuggingFaceEmbeddings

from langchain.vectorstores import Chroma

from google.colab import userdata  # Hanya jika di Google Colab

token = userdata.get("HF_TOKEN")

# === 1. LOAD PDF DOKUMEN ===

pdf_path = "dsn.pdf"

loader = PyPDFLoader(pdf_path)
```

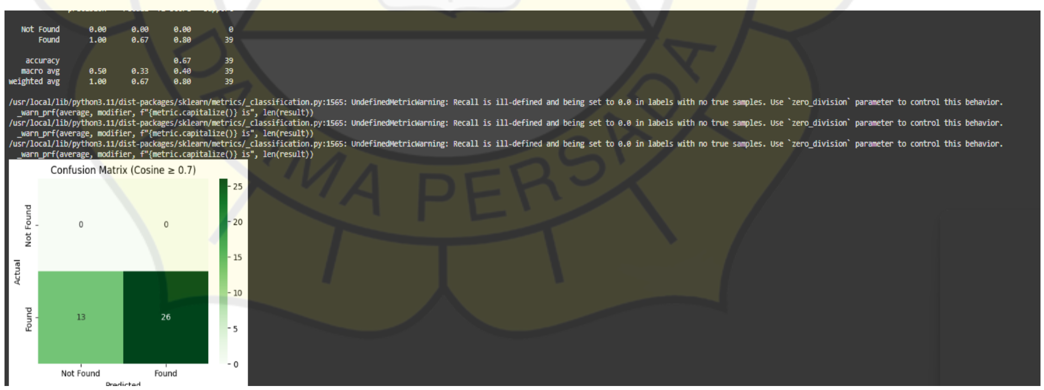
```
documents = loader.load()

# === 2. LOAD FILE QnA JSON ===

faq_documents = []

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

    qna_data = json.load(f)

for item in qna_data:

    for question in item["questions"]:

        content = f"Q: {question}\nA: {item['answer']}"

        metadata = {

            "source": item.get("source",
"qna_rag_cvdarma.json"),

            "label": item.get("label", ""),

            "page": item.get("page", ""),

            "context": item.get("context", "")

        }

        faq_documents.append(Document(page_content=content,
metadata=metadata))

# === 3. GABUNGKAN PDF + QNA ===

combined_documents = documents + faq_documents

# === 4. SPLIT MENJADI POTONGAN KECIL ===

text_splitter = RecursiveCharacterTextSplitter(chunk_size=500,
chunk_overlap=100)

texts = text_splitter.split_documents(combined_documents)

# === 5. BUAT EMBEDDING DAN SIMPAN KE CHROMA ===
```

```
drive_path = "/content/drive/MyDrive/projekku/model-
embedding/all-mpnet-base-v2"

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

embedding_model = HuggingFaceEmbeddings(model_name=drive_path)

try:

    db = Chroma.from_documents(

        documents=texts,

        embedding=embedding_model,

        persist_directory="/content/data"

    )

    db.persist()

    print(" Vectorstore Chroma berhasil dibuat dan disimpan!")
except Exception as e:

    print(f" Gagal membuat vectorstore: {e}")


# === 6. TEST PENCARIAN DOKUMEN ===

try:

    test_query = "Apakah ada garansi untuk produk saya?"

    search_results = db.similarity_search(test_query, k=3)

    unique_results = OrderedDict()

    for doc in search_results:

        if doc.page_content not in unique_results:
```

```
 unique_results[doc.page_content] = doc

    final_results = list(unique_results.values())

    print("\n Hasil pencarian untuk query:")

    for i, result in enumerate(final_results):

        print(f"\n--- Hasil {i+1} ---")

        print(result.page_content)

except Exception as e:

    print(f" Gagal melakukan pencarian: {e}")

# Cosine Similarity (MPNet)

from sklearn.metrics.pairwise import cosine_similarity

def evaluate_confusion_cosine_all(qna_data, vectorstore,
embedding_model, k=3, threshold=0.70):

    y_true = []

    y_pred = []

    print(f"\n Evaluasi Cosine Similarity Retrieval (Top-{k},
threshold={threshold})")

    print("=" * 60)

    for item in qna_data:

        expected = item["answer"].strip()

        # Embed expected answer

        expected_vec = embedding_model.embed_query(expected)


        for question in item["questions"]:
```

```
   results = vectorstore.similarity_search(question, k=k)
            found = False
            for doc in results:
                doc_vec =
embedding_model.embed_query(doc.page_content)
                score = cosine_similarity([expected_vec],
[doc_vec])[0][0]
                if score >= threshold:
                    found = True
                    break
            y_true.append(1)
            y_pred.append(1 if found else 0)
            print(f"{'V' if found else 'X'} Q: {question}")
            print(f"   Similarity ≥ {threshold}: {found}")
            print("-" * 60)
    # Confusion Matrix dan Laporan
    from sklearn.metrics import confusion_matrix,
classification_report
    import matplotlib.pyplot as plt
    import seaborn as sns
    cm = confusion_matrix(y_true, y_pred)
    labels = ["Not Found", "Found"]
    print("\n Confusion Matrix:")
    print(cm)
```

```
    print("\n Classification Report:")

    print(classification_report(y_true, y_pred,
target_names=labels))

    # Heatmap

    plt.figure(figsize=(5, 4))

    sns.heatmap(cm, annot=True, fmt='d', cmap='Greens',
xticklabels=labels, yticklabels=labels)

    plt.xlabel("Predicted")

    plt.ylabel("Actual")

    plt.title(f"Confusion Matrix (Cosine ≥ {threshold})")

    plt.show()

evaluate_confusion_cosine_all(qna_data, db, embedding_model,
k=3, threshold=0.70)
```




```
# Lokasi data vektor Chroma

data_directory = "/content/data"

# Inisialisasi embedding dan vectorstore Chroma

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

```
drive_path = "/content/drive/MyDrive/projekku/model-
embedding/all-mpnet-base-v2"

embedding_model = HuggingFaceEmbeddings(model_name=drive_path )

vector_store = Chroma(embedding_function=embedding_model,
persist_directory=data_directory)

import os

# masukan model llm

import torch

from transformers import AutoTokenizer, AutoModelForCausalLM,
pipeline

from langchain.llms import HuggingFacePipeline

from langchain.embeddings import HuggingFaceEmbeddings

# from langchain.vectorstores import Chroma

from langchain_chroma import Chroma

from google.colab import userdata

token = userdata.get("HF_TOKEN")

model_id = "meta-llama/Meta-Llama-3.1-8B-Instruct"

model_drive_path = "/content/drive/MyDrive/llama3-8b-instruct"

if not os.path.exists(model_drive_path):

    print(" Model belum ada, mendownload dan menyimpan ke Google
Drive...")

    tokenizer = AutoTokenizer.from_pretrained(model_id,
token=token)

    tokenizer.save_pretrained(model_drive_path)
```

```
    model = AutoModelForCausalLM.from_pretrained(
        model_id,
        token=token,
        torch_dtype=torch.float16,
        device_map="cpu",
    )
    model.save_pretrained(model_drive_path)
else:
    print(" Model sudah tersimpan di Google Drive.")

tokenizer = AutoTokenizer.from_pretrained(model_drive_path)
if tokenizer.pad_token_id is None:
    tokenizer.pad_token = tokenizer.eos_token
model = AutoModelForCausalLM.from_pretrained(
    model_id,
    token=token,
    torch_dtype=torch.float16,
    device_map="auto",
    low_cpu_mem_usage=True
# kurangi RAM yang digunakan saat loading
)
llama_pipeline = pipeline(
    "text-generation",
```

```
    model=model,
    tokenizer=tokenizer,
    max_new_tokens=80,
#untuk batasi genaration yang di keluarkan
    pad_token_id=tokenizer.eos_token_id,
# cegah warning pad_token_id
    eos_token_id=tokenizer.eos_token_id,
# pastikan eos_token_id benar
    repetition_penalty=1.1,
    temperature=0.3,
    # do_sample=False,
    return_full_text=False
 # hanya kembalikan hasil generasi, bukan input+output)
llm_llama3 = HuggingFacePipeline(pipeline=llama_pipeline)
# Template prompt khusus chatbot CV Darma Sentosa Nuswantoro
from langchain.prompts import PromptTemplate
prompt_template = """
Sebagai asisten cerdas yang berfokus pada layanan dan produk
perusahaan CV Darma Sentosa Nuswantoro,
tugasmu adalah menjawab pertanyaan secara akurat berdasarkan
informasi dalam database internal perusahaan.
Ikuti pedoman ini saat memberikan jawaban:
```

1. Jawaban harus relevan dan berdasarkan informasi yang tersedia dari dokumen perusahaan.

2. Fokus pada topik berikut:

- Layanan pengadaan sparepart dan peralatan berbahan logam

- Proses produksi atau pengerjaan barang logam

- Informasi perusahaan seperti sejarah, lokasi, tahun berdiri, visi-misi, dan lainnya

3. Jika pertanyaan tidak relevan dengan bidang layanan perusahaan, tolong beritahu pengguna dengan sopan bahwa pertanyaan tersebut di luar cakupan asisten.

4. Gunakan bahasa yang jelas, sopan, dan langsung ke poin.

5. Batasi jawaban maksimal 1 kalimat. Hindari kalimat pada pedoman ini dan fokus saja pada jawaban yang ditanyakan saja.dan jangan berikan opsi jawabn lain.

6. jika ada yang mengunakan kalimat sapaan tolong jelaskan ada asisten cerdas.

7. jangan sebutkan pedoman dan jangan munculkan pedoman saat menjawab.

8. jangan jawab diluar data yang ada.

9. Jangan menyebut kembali bahwa Anda adalah asisten cerdas dalam jawaban utama, kecuali diminta secara langsung.

10. Hindari menjawab hal yang sama lebih dari sekali atau dengan kalimat berbeda yang artinya sama.

Konteks:

{context}

```
Pertanyaan: {question}

Jawaban:

"""

custom_prompt = PromptTemplate(

    template=prompt_template,

    input_variables=["context", "question"]

)

from langchain.chains import RetrievalQA

# Buat chain RAG dengan RetrievalQA

rag_chain = RetrievalQA.from_chain_type(

    llm=llm_llama3,

    chain_type="stuff",

    retriever=vector_store.as_retriever(top_k=3),

    chain_type_kwargs={"prompt": custom_prompt})

def get_response(question: str) -> str:

    try:

        result = rag_chain.invoke({"query": question})

        output = result["result"].strip()

        first_line =
output.split("Pertanyaan:")[0].split("###")[0].strip()

        if not first_line:
```

```
return " Maaf, saya belum menemukan jawaban untuk pertanyaan
tersebut."
        return first_line
    except Exception as e:
        return f" Maaf, terjadi kesalahan saat memproses
pertanyaan: {str(e)}"
# === CLI Mode Interaktif ===
if __name__ == "__main__":
    print("=== Selamat datang di dsnBot ===")
    print("Tanya apa pun tentang layanan CV Darma Sentosa
Nuswantoro!")
    print("Ketik 'exit' untuk keluar.\n")
    while True:
        user_input = input(" Kamu: ")
        if user_input.lower() in ['exit', 'keluar']:
            print(" dsnBot: Sampai jumpa! Terima kasih sudah
menghubungi kami!")
            break
        response = get_response(user_input)
        print(f" dsnBot: {response}\n")
```

Lampiran 3. *Source Code* (GPT 3.5 Dengan RAG)

```
!pip install langchain langchain-openai langchain-community
chromadb

!pip install pypdf

# === Import Library ===

import os

import json

from collections import OrderedDict

from langchain.schema import Document

from langchain.document_loaders import PyPDFLoader

from langchain.text_splitter import
RecursiveCharacterTextSplitter

from langchain_openai import OpenAIEmbeddings

from langchain_chroma import Chroma

from google.colab import userdata  # khusus Google Colab

# === Ambil OpenAI API Key dari userdata atau os.environ ===

token = userdata.get("OPENAI_API_KEY")  # kamu simpan di Colab
via secrets

if not token:

    raise ValueError("OPENAI_API_KEY belum tersedia di secrets
Colab.")

os.environ["OPENAI_API_KEY"] = token  # wajib untuk
OpenAIEmbeddings

# === 1. Load dokumen PDF ===

pdf_path = "dsn.pdf"  # Pastikan sudah di-upload
```

```
loader = PyPDFLoader(pdf_path)

documents = loader.load()

# === 2. Load file QnA JSON ===

faq_documents = []

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

    qna_data = json.load(f)


for item in qna_data:

    for question in item["questions"]:

        content = f"Q: {question}\nA: {item['answer']}"

        metadata = {

            "source": item.get("source", "qna.json"),

            "label": item.get("label", ""),

            "page": item.get("page", ""),

            "context": item.get("context", "")

        }

        faq_documents.append(Document(page_content=content,
metadata=metadata))

# === 3. Gabungkan PDF + QnA ===

combined_documents = documents + faq_documents

# === 4. Split dokumen menjadi chunks ===

text_splitter = RecursiveCharacterTextSplitter(chunk_size=250,
chunk_overlap=20)
```

```
texts = text_splitter.split_documents(combined_documents)

# === 5. Inisialisasi OpenAI Embeddings dan simpan ke Chroma ===

embedding_model = OpenAIEmbeddings(

    openai_api_key=os.environ["OPENAI_API_KEY"]

)

try:

    db = Chroma.from_documents(

        documents=texts,

        embedding=embedding_model,

        persist_directory="/content/data"

    )

    db.persist()

    print(" Vectorstore Chroma berhasil dibuat dan disimpan!")

except Exception as e:

    print(f" Gagal membuat vectorstore: {e}")

# === 6. Uji pencarian dokumen ===

try:

    test_query = "Apakah ada garansi untuk produk saya?"

    search_results = db.similarity_search(test_query, k=3)



    unique_results = OrderedDict()

    for doc in search_results:

        if doc.page_content not in unique_results:
```

```
    unique_results[doc.page_content] = doc

    final_results = list(unique_results.values())

    print("\n Hasil pencarian untuk query:")

    for i, result in enumerate(final_results):

        print(f"\n--- Hasil {i+1} ---")

        print(result.page_content)

except Exception as e:

    print(f" Gagal melakukan pencarian: {e}")


import os

from langchain_community.vectorstores import Chroma

from langchain.prompts import PromptTemplate

from langchain.chains import RetrievalQA

from langchain_openai import OpenAIEmbeddings, ChatOpenAI

import warnings

warnings.filterwarnings("ignore")

# Ambil API Key dari environment (pastikan sudah di-set
sebelumnya)

token = os.getenv("OPENAI_API_KEY")

if not token:

    raise ValueError(" API key OpenAI belum di-set! Silakan set
environment variable OPENAI_API_KEY terlebih dahulu.")

os.environ["OPENAI_API_KEY"] = token
```

```
# Lokasi data vektor Chroma

data_directory = "/content/data"

# Inisialisasi embedding dan vectorstore Chroma

embedding_model = OpenAIEmbeddings()

vector_store = Chroma(embedding_function=embedding_model,
persist_directory=data_directory)

# Inisialisasi LLM OpenAI

openai_llm = ChatOpenAI(

    temperature=0.7,

    model_name="gpt-3.5-turbo",

    max_tokens=512

)

# Template prompt khusus chatbot CV Darma Sentosa Nuswantoro

prompt_template = """

Sebagai asisten cerdas yang berfokus pada layanan dan produk
perusahaan CV Darma Sentosa Nuswantoro,

tugasmu adalah menjawab pertanyaan secara akurat berdasarkan
informasi dalam database internal perusahaan.


Ikuti pedoman ini saat memberikan jawaban:

1. Jawaban harus relevan dan berdasarkan informasi yang tersedia
dari dokumen perusahaan.

2. Fokus pada topik berikut:

    - Layanan pengadaan sparepart dan peralatan berbahan logam
```

```
    - Proses produksi atau pengerjaan barang logam

    - Informasi perusahaan seperti sejarah, lokasi, tahun
berdiri, visi-misi, dan lainnya

3. Jika pertanyaan tidak relevan dengan bidang layanan
perusahaan, tolong beritahu pengguna dengan sopan bahwa
pertanyaan tersebut di luar cakupan asisten.

4. Gunakan bahasa yang jelas, sopan, dan langsung ke poin.
Hindari ucapan penutup yang tidak perlu.

5. Batasi jawaban maksimal 3 kalimat. Jangan menyimpulkan atau
menyebut informasi yang tidak ada dalam dokumen.

Konteks:

{context}

Pertanyaan: {question}

Jawaban:

"""

custom_prompt = PromptTemplate(

    template=prompt_template,

    input_variables=["context", "question"]

)

# Buat chain RAG dengan RetrievalQA

rag_chain = RetrievalQA.from_chain_type(

    llm=openai_llm,

    chain_type="stuff",

    retriever=vector_store.as_retriever(top_k=3),
```

```
 chain_type_kwargs={"prompt": custom_prompt}

)

def get_response(question: str) -> str:

    try:

        result = rag_chain({"query": question})

        return result["result"].strip()

    except Exception as e:

        return f" Maaf, terjadi kesalahan saat memproses
pertanyaan: {str(e)}"


# CLI interaktif

if __name__ == "__main__":

    print("=== Selamat datang di dsnBot ===")

    print("Tanya apa pun tentang kami dan layanan!\nKetik 'exit'
untuk keluar.\n")

    while True:

        user_input = input("\U0001F464 Kamu: ")

        if user_input.lower() in ['exit', 'keluar']:

            print("\U0001F457 dsnBot: Sampai jumpa! Terima kasih
sudah menghubungi kami!")

            break

        response = get_response(user_input)

        print(f"\U0001F457 dsnBot: {response}\n")
```

Lampiran 4. Biodata Penulis

Saya lahir di Temanggung pada 03 Oktober 2000 sebagai anak pertama dalam keluarga sederhana yang selalu mendukung pendidikan dan perkembangan diri saya. Sejak kecil, saya tumbuh dalam lingkungan yang mendorong rasa ingin tahu dan semangat belajar. Dukungan dari keluarga menjadi fondasi penting dalam perjalanan pendidikan dan kehidupan saya.

Pendidikan formal saya dimulai di sekolah dasar di Kota Temanggung, kemudian dilanjutkan ke jenjang SMP dan SMK di daerah yang sama. Setelah lulus dari SMK, pada awalnya saya merantau ke Jakarta yang diawali tekat kuat untuk bekerja. Adapun setelah saya bekerja selama 2 tahun, pada saat itu saya merasasakan cita" terpendam saya untuk melanjutkan studi ke jenjang perguruan tinggi. Di karenakan finasial saya yang cukup baik saya memutuskan pada tahun 2021 untuk melanjutkan cita" saya dan ditahun yang sama, saya diterima sebagai mahasiswa di Universitas Darma Persada (UNSADA), program Studi Teknologi Informasi, Fakultas Teknik.

Selama masa perkuliahan, saya tertarik pada bidang artificial intelligence, khususnya pemrosesan bahasa alami (Natural Language Processing) dan sistem cerdas. Ketertarikan tersebut mendorong saya untuk mengambil topik skripsi tentang chatbot berbasis Large Language Model (LLM) dengan pendekatan Retrieval-Augmented Generation (RAG). Penelitian ini saya lakukan di CV Darma Sentosa Nuswantoro sebagai bagian dari tugas akhir.

Melalui penelitian ini, saya ingin mengaplikasikan teknologi LLM untuk menjawab tantangan layanan pelanggan pada toko online. Saya berharap hasil

dari penelitian ini dapat menjadi langkah awal dalam pengembangan teknologi layanan pelanggan yang lebih efisien dan bermanfaat bagi pelaku usaha, serta memberi kontribusi terhadap perkembangan ilmu di bidang Teknologi Informasi.