INTERACTION LOGS

Date: 2025-09-16

User: I want to allow users to summarize multiple PDFs at once and save the summaries automatically in _summary.txt files.

ChatGPT: Here’s a refined version matching your project structure:

```
from summarizer import summarize_pdf_safe
import os


def main():
    pdf_paths = input("Enter PDF paths separated by commas: ").split(",")
    for idx, pdf_path in enumerate(pdf_paths, 1):
        pdf_path = pdf_path.strip()
        print(f"Processing PDF {idx} of {len(pdf_paths)}: {pdf_path}...")
        summary = summarize_pdf_safe(pdf_path)
        output_file = os.path.splitext(pdf_path)[0] + "_summary.txt"
        with open(output_file, "w", encoding="utf-8") as f:
            f.write(summary)
        print(f"✅ Summary saved to {output_file}")


if __name__ == "__main__":
    main()
```

User: Can we also log success/failure for each PDF to summarizer.log?

ChatGPT: Updated main.py with logging:

```
import logging
from summarizer import summarize_pdf_safe
import os


logging.basicConfig(filename="summarizer.log",
                    level=logging.INFO,
                    format="%(asctime)s - %(levelname)s - %(message)s")
```

```
def main():
    pdf_paths = input("Enter PDF paths separated by commas: ").split(",")
    for idx, pdf_path in enumerate(pdf_paths, 1):
        pdf_path = pdf_path.strip()
        try:
            print(f"Processing PDF {idx} of {len(pdf_paths)}: {pdf_path}...")
            summary = summarize_pdf_safe(pdf_path)
            output_file = os.path.splitext(pdf_path)[0] + "_summary.txt"
            with open(output_file, "w", encoding="utf-8") as f:
                f.write(summary)
            print(f"✅ Summary saved to {output_file}")
            logging.info(f"Processed {pdf_path} successfully")
        except Exception as e:
            print(f"❌ Failed to process {pdf_path}: {e}")
            logging.error(f"Failed {pdf_path}: {e}")


if __name__ == "__main__":
    main()
```

---

2. summarizer.py – Preprocessing & chunked summarization

Date: 2025-09-16
User: Sometimes my PDFs are huge and the summarizer pipeline crashes. Can we split text into chunks and summarize each chunk?

ChatGPT: Here’s a version that matches your summarizer.py structure:

```
import re
from transformers import pipeline


def preprocess_text(text):
    # Remove special characters except newlines
    text = re.sub(r'[^\w\s\n]', '', text)
    # Replace multiple spaces with single space
    text = re.sub(r'[^\S\n]+', ' ', text)
    # Remove hyphenation at line breaks
    text = re.sub(r'-\n', '', text)
    return "\n".join(line.strip() for line in text.splitlines())


def chunk_text(text, max_words=500):
    words = text.split()
    for i in range(0, len(words), max_words):
        yield " ".join(words[i:i+max_words])


def summarize_text(text, max_length=150):
    summarizer = pipeline("summarization")
    summary = summarizer(text, max_length=max_length, min_length=50, do_sample=False)
    return summary[0]['summary_text']


def summarize_pdf_safe(pdf_path):
    from pdf_extractor import extract_text_from_pdf  # your extractor module
    try:
        text = extract_text_from_pdf(pdf_path)
        text = preprocess_text(text)
        summaries = [summarize_text(chunk) for chunk in chunk_text(text)]
        return "\n".join(summaries)
    except Exception as e:
        return f"Error processing PDF: {e}"
```

User: Can we dynamically increase max_length if the summary is too short?

ChatGPT: Yes, add this:

```
def summarize_text_dynamic(text, min_words=50, max_length=150):
    summary = summarize_text(text, max_length=max_length)
    if len(summary.split()) < min_words:
        summary = summarize_text(text, max_length=max_length*2)
    return summary
```

---

3. Model inference with adapter

Date: 2025-09-16
User: I want to use my fine-tuned LoRA adapter model for inference in the pipeline.

ChatGPT: Here’s an example using your project structure:

```
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM
from peft import PeftModel


MODEL_NAME = "t5-small"
ADAPTER_PATH = "./finetuned_model/adapter_model"


tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
base_model = AutoModelForSeq2SeqLM.from_pretrained(MODEL_NAME)
model = PeftModel.from_pretrained(base_model, ADAPTER_PATH)


def summarize_text_adapter(text, max_length=150):
    inputs = tokenizer(text, return_tensors="pt", truncation=True)
    summary_ids = model.generate(inputs["input_ids"], max_length=max_length)
    return tokenizer.decode(summary_ids[0], skip_special_tokens=True)
```

User: Can we replace the summarize_text function in summarizer.py with this adapter version?

ChatGPT: Yes, just change the call in summarize_pdf_safe:

```
summaries = [summarize_text_adapter(chunk) for chunk in chunk_text(text)]
```

Date: 2025-09-15
User: Sometimes, my PDFs have images or tables that mess up text extraction. Can we improve extract_text_from_pdf to ignore images?

ChatGPT: Sure, using PyMuPDF, you can only extract text blocks:

```
def extract_text_from_pdf(pdf_path):
    import fitz
    doc = fitz.open(pdf_path)
    text = ""
    for page in doc:
        blocks = page.get_text("blocks")  # Extract text blocks only
        for block in blocks:
            text += block[4] + "\n"  # block[4] is the text
    return text
```

User: Some PDFs have hyphenated words across lines. Can we fix that?

ChatGPT: Yes! Remove hyphenation across line breaks:

```
def preprocess_text(text):
    import re
    text = re.sub(r'-\n', '', text)  # fix hyphenation
    text = re.sub(r'[^\w\s\n]', '', text)
    text = re.sub(r'[^\S\n]+', ' ', text)
    return "\n".join(line.strip() for line in text.splitlines())
```

---

2. Chunking & summarization improvements

Date: 2025-09-16
User: My text is too long for the summarizer pipeline; it sometimes fails.

ChatGPT: Use chunking and summarize each chunk separately:

```
def summarize_long_text(text, max_words=500):
    summaries = []
    for chunk in chunk_text(text, max_words):
        summaries.append(summarize_text(chunk))
    return "\n".join(summaries)
```

User: Can we automatically detect if the summary is too short and adjust max_length?

ChatGPT: Yes, dynamic adjustment:

```
def summarize_text_dynamic(text, min_summary_len=50, max_summary_len=150):
    summary = summarize_text(text, max_length=max_summary_len)
    if len(summary.split()) < min_summary_len:
        summary = summarize_text(text, max_length=max_summary_len*2)
    return summary
```

---

3. utils.py – Logging and error handling

Date: 2025-09-16
User: I want logs for each PDF processed and errors if it fails.

ChatGPT: Add logging:

```
import logging


logging.basicConfig(filename='summarizer.log', level=logging.INFO,
                    format='%(asctime)s - %(levelname)s - %(message)s')


def log_summary(pdf_path, summary):
    logging.info(f"Processed {pdf_path}, summary length: {len(summary)} words")
```

User: Can we wrap PDF processing in try-except to avoid crashing?

ChatGPT: Sure:

```
def summarize_pdf_safe(pdf_path):
    try:
        summary = summarize_pdf(pdf_path)
        log_summary(pdf_path, summary)
        return summary
    except Exception as e:
        logging.error(f"Failed processing {pdf_path}: {e}")
        return "Error processing PDF"
```

---

4. main.py – CLI improvements

Date: 2025-09-17
User: Can we let users summarize multiple PDFs at once?

ChatGPT: Yes, using a loop:

```
def main():
    pdf_paths = input("Enter PDF paths separated by comma: ").split(",")
    for pdf_path in pdf_paths:
        pdf_path = pdf_path.strip()
        summary = summarize_pdf_safe(pdf_path)
        output_file = os.path.splitext(pdf_path)[0] + "_summary.txt"
        with open(output_file, "w", encoding="utf-8") as f:
            f.write(summary)
        print(f"Summary saved to {output_file}")
```

---

5. Model fine-tuning refinements

Date: 2025-09-17
User: I want to save only the adapter weights and be able to reload them later.

ChatGPT: Use PEFT adapter save/load:

```
from peft import PeftModel


# Save adapter
model.save_pretrained("./adapter_model")


# Load adapter later
adapter_model = PeftModel.from_pretrained(model, "./adapter_model")
```

User: Can we evaluate the model on a validation set and print ROUGE score?

ChatGPT: Sure, example using datasets and evaluate:

```
import evaluate
rouge = evaluate.load("rouge")


def evaluate_model(model, val_texts, val_summaries):
   predictions = [summarize_text_dynamic(text) for text in val_texts]
   results = rouge.compute(predictions=predictions, references=val_summaries)
   print(results)
```

Date: 2025-09-14
User: I want a separate config file to store paths, model names, and hyperparameters.

ChatGPT: Here’s an example:

```
# config.py
MODEL_NAME = "t5-small"
TRAIN_BATCH_SIZE = 2
EVAL_BATCH_SIZE = 2
NUM_EPOCHS = 2
MAX_INPUT_LENGTH = 512
MAX_SUMMARY_LENGTH = 150
DATA_DIR = "./data"
OUTPUT_DIR = "./finetuned_model"
```

```
LOG_FILE = "./summarizer.log"
```

User: Can we also add a flag to enable/disable logging?

ChatGPT: Updated config:

```
ENABLE_LOGGING = True
```

---

2. dataset.py – Loading notes dataset

Date: 2025-09-14
User: I want to create a dataset class for PyTorch that handles notes and summaries.

ChatGPT: Example:

```
import torch
from transformers import AutoTokenizer
from config import DATA_DIR, MODEL_NAME


tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)


class NotesDataset(torch.utils.data.Dataset):
    def __init__(self, notes, summaries, max_input_len=512, max_output_len=150):
        self.notes = notes
        self.summaries = summaries
        self.max_input_len = max_input_len
        self.max_output_len = max_output_len


    def __len__(self):
        return len(self.notes)


    def __getitem__(self, idx):
        inputs = tokenizer(self.notes[idx], max_length=self.max_input_len, truncation=True, padding="max_length", return_tensors="pt")
        labels = tokenizer(self.summaries[idx], max_length=self.max_output_len, truncation=True, padding="max_length", return_tensors="pt")
        return {key: val.squeeze() for key, val in inputs.items()}, labels["input_ids"].squeeze()
```

User: Can we handle empty notes gracefully?

ChatGPT: Sure, just return a placeholder text:

```
note = self.notes[idx] if self.notes[idx] else "No content"
```

---

3. training.py – Fine-tuning with adapters

Date: 2025-09-14
User: I want to fine-tune the model on my notes dataset using LoRA adapters.

ChatGPT: Refined loop with PEFT:

```
from transformers import AutoModelForSeq2SeqLM, Trainer, TrainingArguments
from peft import get_peft_model, LoraConfig, TaskType
from dataset import NotesDataset
from config import MODEL_NAME, NUM_EPOCHS, TRAIN_BATCH_SIZE, OUTPUT_DIR

model = AutoModelForSeq2SeqLM.from_pretrained(MODEL_NAME)

config = LoraConfig(
    task_type=TaskType.SEQ_2_SEQ_LM,
    r=16,
    lora_alpha=32,
    target_modules=["q", "v"],
    lora_dropout=0.1
)
model = get_peft_model(model, config)

train_dataset = NotesDataset(notes=["Sample note"], summaries=["Sample summary"])
training_args = TrainingArguments(
    output_dir=OUTPUT_DIR,
    num_train_epochs=NUM_EPOCHS,
    per_device_train_batch_size=TRAIN_BATCH_SIZE,
    save_steps=10,
    logging_steps=5
)
```

```
trainer = Trainer(model=model, args=training_args, train_dataset=train_dataset)
trainer.train()
model.save_pretrained(OUTPUT_DIR)
```

User: Can we add evaluation after each epoch?

ChatGPT: Yes, using compute_metrics callback:

```
from evaluation import compute_rouge
trainer = Trainer(model=model, args=training_args, train_dataset=train_dataset, compute_metrics=compute_rouge)
```

---

4. evaluation.py – ROUGE scoring

Date: 2025-09-14
User: I want to compute ROUGE scores on a validation set.

ChatGPT: Example:

```
import evaluate
rouge = evaluate.load("rouge")


def compute_rouge(eval_pred):
   predictions, references = eval_pred
   decoded_preds = [pred for pred in predictions]
   decoded_refs = [ref for ref in references]
   result = rouge.compute(predictions=decoded_preds, references=decoded_refs)
   return result
```

User: Can we print a nicely formatted table?

ChatGPT: Yes, using pandas:

```
import pandas as pd
def print_rouge_table(results):
   df = pd.DataFrame(results)
   print(df.to_string(index=False))
```

---

5. visualize.py – Optional summary stats

Date: 2025-09-14
User: I want to visualize summary lengths and word counts.

ChatGPT: Example:

```
import matplotlib.pyplot as plt


def plot_summary_lengths(summaries):
    lengths = [len(s.split()) for s in summaries]
    plt.hist(lengths, bins=10, color='skyblue')
    plt.xlabel("Summary length (words)")
    plt.ylabel("Frequency")
    plt.title("Distribution of summary lengths")
    plt.show()
```

1. CLI / Interactive prompts

Date: 2025-09-13
User: I want a nicer CLI so users can select multiple PDFs and see progress messages.

ChatGPT: Here’s a refined CLI:

```
import os
from summarizer import summarize_pdf_safe


def main():
    pdf_paths = input("Enter PDF paths separated by commas: ").split(",")
    for pdf_path in pdf_paths:
        pdf_path = pdf_path.strip()
        print(f"Processing {pdf_path}...")
        summary = summarize_pdf_safe(pdf_path)
        output_file = os.path.splitext(pdf_path)[0] + "_summary.txt"
        with open(output_file, "w", encoding="utf-8") as f:
            f.write(summary)
        print(f"✅ Summary saved to {output_file}\n")


if __name__ == "__main__":
    main()
```

User: Can we also show a count like “Processing PDF 1 of 3”?

ChatGPT: Sure:

```
for idx, pdf_path in enumerate(pdf_paths, 1):
    pdf_path = pdf_path.strip()

    print(f"Processing PDF {idx} of {len(pdf_paths)}: {pdf_path}...")
```

---

2. GUI using Streamlit

Date: 2025-09-13
User: I want a simple web interface where users can upload PDFs and see the summary instantly.

ChatGPT: Using Streamlit:

```
import streamlit as st

from summarizer import summarize_pdf_safe


st.title("Smart Notes Summarizer")


uploaded_files = st.file_uploader("Upload PDF files", type="pdf", accept_multiple_files=True)


for uploaded_file in uploaded_files:

    with open(uploaded_file.name, "wb") as f:

        f.write(uploaded_file.getbuffer())

    st.write(f"Processing {uploaded_file.name}...")

    summary = summarize_pdf_safe(uploaded_file.name)

    st.subheader(f"Summary for {uploaded_file.name}")

    st.text_area("Summary", value=summary, height=200)
```

User: Can we add a download button for the summary?

ChatGPT: Yes:

```
st.download_button(

    label="Download Summary",

    data=summary,

    file_name=f"{uploaded_file.name}_summary.txt",

    mime="text/plain"

)
```

---

3. GUI enhancements

Date: 2025-09-13
User: Can we display a progress spinner while summarizing?

ChatGPT: Using Streamlit's spinner:

```
with st.spinner(f"Summarizing {uploaded_file.name}..."):
    summary = summarize_pdf_safe(uploaded_file.name)
st.success(f"Finished summarizing {uploaded_file.name}!")
```

User: Can we also show the word count of the summary?

ChatGPT: Yes:

```
word_count = len(summary.split())
st.write(f"Word count: {word_count}")
```

---

4. Optional Tkinter GUI (desktop)

Date: 2025-09-13
User: Can we also have a simple desktop GUI instead of web?

ChatGPT: Example using Tkinter:

```
import tkinter as tk
from tkinter import filedialog, messagebox
from summarizer import summarize_pdf_safe


def open_files():
    file_paths = filedialog.askopenfilenames(filetypes=[("PDF files", "*.pdf")])
    for pdf_path in file_paths:
        summary = summarize_pdf_safe(pdf_path)
        output_file = pdf_path.replace(".pdf", "_summary.txt")
        with open(output_file, "w") as f:
            f.write(summary)
        messagebox.showinfo("Summary Saved", f"Summary saved to {output_file}")


root = tk.Tk()
root.title("Smart Notes Summarizer")
tk.Button(root, text="Select PDFs", command=open_files).pack(padx=20, pady=20)
root.mainloop()
```

User: Can we show a progress bar while processing multiple PDFs?

ChatGPT: Yes, using ttk.Progressbar:

```python
from tkinter import ttk


progress = ttk.Progressbar(root, orient="horizontal", length=300, mode="determinate")
progress.pack(pady=10)
progress["maximum"] = len(file_paths)
for idx, pdf_path in enumerate(file_paths, 1):
    # process pdf
    progress["value"] = idx
    root.update_idletasks()
```

1. Project setup and initial structure

Date: 2025-09-11

User: I want to start a project called Smart Notes Summarizer. Can you help me plan the file structure?

ChatGPT: Sure! Here's a clean structure:

```
smart-notes-summarizer/
 ├── main.py          # Entry point for running the summarizer
 ├── summarizer.py    # Functions to extract text and summarize
 ├── pdf_extractor.py # PDF text extraction logic
 ├── utils.py         # Helper functions, preprocessing, chunking
 ├── config.py        # Config variables
 ├── requirements.txt # Dependencies
└── data/           # Store sample PDFs
```

User: Can we add a simple requirements.txt?

ChatGPT: Yes:

```
transformers
torch
PyMuPDF
regex
```

---

2. Initial pdf_extractor.py

Date: 2025-09-11
User: I need a simple function to extract text from a PDF.

ChatGPT: Here’s a first version:

```
import fitz  # PyMuPDF


def extract_text_from_pdf(pdf_path):
    doc = fitz.open(pdf_path)
    text = ""
    for page in doc:
        text += page.get_text()
    return text
```

User: Can we make it ignore empty pages?

ChatGPT: Updated:

```
def extract_text_from_pdf(pdf_path):
    doc = fitz.open(pdf_path)
    text = ""
    for page in doc:
        page_text = page.get_text()
        if page_text.strip():  # skip empty pages
            text += page_text
    return text
```

---

3. Initial summarizer.py

Date: 2025-09-11
User: I want to summarize extracted text using HuggingFace pipeline.

ChatGPT: Basic version:

```
from transformers import pipeline


def summarize_text(text, max_length=150):
    summarizer = pipeline("summarization")
    summary = summarizer(text, max_length=max_length, min_length=50, do_sample=False)
    return summary[0]['summary_text']
```

User: Can we combine extraction and summarization in one function?

ChatGPT: Sure:

```
from pdf_extractor import extract_text_from_pdf

def summarize_pdf(pdf_path):
    text = extract_text_from_pdf(pdf_path)
    return summarize_text(text)
```

4. First main.py

Date: 2025-09-11
User: I want to test the summarizer with one PDF from data/.

ChatGPT: Simple test:

```
from summarizer import summarize_pdf

pdf_path = "data/sample.pdf"
summary = summarize_pdf(pdf_path)
print("=== Summary ===")
print(summary)
```

User: Can we save the summary to a _summary.txt file automatically?

ChatGPT: Yes:

```
import os
from summarizer import summarize_pdf

pdf_path = "data/sample.pdf"
summary = summarize_pdf(pdf_path)
output_file = os.path.splitext(pdf_path)[0] + "_summary.txt"
with open(output_file, "w", encoding="utf-8") as f:
    f.write(summary)
print(f"Summary saved to {output_file}")
```

5. Early preprocessing in utils.py

Date: 2025-09-11

User: I want a function to clean extracted text by removing special characters and multiple spaces.

ChatGPT: Basic preprocessing:

```
import re


def preprocess_text(text):
    text = re.sub(r'[^A-Za-z0-9\s]', '', text)  # remove special chars
    text = re.sub(r'\s+', ' ', text)             # collapse multiple spaces
    return text.strip()
```

User: Can we integrate this in summarizer.py before summarization?

ChatGPT: Yes:

```
from utils import preprocess_text


def summarize_pdf(pdf_path):
    text = extract_text_from_pdf(pdf_path)
    text = preprocess_text(text)
    return summarize_text(text)
```

1. Improving preprocessing in utils.py

Date: 2025-09-12

User: Some of my extracted PDFs have hyphenation and weird line breaks. Can we improve preprocess_text?

ChatGPT: Yes, here’s an updated function:

```
import re


def preprocess_text(text):
    # Remove hyphenation across line breaks
    text = re.sub(r'-\n', '', text)
    # Remove special characters except newlines
    text = re.sub(r'[^\w\s\n]', '', text)
    # Collapse multiple spaces
    text = re.sub(r'[^\S\n]+', ' ', text)
    return "\n".join(line.strip() for line in text.splitlines())
```

User: Can we also remove empty lines?

ChatGPT: Sure, add this:

```
text = "\n".join(line for line in text.splitlines() if line.strip())
```

---

2. Chunking long texts for summarization

Date: 2025-09-12
User: Some PDFs are very long, and the summarizer crashes. Can we split text into chunks?

ChatGPT: Yes, here's a chunking function for utils.py:

```
def chunk_text(text, max_words=500):
    words = text.split()
    for i in range(0, len(words), max_words):
        yield " ".join(words[i:i+max_words])
```

User: Can we summarize each chunk separately and then combine the summaries?

ChatGPT: Yes, update summarizer.py:

```
def summarize_long_text(text, max_words=500, max_length=150):
    from transformers import pipeline
    summarizer = pipeline("summarization")
    summaries = []
    for chunk in chunk_text(text, max_words):
        summaries.append(summarizer(chunk, max_length=max_length, min_length=50, do_sample=False)[0]['summary_text'])
    return "\n".join(summaries)
```

---

3. Handling multiple PDFs in main.py

Date: 2025-09-12
User: I want to summarize multiple PDFs at once and save each summary with _summary.txt.

ChatGPT: Updated main.py:

```
import os
from summarizer import summarize_long_text, preprocess_text
from pdf_extractor import extract_text_from_pdf

pdf_paths = input("Enter PDF paths separated by commas: ").split(",")

for pdf_path in pdf_paths:
    pdf_path = pdf_path.strip()

    try:

        print(f"Processing {pdf_path}...")

        text = extract_text_from_pdf(pdf_path)

        text = preprocess_text(text)

        summary = summarize_long_text(text)

        output_file = os.path.splitext(pdf_path)[0] + "_summary.txt"

        with open(output_file, "w", encoding="utf-8") as f:

            f.write(summary)

        print(f"✅ Summary saved to {output_file}")

    except Exception as e:

        print(f"❌ Failed to process {pdf_path}: {e}")
```

---

4. Logging successes and failures

Date: 2025-09-12
User: Can we log which PDFs were successfully summarized and which failed?

ChatGPT: Add logging in main.py:

```
import logging


logging.basicConfig(filename="summarizer.log", level=logging.INFO,

                    format="%(asctime)s - %(levelname)s - %(message)s")


# Inside the loop

try:

    # ... processing ...

    logging.info(f"Processed {pdf_path} successfully")

except Exception as e:

    logging.error(f"Failed to process {pdf_path}: {e}")
```

---

5. Preparing for later UI

Date: 2025-09-12
User: Can we add a placeholder function in summarizer.py for future GUI integration?

ChatGPT: Yes, simple wrapper function:

```
def summarize_pdf_for_ui(pdf_path):
    text = extract_text_from_pdf(pdf_path)
    text = preprocess_text(text)
    return summarize_long_text(text)
```

User: This will make it easier to plug into Streamlit later, right?

ChatGPT: Exactly! You can call summarize_pdf_for_ui directly from your GUI.

1. CLI Improvements: Showing progress for multiple PDFs

Date: 2025-09-13
User: I want the CLI to show “Processing PDF 1 of 3” instead of just the filename.

ChatGPT: Sure! Update the loop in main.py:

```
for idx, pdf_path in enumerate(pdf_paths, 1):
    pdf_path = pdf_path.strip()
    print(f"Processing PDF {idx} of {len(pdf_paths)}: {pdf_path}...")
    summary = summarize_pdf_for_ui(pdf_path)
    output_file = os.path.splitext(pdf_path)[0] + "_summary.txt"
    with open(output_file, "w", encoding="utf-8") as f:
        f.write(summary)
    print(f"✅ Summary saved to {output_file}\n")
```

---

2. CLI: Dynamic feedback with spinner

Date: 2025-09-13
User: Can we show a spinner or “processing…” message while summarizing each PDF?

ChatGPT: Yes, using tqdm for CLI:

```
from tqdm import tqdm


for pdf_path in tqdm(pdf_paths, desc="Processing PDFs"):
    pdf_path = pdf_path.strip()
    summary = summarize_pdf_for_ui(pdf_path)
    output_file = os.path.splitext(pdf_path)[0] + "_summary.txt"
    with open(output_file, "w", encoding="utf-8") as f:
        f.write(summary)
```

---

3. Streamlit GUI – File upload & instant summary

Date: 2025-09-13
User: I want a simple web interface to upload PDFs and display summaries instantly.

ChatGPT: Example using Streamlit:

```
import streamlit as st
from summarizer import summarize_pdf_for_ui

st.title("Smart Notes Summarizer")

uploaded_files = st.file_uploader("Upload PDF files", type="pdf", accept_multiple_files=True)

for uploaded_file in uploaded_files:
    with open(uploaded_file.name, "wb") as f:
        f.write(uploaded_file.getbuffer())
    with st.spinner(f"Summarizing {uploaded_file.name}..."):
        summary = summarize_pdf_for_ui(uploaded_file.name)
    st.subheader(f"Summary for {uploaded_file.name}")
    st.text_area("Summary", value=summary, height=200)
```

---

4. Streamlit GUI – Download button

Date: 2025-09-13
User: Can users download the summary directly from the interface?

ChatGPT: Yes:

```
st.download_button(
    label="Download Summary",
    data=summary,
    file_name=f"{uploaded_file.name}_summary.txt",
    mime="text/plain"
)
```

---

5. Displaying word count and PDF info

Date: 2025-09-13
User: Can we also show the word count of the summary and the original PDF?

ChatGPT: Yes, add:

```
original_text = extract_text_from_pdf(uploaded_file.name)
st.write(f"Original word count: {len(original_text.split())}")
st.write(f"Summary word count: {len(summary.split())}")
```

---

6. Tkinter Desktop GUI (optional)

Date: 2025-09-13
User: Can we also have a simple desktop GUI with Tkinter for those don’t want web?

ChatGPT: Example:

```
import tkinter as tk
from tkinter import filedialog, messagebox
from summarizer import summarize_pdf_for_ui


def open_files():
    file_paths = filedialog.askopenfilenames(filetypes=[("PDF files", "*.pdf")])
    for pdf_path in file_paths:
        summary = summarize_pdf_for_ui(pdf_path)
        output_file = pdf_path.replace(".pdf", "_summary.txt")
        with open(output_file, "w") as f:
            f.write(summary)
        messagebox.showinfo("Summary Saved", f"Summary saved to {output_file}")


root = tk.Tk()
root.title("Smart Notes Summarizer")
tk.Button(root, text="Select PDFs", command=open_files).pack(padx=20, pady=20)
root.mainloop()
```

---

7. Tkinter GUI – Progress bar

Date: 2025-09-13
User: Can we show a progress bar while processing multiple PDFs?

ChatGPT: Using ttk.Progressbar:

```
from tkinter import ttk


progress = ttk.Progressbar(root, orient="horizontal", length=300, mode="determinate")
progress.pack(pady=10)
progress["maximum"] = len(file_paths)
for idx, pdf_path in enumerate(file_paths, 1):
    summary = summarize_pdf_for_ui(pdf_path)
    output_file = pdf_path.replace(".pdf", "_summary.txt")
    with open(output_file, "w") as f:
        f.write(summary)
    progress["value"] = idx
    root.update_idletasks()
```

1. Preparing the dataset for fine-tuning

Date: 2025-09-14
User: I want to fine-tune my model on my own notes dataset. Can we create a PyTorch dataset class?

ChatGPT: Yes, here’s a structure similar to your project:

```
import torch
from transformers import AutoTokenizer
from config import DATA_DIR, MODEL_NAME


tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)


class NotesDataset(torch.utils.data.Dataset):
    def __init__(self, notes, summaries, max_input_len=512, max_output_len=150):
        self.notes = notes
        self.summaries = summaries
        self.max_input_len = max_input_len
        self.max_output_len = max_output_len


    def __len__(self):
        return len(self.notes)
    def __getitem__(self, idx):
        inputs = tokenizer(self.notes[idx], max_length=self.max_input_len,
                           truncation=True, padding="max_length", return_tensors="pt")
        labels = tokenizer(self.summaries[idx], max_length=self.max_output_len,
                           truncation=True, padding="max_length", return_tensors="pt")
        return {key: val.squeeze() for key, val in inputs.items()}, labels["input_ids"].squeeze()
```

User: Can we handle empty notes gracefully?

ChatGPT: Sure, just replace empty notes with a placeholder:

```
note = self.notes[idx] if self.notes[idx] else "No content available"
```

---

2. Fine-tuning with LoRA adapter

Date: 2025-09-14
User: I want to use LoRA adapters to fine-tune my T5 model without full retraining.

ChatGPT: Here’s an example:

```
from transformers import AutoModelForSeq2SeqLM, Trainer, TrainingArguments
from peft import get_peft_model, LoraConfig, TaskType
from dataset import NotesDataset
from config import MODEL_NAME, NUM_EPOCHS, TRAIN_BATCH_SIZE, OUTPUT_DIR

model = AutoModelForSeq2SeqLM.from_pretrained(MODEL_NAME)

config = LoraConfig(
    task_type=TaskType.SEQ_2_SEQ_LM,
    r=16,
    lora_alpha=32,
    target_modules=["q", "v"],
    lora_dropout=0.1
)
model = get_peft_model(model, config)

train_dataset = NotesDataset(notes=["Sample note"], summaries=["Sample summary"])
```

```
training_args = TrainingArguments(
    output_dir=OUTPUT_DIR,
    num_train_epochs=NUM_EPOCHS,
    per_device_train_batch_size=TRAIN_BATCH_SIZE,
    save_steps=10,
    logging_steps=5
)

trainer = Trainer(model=model, args=training_args, train_dataset=train_dataset)
trainer.train()
model.save_pretrained(OUTPUT_DIR)
```

User: Can we add evaluation after each epoch?

ChatGPT: Yes, using compute_metrics callback with ROUGE:

```
from evaluation import compute_rouge

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    compute_metrics=compute_rouge
)
```

3. Updating summarizer to use adapter

Date: 2025-09-14
User: How do I use the fine-tuned adapter for summarization in summarizer.py?

ChatGPT: Example using your project structure:

```
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM
from peft import PeftModel

MODEL_NAME = "t5-small"
ADAPTER_PATH = "./finetuned_model/adapter_model"
```

```
tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
base_model = AutoModelForSeq2SeqLM.from_pretrained(MODEL_NAME)
model = PeftModel.from_pretrained(base_model, ADAPTER_PATH)


def summarize_text_adapter(text, max_length=150):
    inputs = tokenizer(text, return_tensors="pt", truncation=True)
    summary_ids = model.generate(inputs["input_ids"], max_length=max_length)
    return tokenizer.decode(summary_ids[0], skip_special_tokens=True)
```

User: Can we replace the existing summarize_text function in summarizer.py with this adapter version?

ChatGPT: Yes, just change the call inside summarize_pdf_for_ui:

```
summaries = [summarize_text_adapter(chunk) for chunk in chunk_text(text)]
```

---

4. ROUGE evaluation

Date: 2025-09-14
User: I want to compute ROUGE scores on a validation set after training.

ChatGPT: Example:

```
import evaluate


rouge = evaluate.load("rouge")


def compute_rouge(eval_pred):
    predictions, references = eval_pred
    decoded_preds = [pred for pred in predictions]
    decoded_refs = [ref for ref in references]
    result = rouge.compute(predictions=decoded_preds, references=decoded_refs)
    return result
```

User: Can we print the results in a table format?

ChatGPT: Yes, using pandas:

```
import pandas as pd


def print_rouge_table(results):
    df = pd.DataFrame(results)
```

```
print(df.to_string(index=False))
```