Algorithm:

1. *Import necessary libraries* and set the OpenAI API key.

2. *Define helper functions* to load PDF, split text into chunks, and generate summaries.

3. *Configure Streamlit layout* and create a sidebar for user inputs like chain type, chunk size, and model selection.

4. *Load and process the PDF file* based on the provided path and chunk settings.

5. *Generate and display summaries* using the selected language model and user-defined prompt when the "Summarize" button is clicked.import openai

Program:

```
import streamlit as st

import os

from langchain.document_loaders import PyPDFLoader

from langchain.prompts import PromptTemplate  # Correct import for PromptTemplate

from langchain.text_splitter import RecursiveCharacterTextSplitter

from langchain.chains.summarize import load_summarize_chain

from langchain.chat_models import ChatOpenAI

openai.api_key = os.environ.get('OPENAI_API_KEY')


def setup_documents(pdf_file_path, chunk_size, chunk_overlap):

    loader = PyPDFLoader(pdf_file_path)

    docs_raw = loader.load()

    docs_raw_text = [doc.page_content for doc in docs_raw]

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

    docs = text_splitter.create_documents(docs_raw_text)

    return docs


def custom_summary(docs, llm, custom_prompt, chain_type, num_summaries):

    custom_prompt = custom_prompt + ":\n\n{text}"
    COMBINE_PROMPT = PromptTemplate(template=custom_prompt,
input_variables=["text"])
    MAP_PROMPT = PromptTemplate(template="Summarize:\n\n{text}",
input_variables=["text"]
    if chain_type == "map_reduce":
        chain = load_summarize_chain(llm, chain_type=chain_type,
                                map_prompt=MAP_PROMPT, combine_prompt=COMBINE_PROMPT)
    else:
        chain = load_summarize_chain(llm, chain_type=chain_type)
    summaries = []
    for i in range(num_summaries):
        summary_output = chain({"input_documents": docs},
return_only_outputs=True)["output_text"]
   summaries.append(summary_output)
    return summaries


def color_chunks(text: str, chunk_size: int, overlap_size: int) -> str:
    overlap_color = "#808080"
    chunk_colors = ["#a8d08d", "#c6dbef", "#e6550d", "#fd8d3c", "#fdae6b", "#fdd0a2"]
    colored_text = ""
    overlap = ""
    color_index = 0
    for i in range(0, len(text), chunk_size-overlap_size):
        chunk = text[i:i+chunk_size]
        if overlap:
            colored_text += f'<mark style="background-color:
{overlap_color};">{overlap}</mark>'
        chunk = chunk[len(overlap):]
        colored_text += f'<mark style="background-color:
{chunk_colors[color_index]};">{chunk}</mark>'
        color_index = (color_index + 1) % len(chunk_colors)
        overlap = text[i+chunk_size-overlap_size:i+chunk_size]
    return colored_text


def main():
    st.set_page_config(layout="wide")
    st.title("Custom Summarization App")
    chain_type = st.sidebar.selectbox("Chain Type", ["map_reduce", "stuff", "refine"])
    chunk_size = st.sidebar.slider("Chunk Size", min_value=100, max_value=10000, step=100, 
value=1900)
    chunk_overlap = st.sidebar.slider("Chunk Overlap", min_value=100, max_value=10000, 
step=100, value=200)
    if st.sidebar.checkbox("Debug chunk size"):
        st.header("Interactive Text Chunk Visualization")
      text_input = st.text_area("Input Text", "This is a test text to showcase the functionality of 
the interactive text chunk visualizer.")
        html_code = color_chunks(text_input, chunk_size, chunk_overlap)
        st.markdown(html_code, unsafe_allow_html=True)
    else:
        user_prompt = st.text_input("Enter the user prompt")
        pdf_file_path = st.text_input("Enter the pdf file path")
        temperature = st.sidebar.number_input("ChatGPT Temperature", min_value=0.0, 
max_value=1.0, step=0.1, value=0.0)
        num_summaries = st.sidebar.number_input("Number of Summaries", min_value=1, 
max_value=10, step=1, value=1)
        llm_choice = st.sidebar.selectbox("LLM", ["ChatGPT", "GPT4"])
        if llm_choice == "ChatGPT":
            llm = ChatOpenAI(temperature=temperature)
        elif llm_choice == "GPT4":
            llm = ChatOpenAI(model_name="gpt-4", temperature=temperature)
        if pdf_file_path:
            docs = setup_documents(pdf_file_path, chunk_size, chunk_overlap)
            st.write("PDF was loaded successfully")
            if st.button("Summarize"):
                result = custom_summary(docs, llm, user_prompt, chain_type, num_summaries)
                st.write("Summaries:")
                for summary in result:
                    st.write(summary)

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