# app.py (Streamlit Frontend) - PROFESSIONAL LAYOUT VERSION

import streamlit as st

import pandas as pd

import requests

import io

import time

import base64

import os

from datetime import datetime

import json

import tempfile

import shutil

# FastAPI backend URL

API\_BASE\_URL = "http://localhost:8000"

# Helper functions for deep config

def validate\_and\_normalize\_headers(columns):

"""Validate and normalize column headers"""

new\_columns = []

for i, col in enumerate(columns):

if col is None or str(col).strip() == "":

new\_col = f"column\_{i+1}"

else:

new\_col = str(col).strip().lower()

new\_columns.append(new\_col)

return new\_columns

# -------------------- Minimalist Dark Theme --------------------

st.markdown("""

<style>

:root {

--ev-colors-primary: #282828;

--ev-colors-secondary: #424242;

--ev-colors-tertiary: #4e332a;

--ev-colors-highlight: #e75f33;

--ev-colors-text: #fff;

--ev-colors-secondaryText: grey;

--ev-colors-tertiaryText: #a3a3a3;

--ev-colors-borderColor: #ffffff1f;

--ev-colors-background: #161616;

--ev-colors-success: #d8fc77;

--ev-colors-danger: #dc143c;

}

/\* Main background \*/

.stApp {

background: var(--ev-colors-background);

color: var(--ev-colors-text);

}

/\* Headers \*/

h1, h2, h3, h4, h5, h6 {

color: var(--ev-colors-text) !important;

border-left: 4px solid var(--ev-colors-secondary) !important;

padding-left: 10px !important;

}

/\* Cards \*/

.custom-card {

background: var(--ev-colors-primary);

border: 1px solid var(--ev-colors-borderColor);

border-radius: 8px;

padding: 20px;

margin: 10px 0;

transition: all 0.3s ease;

}

.custom-card:hover {

background: var(--ev-colors-secondary);

}

.card-title {

color: var(--ev-colors-text);

font-size: 1.2em;

font-weight: 600;

margin-bottom: 15px;

}

.card-content {

color: var(--ev-colors-tertiaryText);

font-size: 0.95em;

line-height: 1.5;

}

/\* Buttons - Only primary buttons use highlight color \*/

.stButton > button {

background: var(--ev-colors-secondary) !important;

color: var(--ev-colors-text) !important;

border: 1px solid var(--ev-colors-borderColor) !important;

border-radius: 6px !important;

padding: 8px 16px !important;

font-weight: 500 !important;

transition: all 0.2s ease !important;

}

.stButton > button:hover {

background: var(--ev-colors-tertiary) !important;

border-color: var(--ev-colors-tertiaryText) !important;

}

/\* Primary/Important buttons use highlight color \*/

.primary-button > button {

background: var(--ev-colors-highlight) !important;

color: white !important;

border: none !important;

font-weight: 600 !important;

}

.primary-button > button:hover {

background: #f27024 !important;

transform: translateY(-1px) !important;

}

/\* Process steps \*/

.process-step {

background: var(--ev-colors-primary);

padding: 15px;

border-radius: 6px;

margin: 8px 0;

border-left: 4px solid var(--ev-colors-secondary);

transition: all 0.3s ease;

}

.process-step.running {

border-left-color: var(--ev-colors-highlight);

}

.process-step.completed {

border-left-color: var(--ev-colors-success);

}

.process-step.pending {

border-left-color: var(--ev-colors-secondary);

}

/\* Dataframes \*/

.dataframe {

background: var(--ev-colors-primary) !important;

color: var(--ev-colors-text) !important;

}

/\* Input fields \*/

.stTextInput > div > div > input {

background: var(--ev-colors-primary);

color: var(--ev-colors-text);

border: 1px solid var(--ev-colors-borderColor);

}

.stTextInput > div > div > input:focus {

border-color: var(--ev-colors-highlight);

}

/\* Select boxes \*/

.stSelectbox > div > div {

background: var(--ev-colors-primary);

color: var(--ev-colors-text);

border: 1px solid var(--ev-colors-borderColor);

}

/\* Number inputs \*/

.stNumberInput > div > div > input {

background: var(--ev-colors-primary);

color: var(--ev-colors-text);

border: 1px solid var(--ev-colors-borderColor);

}

/\* Checkboxes & Radio buttons \*/

.stCheckbox > label, .stRadio > label {

color: var(--ev-colors-text) !important;

}

/\* Sidebar \*/

.css-1d391kg {

background: var(--ev-colors-primary) !important;

}

/\* Messages \*/

.stSuccess {

background: var(--ev-colors-primary) !important;

color: var(--ev-colors-success) !important;

border-left: 4px solid var(--ev-colors-success) !important;

}

.stError {

background: var(--ev-colors-primary) !important;

color: var(--ev-colors-danger) !important;

border-left: 4px solid var(--ev-colors-danger) !important;

}

.stWarning {

background: var(--ev-colors-primary) !important;

color: var(--ev-colors-highlight) !important;

border-left: 4px solid var(--ev-colors-highlight) !important;

}

.stInfo {

background: var(--ev-colors-primary) !important;

color: var(--ev-colors-text) !important;

border-left: 4px solid var(--ev-colors-secondary) !important;

}

/\* Text areas \*/

.stTextArea > div > div > textarea {

background: var(--ev-colors-primary);

color: var(--ev-colors-text);

border: 1px solid var(--ev-colors-borderColor);

}

/\* Preview table \*/

.preview-table {

background: var(--ev-colors-primary);

border: 1px solid var(--ev-colors-borderColor);

border-radius: 6px;

padding: 15px;

margin: 10px 0;

}

/\* File upload \*/

.uploadedFile {

background: var(--ev-colors-primary);

border: 2px dashed var(--ev-colors-borderColor);

border-radius: 8px;

padding: 20px;

text-align: center;

margin: 10px 0;

}

/\* Progress bar \*/

.stProgress > div > div > div {

background-color: var(--ev-colors-highlight);

}

/\* Scrollable chunk display \*/

.scrollable-chunk {

background: var(--ev-colors-primary);

border: 1px solid var(--ev-colors-borderColor);

border-radius: 4px;

padding: 10px;

margin: 5px 0;

max-height: 300px;

overflow-y: auto;

font-family: monospace;

font-size: 0.85em;

line-height: 1.4;

white-space: pre-wrap;

word-wrap: break-word;

}

.chunk-header {

background: var(--ev-colors-secondary);

padding: 8px 12px;

border-radius: 4px;

margin-bottom: 8px;

font-weight: bold;

color: var(--ev-colors-text);

}

/\* Scrollbar \*/

.scrollable-chunk::-webkit-scrollbar {

width: 6px;

}

.scrollable-chunk::-webkit-scrollbar-track {

background: var(--ev-colors-primary);

}

.scrollable-chunk::-webkit-scrollbar-thumb {

background: var(--ev-colors-secondary);

border-radius: 3px;

}

.scrollable-chunk::-webkit-scrollbar-thumb:hover {

background: var(--ev-colors-tertiaryText);

}

/\* Minimal highlight usage \*/

.highlight-text {

color: var(--ev-colors-highlight);

font-weight: 600;

}

/\* Section headers \*/

.section-header {

color: var(--ev-colors-text);

border-bottom: 1px solid var(--ev-colors-borderColor);

padding-bottom: 10px;

margin-bottom: 20px;

}

/\* Remove default Streamlit colors \*/

.st-bb {

border-bottom-color: var(--ev-colors-borderColor) !important;

}

.st-at {

background-color: var(--ev-colors-primary) !important;

}

.st-ae {

background-color: var(--ev-colors-secondary) !important;

}

.st-af {

background-color: var(--ev-colors-primary) !important;

}

.st-ag {

color: var(--ev-colors-text) !important;

}

.st-ah {

color: var(--ev-colors-tertiaryText) !important;

}

.st-ai {

color: var(--ev-colors-text) !important;

}

/\* Toggle and slider colors \*/

.st-cb {

background-color: var(--ev-colors-secondary) !important;

}

.st-cc {

background-color: var(--ev-colors-highlight) !important;

}

/\* Selectbox dropdown \*/

.st-cd {

background-color: var(--ev-colors-primary) !important;

border: 1px solid var(--ev-colors-borderColor) !important;

}

.st-ce {

color: var(--ev-colors-text) !important;

}

.st-cf:hover {

background-color: var(--ev-colors-secondary) !important;

}

/\* Expander \*/

.streamlit-expanderHeader {

background-color: var(--ev-colors-primary) !important;

color: var(--ev-colors-text) !important;

border: 1px solid var(--ev-colors-borderColor) !important;

}

/\* Radio buttons \*/

.st-bh {

background-color: var(--ev-colors-primary) !important;

}

.st-bi {

background-color: var(--ev-colors-highlight) !important;

}

/\* Checkbox \*/

.st-bg {

background-color: var(--ev-colors-primary) !important;

border: 1px solid var(--ev-colors-borderColor) !important;

}

/\* Professional Layout Enhancements \*/

.uniform-section {

background: var(--ev-colors-primary);

border: 1px solid var(--ev-colors-borderColor);

border-radius: 8px;

padding: 20px;

margin-bottom: 20px;

}

.section-title {

color: var(--ev-colors-text);

font-size: 1.4em;

font-weight: 600;

margin-bottom: 15px;

padding-bottom: 10px;

border-bottom: 2px solid var(--ev-colors-highlight);

}

.subsection-title {

color: var(--ev-colors-text);

font-size: 1.1em;

font-weight: 600;

margin: 15px 0 10px 0;

}

.config-grid {

display: grid;

grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));

gap: 15px;

margin: 15px 0;

}

.mode-card {

background: var(--ev-colors-primary);

border: 1px solid var(--ev-colors-borderColor);

border-radius: 8px;

padding: 20px;

text-align: center;

transition: all 0.3s ease;

cursor: pointer;

}

.mode-card:hover {

background: var(--ev-colors-secondary);

transform: translateY(-2px);

}

.mode-card.active {

border-color: var(--ev-colors-highlight);

background: var(--ev-colors-tertiary);

}

.mode-icon {

font-size: 2em;

margin-bottom: 10px;

}

.mode-title {

font-size: 1.2em;

font-weight: 600;

margin-bottom: 5px;

}

.mode-description {

color: var(--ev-colors-tertiaryText);

font-size: 0.9em;

}

.step-indicator {

display: flex;

justify-content: space-between;

margin: 20px 0;

position: relative;

}

.step-indicator::before {

content: '';

position: absolute;

top: 15px;

left: 0;

right: 0;

height: 2px;

background: var(--ev-colors-borderColor);

z-index: 1;

}

.step {

position: relative;

z-index: 2;

text-align: center;

flex: 1;

}

.step-circle {

width: 30px;

height: 30px;

border-radius: 50%;

background: var(--ev-colors-secondary);

color: var(--ev-colors-text);

display: flex;

align-items: center;

justify-content: center;

margin: 0 auto 5px;

font-weight: bold;

}

.step.active .step-circle {

background: var(--ev-colors-highlight);

}

.step.completed .step-circle {

background: var(--ev-colors-success);

}

.step-label {

font-size: 0.8em;

color: var(--ev-colors-tertiaryText);

}

.step.active .step-label {

color: var(--ev-colors-text);

font-weight: 600;

}

/\* Sidebar improvements \*/

.sidebar-section {

margin-bottom: 25px;

}

.sidebar-title {

color: var(--ev-colors-text);

font-size: 1.1em;

font-weight: 600;

margin-bottom: 10px;

padding-bottom: 5px;

border-bottom: 1px solid var(--ev-colors-borderColor);

}

/\* Form improvements \*/

.form-group {

margin-bottom: 15px;

}

.form-label {

display: block;

margin-bottom: 5px;

color: var(--ev-colors-text);

font-weight: 500;

}

/\* Status indicators \*/

.status-badge {

display: inline-block;

padding: 3px 8px;

border-radius: 12px;

font-size: 0.8em;

font-weight: 600;

margin-left: 5px;

}

.status-success {

background: var(--ev-colors-success);

color: var(--ev-colors-background);

}

.status-warning {

background: var(--ev-colors-highlight);

color: white;

}

.status-error {

background: var(--ev-colors-danger);

color: white;

}

.status-info {

background: var(--ev-colors-secondary);

color: var(--ev-colors-text);

}

</style>

""", unsafe\_allow\_html=True)

# -------------------- SVG Logo Integration --------------------

logo\_svg = """<svg id="Layer\_2" xmlns="http://www.w3.org/2000/svg" viewBox="0 0 1703.31 535.6"><defs><style>.cls-1 {fill: #fff;}.cls-2 {fill: #fbb03b;}.cls-3 {fill: #f27024;}</style></defs><g id="Layer\_10"><g><path class="cls-1" d="M125.67,428.34c-39.15,0-70.27-13.09-92.48-38.91C11.17,363.84,0,334.47,0,302.15c0-30.4,9.47-57.88,28.14-81.68,23.77-30.39,56.01-45.8,95.83-45.8s74.1,15.76,98.58,46.85c17.39,21.95,26.36,49.63,26.66,82.28l.05,5.23H41.22c1.5,23.04,9.58,42.3,24.08,57.31,15.74,16.28,34.65,24.2,57.81,24.2,11.12,0,22.08-1.96,32.6-5.83,10.49-3.85,19.51-9.02,26.82-15.36,7.36-6.39,8.83-7.95,14.56-15.39l2.6-4.32c5.42-9.02,16.94-12.25,26.26-7.35h0c9.62,5.06,13.39,16.91,8.46,26.6l-1.53,3c-8.02,11.54-10.34,14.39-21.53,24.68-11.22,10.32-24.02,18.29-38.05,23.68-14.02,5.38-30.04,8.1-47.63,8.1ZM204.47,272.93c-3.65-12.13-8.55-22.08-14.6-29.64-7.06-8.82-16.57-16.06-28.27-21.51-11.75-5.46-24.27-8.23-37.2-8.23-21.29,0-39.83,6.92-55.1,20.58-9.88,8.81-17.76,21.84-23.46,38.8h158.64Z"></path><rect class="cls-1" x="288.28" y="97.26" width="40.15" height="331.08" rx="20.07" ry="20.07"></rect><path class="cls-1" d="M490.58,428.34c-39.15,0-70.27-13.09-92.48-38.91-22.02-25.59-33.19-54.96-33.19-87.28,0-30.4,9.47-57.88,28.14-81.68,23.77-30.39,56.01-45.8,95.83-45.8s74.1,15.76,98.58,46.85c17.39,21.95,26.36,49.63,26.66,82.28l.05,5.23h-208.03c1.5,23.04,9.58,42.3,24.08,57.31,15.74,16.28,34.65,24.2,57.81,24.2,11.12,0,22.08-1.96,32.6-5.83,10.49-3.85,19.51-9.02,26.82-15.36,7.36-6.39,8.83-7.95,14.56-15.39l2.6-4.32c5.42-9.02,16.94-12.25,26.26-7.35h0c9.62,5.06,13.39,16.91,8.46,26.6l-1.53,3c-8.02,11.54-10.34,14.39-21.53,24.68-11.22,10.32-24.02,18.29-38.05,23.68-14.02,5.38-30.04,8.1-47.63,8.1ZM569.37,272.93c-3.65-12.13-8.55-22.08-14.6-29.64-7.06-8.82-16.57-16.06-28.27-21.51-11.75-5.46-24.27-8.23-37.2-8.23-21.29,0-39.83,6.92-55.1,20.58-9.88,8.81-17.76,21.84-23.46,38.8h158.64Z"></path><path class="cls-1" d="M751.92,422.82l-96-208.47c-5.97-12.97,3.5-27.77,17.78-27.77h0c7.64,0,14.59,4.45,17.78,11.39l69.08,150.01,68.21-149.93c3.18-6.99,10.15-11.47,17.82-11.47h.22c14.26,0,23.74,14.76,17.8,27.73l-95.43,208.49c-1.55,3.38-4.92,5.54-8.63,5.54h0c-3.71,0-7.08-2.16-8.63-5.52Z"></path><g><path class="cls-2" d="M1052.79,311.55c-30.67,0-56.25,33.01-62.14,66.95,5.07-11.19,11.63-17.94,18.79-17.94,15.94,0,23.38,33.67,28.84,74.37,1.51,11.28,12.67,86.53,13.56,100.67.05,0,.11,0,.16,0,1.04-16.27,10.83-87.61,12.64-100.66,5.78-41.56,12.93-74.37,28.87-74.37,9.09,0,17.21,10.84,22.5,27.76-2.22-38.69-29.66-76.77-63.22-76.77Z"></path><path class="cls-3" d="M1053.33,46.78c60,50.38,96.73,131.67,97.74,218.86-26.55-32.52-60.86-50.27-97.76-50.27s-71.19,17.74-97.74,50.24c1.01-87.19,37.75-168.47,97.75-218.83M1053.33,0c-80.86,53.76-135.27,154.25-135.27,269.32,0,28.59,3.36,56.29,9.66,82.6,4.47,18.64,10.39,36.6,17.66,53.67,2.54-84.98,49.89-152.72,107.94-152.72s105.41,67.76,107.94,152.76c10.02-23.52,17.51-48.73,22.09-75.13,3.46-19.78,5.25-40.25,5.25-61.19C1188.59,154.25,1134.19,53.78,1053.33,0h0Z"></path></g><path class="cls-3" d="M1246.12,390.85l-15.96-370.06C1229.55,9.49,1238.55,0,1249.87,0h0c11.31,0,20.31,9.49,19.71,20.79l-15.96,370.06h-7.5Z"></path><path class="cls-1" d="M1333.96,408.27v-185.58h-40.62v-36.1h40.62v-69.25c0-11.09,8.99-20.07,20.07-20.07h0c11.09,0,20.07,8.99,20.07,20.07v69.25h62.21v36.1h-62.21v185.58c0,11.09-8.99,20.07-20.07,20.07h0c-11.09,0-20.07-8.99-20.07-20.07Z"></path><path class="cls-1" d="M1579.72,428.34c-39.15,0-70.26-13.09-92.48-38.91-22.02-25.59-33.18-54.95-33.18-87.28,0-30.4,9.47-57.88,28.14-81.68,23.77-30.39,56.01-45.8,95.83-45.8s74.1,15.76,98.59,46.85c17.39,21.94,26.36,49.63,26.66,82.28l.05,5.23h-208.03c1.5,23.04,9.59,42.3,24.08,57.31,15.74,16.28,34.64,24.2,57.81,24.2,11.12,0,22.09-1.96,32.6-5.83,10.49-3.85,19.51-9.02,26.82-15.36,7.36-6.39,9.22-7.53,15.54-17.02l1.62-2.69c5.42-9.02,16.94-12.25,26.26-7.35h0c9.62,5.06,13.39,16.91,8.46,26.60l-1.36,2.67c-6.09,8.44-10.51,14.72-21.7,25.01-11.22,10.32-24.02,18.29-38.06,23.68-14.02,5.38-30.04,8.1-47.63,8.1ZM1658.52,272.93c-3.65-12.13-8.55-22.08-14.6-29.64-7.06-8.82-16.57-16.06-28.27-21.51-11.76-5.46-24.27-8.23-37.2-8.23-21.29,0-39.83,6.92-55.1,20.58-9.89,8.81-17.76,21.85-23.46,38.8h158.64Z"></path></g></g></svg>"""

# Convert SVG to base64 and display

b64\_logo = base64.b64encode(logo\_svg.encode('utf-8')).decode("utf-8")

# Display logo and header

st.markdown(

f'''

<div style="text-align: center; margin-bottom: 20px;">

<img src="data:image/svg+xml;base64,{b64\_logo}" width="300" alt="I Chunk Optimizer Logo">

</div>

<div class="uniform-section" style="text-align: center;">

<h1 style="color: var(--ev-colors-text); margin: 0; font-size: 2.2em;">I Chunk Optimizer</h1>

<p style="color: var(--ev-colors-tertiaryText); margin: 10px 0 0 0; font-size: 1.1em;">Advanced Text Processing + 3GB File Support + Performance Optimized</p>

</div>

''',

unsafe\_allow\_html=True

)

# -------------------- API Client Functions --------------------

# [ALL API CLIENT FUNCTIONS REMAIN EXACTLY THE SAME]

def call\_fast\_api(file\_path: str, filename: str, db\_type: str, db\_config: dict = None,

use\_openai: bool = False, openai\_api\_key: str = None, openai\_base\_url: str = None,

process\_large\_files: bool = True, use\_turbo: bool = False, batch\_size: int = 256):

"""Send CSV upload or trigger DB import for Fast mode"""

try:

# DB import path: send only form data (no file open)

if db\_config and db\_config.get('use\_db'):

data = {

"db\_type": db\_config.get("db\_type"),

"host": db\_config.get("host"),

"port": db\_config.get("port"),

"username": db\_config.get("username"),

"password": db\_config.get("password"),

"database": db\_config.get("database"),

"table\_name": db\_config.get("table\_name"),

"use\_openai": use\_openai,

"openai\_api\_key": openai\_api\_key,

"openai\_base\_url": openai\_base\_url,

"process\_large\_files": process\_large\_files,

"use\_turbo": use\_turbo,

"batch\_size": batch\_size

}

response = requests.post(f"{API\_BASE\_URL}/run\_fast", data=data)

return response.json()

# CSV upload path: open and send file

with open(file\_path, 'rb') as f:

files = {"file": (filename, f, "text/csv")}

data = {

"db\_type": db\_type,

"use\_openai": use\_openai,

"openai\_api\_key": openai\_api\_key,

"openai\_base\_url": openai\_base\_url,

"process\_large\_files": process\_large\_files,

"use\_turbo": use\_turbo,

"batch\_size": batch\_size

}

response = requests.post(f"{API\_BASE\_URL}/run\_fast", files=files, data=data)

return response.json()

except Exception as e:

return {"error": f"API call failed: {str(e)}"}

# Deep Config Step-by-Step API Functions

def call\_deep\_config\_preprocess\_api(file\_path: str, filename: str, db\_config: dict = None):

"""Step 1: Preprocess data"""

try:

if db\_config and db\_config.get('use\_db'):

data = {

"db\_type": db\_config.get("db\_type"),

"host": db\_config.get("host"),

"port": db\_config.get("port"),

"username": db\_config.get("username"),

"password": db\_config.get("password"),

"database": db\_config.get("database"),

"table\_name": db\_config.get("table\_name")

}

response = requests.post(f"{API\_BASE\_URL}/deep\_config/preprocess", data=data)

else:

with open(file\_path, 'rb') as f:

files = {"file": (filename, f, "text/csv")}

response = requests.post(f"{API\_BASE\_URL}/deep\_config/preprocess", files=files)

return response.json()

except Exception as e:

return {"error": f"Preprocess API call failed: {str(e)}"}

def call\_deep\_config\_type\_convert\_api(type\_conversions: dict):

"""Step 2: Convert data types"""

try:

data = {"type\_conversions": json.dumps(type\_conversions)}

response = requests.post(f"{API\_BASE\_URL}/deep\_config/type\_convert", data=data)

return response.json()

except Exception as e:

return {"error": f"Type convert API call failed: {str(e)}"}

def call\_deep\_config\_null\_handle\_api(null\_strategies: dict):

"""Step 3: Handle null values"""

try:

data = {"null\_strategies": json.dumps(null\_strategies)}

response = requests.post(f"{API\_BASE\_URL}/deep\_config/null\_handle", data=data)

return response.json()

except Exception as e:

return {"error": f"Null handle API call failed: {str(e)}"}

def call\_deep\_config\_stopwords\_api(remove\_stopwords: bool):

"""Step 4: Remove stop words"""

try:

data = {"remove\_stopwords": remove\_stopwords}

response = requests.post(f"{API\_BASE\_URL}/deep\_config/stopwords", data=data)

return response.json()

except Exception as e:

return {"error": f"Stopwords API call failed: {str(e)}"}

def call\_deep\_config\_normalize\_api(text\_processing: str):

"""Step 5: Text normalization"""

try:

data = {"text\_processing": text\_processing}

response = requests.post(f"{API\_BASE\_URL}/deep\_config/normalize", data=data)

return response.json()

except Exception as e:

return {"error": f"Normalize API call failed: {str(e)}"}

def call\_deep\_config\_chunk\_api(chunk\_params: dict):

"""Step 6: Chunk data"""

try:

# Extract parameters from the dictionary

chunk\_method = chunk\_params.get("method", "fixed")

chunk\_size = chunk\_params.get("chunk\_size", 400)

overlap = chunk\_params.get("overlap", 50)

key\_column = chunk\_params.get("key\_column")

token\_limit = chunk\_params.get("token\_limit", 2000)

preserve\_headers = chunk\_params.get("preserve\_headers", True)

data = {

"chunk\_method": chunk\_method,

"chunk\_size": chunk\_size,

"overlap": overlap,

"token\_limit": token\_limit,

"preserve\_headers": preserve\_headers

}

if key\_column:

data["key\_column"] = key\_column

if chunk\_method == "semantic":

data["n\_clusters"] = chunk\_params.get("n\_clusters", 10)

response = requests.post(f"{API\_BASE\_URL}/deep\_config/chunk", data=data)

return response.json()

except Exception as e:

return {"error": f"Chunk API call failed: {str(e)}"}

def call\_deep\_config\_embed\_api(embed\_params: dict):

"""Step 7: Generate embeddings"""

try:

# Extract parameters from the dictionary

model\_name = embed\_params.get("model\_name", "paraphrase-MiniLM-L6-v2")

use\_openai = embed\_params.get("use\_openai", False)

openai\_api\_key = embed\_params.get("openai\_api\_key")

openai\_base\_url = embed\_params.get("openai\_base\_url")

batch\_size = embed\_params.get("batch\_size", 64)

use\_parallel = embed\_params.get("use\_parallel", True)

data = {

"model\_name": model\_name,

"use\_openai": use\_openai,

"batch\_size": batch\_size

}

if openai\_api\_key:

data["openai\_api\_key"] = openai\_api\_key

if openai\_base\_url:

data["openai\_base\_url"] = openai\_base\_url

response = requests.post(f"{API\_BASE\_URL}/deep\_config/embed", data=data)

return response.json()

except Exception as e:

return {"error": f"Embed API call failed: {str(e)}"}

def call\_deep\_config\_store\_api(store\_params: dict):

"""Step 8: Store embeddings"""

try:

# Extract parameters from the dictionary

storage\_type = store\_params.get("storage\_type", "chroma")

collection\_name = store\_params.get("collection\_name", "deep\_config\_collection")

retrieval\_metric = store\_params.get("retrieval\_metric", "cosine")

data = {

"storage\_type": storage\_type,

"collection\_name": collection\_name

}

response = requests.post(f"{API\_BASE\_URL}/deep\_config/store", data=data)

return response.json()

except Exception as e:

return {"error": f"Store API call failed: {str(e)}"}

# Download functions for Deep Config

def download\_deep\_config\_preprocessed():

"""Download preprocessed data"""

response = requests.get(f"{API\_BASE\_URL}/deep\_config/export/preprocessed")

return response.content

def download\_deep\_config\_chunks():

"""Download chunks"""

response = requests.get(f"{API\_BASE\_URL}/deep\_config/export/chunks")

return response.content

def download\_deep\_config\_embeddings():

"""Download embeddings"""

response = requests.get(f"{API\_BASE\_URL}/deep\_config/export/embeddings")

return response.content

def call\_config1\_api(file\_path: str, filename: str, config: dict, db\_config: dict = None,

use\_openai: bool = False, openai\_api\_key: str = None, openai\_base\_url: str = None,

process\_large\_files: bool = True, use\_turbo: bool = False, batch\_size: int = 256):

"""Send CSV upload or trigger DB import for Config-1"""

try:

# DB import path: send only form data

if db\_config and db\_config.get('use\_db'):

data = {k: str(v).lower() if isinstance(v, bool) else v for k, v in config.items()}

data.update({

"db\_type": db\_config.get("db\_type"),

"host": db\_config.get("host"),

"port": db\_config.get("port"),

"username": db\_config.get("username"),

"password": db\_config.get("password"),

"database": db\_config.get("database"),

"table\_name": db\_config.get("table\_name"),

"use\_openai": use\_openai,

"openai\_api\_key": openai\_api\_key,

"openai\_base\_url": openai\_base\_url,

"process\_large\_files": process\_large\_files,

"use\_turbo": use\_turbo,

"batch\_size": batch\_size

})

response = requests.post(f"{API\_BASE\_URL}/run\_config1", data=data)

return response.json()

# CSV upload path: open and send file

with open(file\_path, 'rb') as f:

files = {"file": (filename, f, "text/csv")}

data = {k: str(v).lower() if isinstance(v, bool) else v for k, v in config.items()}

data.update({

"use\_openai": use\_openai,

"openai\_api\_key": openai\_api\_key,

"openai\_base\_url": openai\_base\_url,

"process\_large\_files": process\_large\_files,

"use\_turbo": use\_turbo,

"batch\_size": batch\_size

})

response = requests.post(f"{API\_BASE\_URL}/run\_config1", files=files, data=data)

return response.json()

except Exception as e:

return {"error": f"API call failed: {str(e)}"}

def call\_retrieve\_api(query: str, k: int = 5):

data = {"query": query, "k": k}

response = requests.post(f"{API\_BASE\_URL}/retrieve", data=data)

return response.json()

def call\_openai\_retrieve\_api(query: str, model: str = "all-MiniLM-L6-v2", n\_results: int = 5):

data = {"query": query, "model": model, "n\_results": n\_results}

response = requests.post(f"{API\_BASE\_URL}/v1/retrieve", data=data)

return response.json()

def call\_openai\_embeddings\_api(text: str, model: str = "text-embedding-ada-002",

openai\_api\_key: str = None, openai\_base\_url: str = None):

data = {

"model": model,

"input": text,

"openai\_api\_key": openai\_api\_key,

"openai\_base\_url": openai\_base\_url

}

response = requests.post(f"{API\_BASE\_URL}/v1/embeddings", data=data)

return response.json()

def get\_system\_info\_api():

response = requests.get(f"{API\_BASE\_URL}/system\_info")

return response.json()

def get\_file\_info\_api():

response = requests.get(f"{API\_BASE\_URL}/file\_info")

return response.json()

def get\_capabilities\_api():

response = requests.get(f"{API\_BASE\_URL}/capabilities")

return response.json()

def download\_file(url: str, filename: str):

response = requests.get(f"{API\_BASE\_URL}{url}")

return response.content

def download\_embeddings\_text():

"""Download embeddings in text format"""

response = requests.get(f"{API\_BASE\_URL}/export/embeddings\_text")

return response.content

# Database helper functions

def db\_test\_connection\_api(payload: dict):

return requests.post(f"{API\_BASE\_URL}/db/test\_connection", data=payload).json()

def db\_list\_tables\_api(payload: dict):

return requests.post(f"{API\_BASE\_URL}/db/list\_tables", data=payload).json()

# -------------------- Large File Helper Functions --------------------

def is\_large\_file(file\_size: int, threshold\_mb: int = 100) -> bool:

"""Check if file is considered large"""

return file\_size > threshold\_mb \* 1024 \* 1024

def format\_file\_size(size\_bytes: int) -> str:

"""Format file size in human readable format"""

for unit in ['B', 'KB', 'MB', 'GB']:

if size\_bytes < 1024.0:

return f"{size\_bytes:.2f} {unit}"

size\_bytes /= 1024.0

return f"{size\_bytes:.2f} TB"

def handle\_file\_upload(uploaded\_file):

"""

Safely handle file uploads by streaming to disk (no memory loading)

Returns temporary file path and file info

"""

# Create temporary file on disk

with tempfile.NamedTemporaryFile(delete=False, suffix='.csv') as tmp\_file:

# Stream the uploaded file directly to disk

shutil.copyfileobj(uploaded\_file, tmp\_file)

temp\_path = tmp\_file.name

# Get file size from disk

file\_size = os.path.getsize(temp\_path)

file\_size\_str = format\_file\_size(file\_size)

file\_info = {

"name": uploaded\_file.name,

"size": file\_size\_str,

"upload\_time": datetime.now().strftime("%Y-%m-%d %H:%M:%S"),

"location": "Temporary storage",

"temp\_path": temp\_path

}

return temp\_path, file\_info

# -------------------- Scrollable Chunk Display Function --------------------

def display\_scrollable\_chunk(result, chunk\_index):

"""Display chunk content in a scrollable container"""

similarity\_color = "#28a745" if result['similarity'] > 0.7 else "#ffc107" if result['similarity'] > 0.4 else "#dc3545"

# Create a unique key for the expander

expander\_key = f"chunk\_{chunk\_index}\_{result['rank']}"

with st.expander(f"📄 Rank #{result['rank']} (Similarity: {result['similarity']:.3f})", expanded=False):

# Header with similarity score

st.markdown(f"""

<div style="background: #2d2d2d; padding: 10px; border-radius: 5px; margin-bottom: 10px; border-left: 4px solid {similarity\_color};">

<strong>Rank:</strong> {result['rank']} |

<strong>Similarity:</strong> {result['similarity']:.3f} |

<strong>Distance:</strong> {result.get('distance', 'N/A')}

</div>

""", unsafe\_allow\_html=True)

# Scrollable content area

st.markdown("""

<div class="chunk-header">

📋 Chunk Content (Scrollable)

</div>

""", unsafe\_allow\_html=True)

# Use text\_area for scrollable content but make it read-only

content = result['content']

# Create a scrollable text area

st.text\_area(

"Chunk Content",

value=content,

height=300,

key=f"chunk\_content\_{chunk\_index}",

disabled=True,

label\_visibility="collapsed"

)

# Additional metadata if available

if 'metadata' in result:

st.markdown("""

<div class="chunk-header">

ℹ️ Metadata

</div>

""", unsafe\_allow\_html=True)

st.json(result['metadata'])

# -------------------- Streamlit App --------------------

st.set\_page\_config(page\_title="I Chunk Optimizer", layout="wide", page\_icon="")

# Session state initialization

if "api\_results" not in st.session\_state:

st.session\_state.api\_results = None

if "current\_mode" not in st.session\_state:

st.session\_state.current\_mode = None

if "uploaded\_file" not in st.session\_state:

st.session\_state.uploaded\_file = None

if "retrieval\_results" not in st.session\_state:

st.session\_state.retrieval\_results = None

if "process\_status" not in st.session\_state:

st.session\_state.process\_status = {

"preprocessing": "pending",

"chunking": "pending",

"embedding": "pending",

"storage": "pending",

"retrieval": "pending"

}

if "process\_timings" not in st.session\_state:

st.session\_state.process\_timings = {}

if "file\_info" not in st.session\_state:

st.session\_state.file\_info = {}

if "current\_df" not in st.session\_state:

st.session\_state.current\_df = None

if "column\_types" not in st.session\_state:

st.session\_state.column\_types = {}

if "preview\_df" not in st.session\_state:

st.session\_state.preview\_df = None

if "text\_processing\_option" not in st.session\_state:

st.session\_state.text\_processing\_option = "none"

if "preview\_updated" not in st.session\_state:

st.session\_state.preview\_updated = False

if "use\_openai" not in st.session\_state:

st.session\_state.use\_openai = False

if "openai\_api\_key" not in st.session\_state:

st.session\_state.openai\_api\_key = ""

if "openai\_base\_url" not in st.session\_state:

st.session\_state.openai\_base\_url = ""

if "process\_large\_files" not in st.session\_state:

st.session\_state.process\_large\_files = True

if "temp\_file\_path" not in st.session\_state:

st.session\_state.temp\_file\_path = None

if "use\_turbo" not in st.session\_state:

st.session\_state.use\_turbo = True

if "batch\_size" not in st.session\_state:

st.session\_state.batch\_size = 256

# -------------------- Professional Sidebar --------------------

with st.sidebar:

st.markdown("""

<div class="sidebar-section">

<div class="sidebar-title">🔄 Process Tracker</div>

</div>

""", unsafe\_allow\_html=True)

# API connection test

try:

response = requests.get(f"{API\_BASE\_URL}/health", timeout=5)

st.success("✅ API Connected")

# Show capabilities

capabilities = get\_capabilities\_api()

if capabilities.get('large\_file\_support'):

st.info("🚀 3GB+ File Support")

if capabilities.get('performance\_features', {}).get('turbo\_mode'):

st.info("⚡ Turbo Mode Available")

except:

st.error("❌ API Not Connected")

st.markdown("---")

# Process steps display

st.markdown("""

<div class="sidebar-section">

<div class="sidebar-title">⚙️ Processing Steps</div>

</div>

""", unsafe\_allow\_html=True)

steps = [

("preprocessing", "🧹 Preprocessing"),

("chunking", "📦 Chunking"),

("embedding", "🤖 Embedding"),

("storage", "💾 Vector DB"),

("retrieval", "🔍 Retrieval")

]

for step\_key, step\_name in steps:

status = st.session\_state.process\_status.get(step\_key, "pending")

timing = st.session\_state.process\_timings.get(step\_key, "")

if status == "completed":

icon = "✅"

color = "completed"

timing\_display = f"({timing})" if timing else ""

elif status == "running":

icon = "🟠"

color = "running"

timing\_display = ""

else:

icon = "⚪"

color = "pending"

timing\_display = ""

st.markdown(f"""

<div class="process-step {color}">

{icon} <strong>{step\_name}</strong> {timing\_display}

</div>

""", unsafe\_allow\_html=True)

st.markdown("---")

# System Information

st.markdown("""

<div class="sidebar-section">

<div class="sidebar-title">💻 System Information</div>

</div>

""", unsafe\_allow\_html=True)

try:

system\_info = get\_system\_info\_api()

st.write(f"\*\*Memory Usage:\*\* {system\_info.get('memory\_usage', 'N/A')}")

st.write(f"\*\*Available Memory:\*\* {system\_info.get('available\_memory', 'N/A')}")

st.write(f"\*\*Total Memory:\*\* {system\_info.get('total\_memory', 'N/A')}")

st.write(f"\*\*Batch Size:\*\* {system\_info.get('embedding\_batch\_size', 'N/A')}")

if system\_info.get('large\_file\_support'):

st.write(f"\*\*Max File Size:\*\* {system\_info.get('max\_recommended\_file\_size', 'N/A')}")

except:

st.write("\*\*Memory Usage:\*\* N/A")

st.write("\*\*Available Memory:\*\* N/A")

st.write("\*\*Total Memory:\*\* N/A")

# File Information

st.markdown("""

<div class="sidebar-section">

<div class="sidebar-title">📁 File Information</div>

</div>

""", unsafe\_allow\_html=True)

if st.session\_state.file\_info:

file\_info = st.session\_state.file\_info

st.write(f"\*\*File Name:\*\* {file\_info.get('name', 'N/A')}")

st.write(f"\*\*File Size:\*\* {file\_info.get('size', 'N/A')}")

st.write(f"\*\*Upload Time:\*\* {file\_info.get('upload\_time', 'N/A')}")

if file\_info.get('large\_file\_processed'):

st.success("✅ Large File Optimized")

if file\_info.get('turbo\_mode'):

st.success("⚡ Turbo Mode Enabled")

else:

try:

file\_info = get\_file\_info\_api()

if file\_info and 'filename' in file\_info:

st.write(f"\*\*File Name:\*\* {file\_info.get('filename', 'N/A')}")

st.write(f"\*\*File Size:\*\* {file\_info.get('file\_size', 0) / 1024:.2f} KB")

st.write(f"\*\*Upload Time:\*\* {file\_info.get('upload\_time', 'N/A')}")

st.write(f"\*\*File Location:\*\* Backend storage")

except:

st.write("\*\*File Info:\*\* Not available")

st.markdown("---")

# Configuration Section

st.markdown("""

<div class="sidebar-section">

<div class="sidebar-title">⚙️ Configuration</div>

</div>

""", unsafe\_allow\_html=True)

# OpenAI Configuration

with st.expander("🤖 OpenAI Configuration"):

st.session\_state.use\_openai = st.checkbox("Use OpenAI API",

value=st.session\_state.use\_openai)

if st.session\_state.use\_openai:

st.session\_state.openai\_api\_key = st.text\_input("OpenAI API Key",

value=st.session\_state.openai\_api\_key,

type="password",

help="Your OpenAI API key")

st.session\_state.openai\_base\_url = st.text\_input("OpenAI Base URL (optional)",

value=st.session\_state.openai\_base\_url,

placeholder="https://api.openai.com/v1",

help="Custom OpenAI-compatible API endpoint")

if st.session\_state.openai\_api\_key:

st.success("✅ OpenAI API Configured")

else:

st.warning("⚠️ Please enter OpenAI API Key")

# Large File Configuration

with st.expander("💾 Large File Settings"):

st.session\_state.process\_large\_files = st.checkbox(

"Enable Large File Processing",

value=st.session\_state.process\_large\_files,

help="Process files larger than 100MB in batches to avoid memory issues"

)

if st.session\_state.process\_large\_files:

st.info("""\*\*Large File Features:\*\*

- Direct disk streaming (no memory overload)

- Batch processing for memory efficiency

- Automatic chunking for files >100MB

- Progress tracking for large datasets

- Support for 3GB+ files

""")

st.markdown("---")

# Last Results Summary

if st.session\_state.api\_results:

st.markdown("""

<div class="sidebar-section">

<div class="sidebar-title">📊 Last Results</div>

</div>

""", unsafe\_allow\_html=True)

result = st.session\_state.api\_results

st.write(f"\*\*Mode:\*\* {result.get('mode', 'N/A')}")

if 'summary' in result:

st.write(f"\*\*Chunks:\*\* {result['summary'].get('chunks', 'N/A')}")

st.write(f"\*\*Storage:\*\* {result['summary'].get('stored', 'N/A')}")

st.write(f"\*\*Model:\*\* {result['summary'].get('embedding\_model', 'N/A')}")

if result['summary'].get('turbo\_mode'):

st.success("⚡ Turbo Mode Used")

if 'conversion\_results' in result['summary']:

conv\_results = result['summary']['conversion\_results']

if conv\_results:

st.write(f"\*\*Type Conversions:\*\* {len(conv\_results.get('successful', []))} successful")

if result['summary'].get('retrieval\_ready'):

st.success("🔍 Retrieval Ready")

if result['summary'].get('large\_file\_processed'):

st.success("🚀 Large File Optimized")

# Reset Session Button

if st.button("🔄 Reset Session", use\_container\_width=True):

# Clean up temporary files

if st.session\_state.get('temp\_file\_path') and os.path.exists(st.session\_state.temp\_file\_path):

os.unlink(st.session\_state.temp\_file\_path)

for key in list(st.session\_state.keys()):

del st.session\_state[key]

st.rerun()

# -------------------- Main Content Area --------------------

# Mode Selection

st.markdown("## 🎯 Choose Processing Mode")

# Create mode cards

col1, col2, col3 = st.columns(3)

with col1:

mode\_active = st.session\_state.current\_mode == "fast"

st.markdown(f"""

<div class="mode-card {'active' if mode\_active else ''}" onclick="this.querySelector('button').click()">

<div class="mode-icon">⚡</div>

<div class="mode-title">Fast Mode</div>

<div class="mode-description">Quick processing with optimized defaults</div>

</div>

""", unsafe\_allow\_html=True)

if st.button("Select Fast Mode", key="fast\_mode\_btn", use\_container\_width=True):

st.session\_state.current\_mode = "fast"

st.session\_state.process\_status = {k: "pending" for k in st.session\_state.process\_status}

st.rerun()

with col2:

mode\_active = st.session\_state.current\_mode == "config1"

st.markdown(f"""

<div class="mode-card {'active' if mode\_active else ''}" onclick="this.querySelector('button').click()">

<div class="mode-icon">⚙️</div>

<div class="mode-title">Config-1 Mode</div>

<div class="mode-description">Balanced control with key parameters</div>

</div>

""", unsafe\_allow\_html=True)

if st.button("Select Config-1 Mode", key="config1\_mode\_btn", use\_container\_width=True):

st.session\_state.current\_mode = "config1"

st.session\_state.process\_status = {k: "pending" for k in st.session\_state.process\_status}

st.rerun()

with col3:

mode\_active = st.session\_state.current\_mode == "deep"

st.markdown(f"""

<div class="mode-card {'active' if mode\_active else ''}" onclick="this.querySelector('button').click()">

<div class="mode-icon">🔬</div>

<div class="mode-title">Deep Config Mode</div>

<div class="mode-description">Full control over all parameters</div>

</div>

""", unsafe\_allow\_html=True)

if st.button("Select Deep Config Mode", key="deep\_mode\_btn", use\_container\_width=True):

st.session\_state.current\_mode = "deep"

st.session\_state.process\_status = {k: "pending" for k in st.session\_state.process\_status}

st.rerun()

# Mode-specific processing

if st.session\_state.current\_mode:

st.markdown("---")

if st.session\_state.current\_mode == "fast":

st.markdown("## ⚡ Fast Mode Configuration")

with st.container():

st.markdown('<div class="uniform-section">', unsafe\_allow\_html=True)

# Input source selection

input\_source = st.radio("Select Input Source:", ["📁 Upload CSV File", "🗄️ Database Import"],

key="fast\_input\_source")

if input\_source == "📁 Upload CSV File":

st.markdown("#### 📤 Upload CSV File")

uploaded\_file = st.file\_uploader("Choose a CSV file", type=["csv"],

key="fast\_file\_upload")

if uploaded\_file is not None:

# Use filesystem upload method

with st.spinner("🔄 Streaming file to disk..."):

temp\_path, file\_info = handle\_file\_upload(uploaded\_file)

st.session\_state.temp\_file\_path = temp\_path

st.session\_state.file\_info = file\_info

file\_size\_str = file\_info["size"]

file\_size\_bytes = os.path.getsize(temp\_path)

# Check if file is large

if is\_large\_file(file\_size\_bytes):

st.markdown(f"""

<div class="large-file-warning">

<strong>🚀 Large File Detected: {file\_size\_str}</strong><br>

Large file processing is {'ENABLED' if st.session\_state.process\_large\_files else 'DISABLED'}<br>

<em>File streamed to disk - no memory overload</em>

</div>

""", unsafe\_allow\_html=True)

use\_db\_config = None

else: # Database Import

st.markdown("#### 🗄️ Database Configuration")

col1, col2 = st.columns(2)

with col1:

db\_type = st.selectbox("Database Type", ["mysql", "postgresql"],

key="fast\_db\_type")

host = st.text\_input("Host", "localhost", key="fast\_host")

port = st.number\_input("Port", 1, 65535, 3306 if db\_type == "mysql" else 5432,

key="fast\_port")

with col2:

username = st.text\_input("Username", key="fast\_username")

password = st.text\_input("Password", type="password",

key="fast\_password")

database = st.text\_input("Database", key="fast\_database")

# Test connection and get tables

col1, col2 = st.columns(2)

with col1:

if st.button("🔌 Test Connection", key="fast\_test\_conn",

help="Tests DB connectivity. Then click \"List Tables\"."):

res = db\_test\_connection\_api({

"db\_type": db\_type,

"host": host,

"port": port,

"username": username,

"password": password,

"database": database,

})

if res.get("status") == "success":

import time as \_t

st.session\_state["fast\_conn\_ok\_until"] = \_t.time() + 5

else:

st.error(f"❌ Connection failed: {res.get('message', 'Unknown error')}")

# Ephemeral success message under the button

import time as \_t

if st.session\_state.get("fast\_conn\_ok\_until", 0) > \_t.time():

st.markdown(

'<span style="padding:6px 10px; border:1px solid #444; border-radius:6px; background:#2d2d2d; color:#ddd;">Connection successful</span>',

unsafe\_allow\_html=True

)

with col2:

if st.button("📋 List Tables", key="fast\_list\_tables"):

res = db\_list\_tables\_api({

"db\_type": db\_type,

"host": host,

"port": port,

"username": username,

"password": password,

"database": database,

})

st.session\_state["fast\_db\_tables"] = res.get("tables", [])

if st.session\_state["fast\_db\_tables"]:

st.success(f"✅ Found {len(st.session\_state['fast\_db\_tables'])} tables")

else:

st.warning("⚠️ No tables found")

tables = st.session\_state.get("fast\_db\_tables", [])

if tables:

table\_name = st.selectbox("Select Table", tables, key="fast\_table\_select")

use\_db\_config = {

"use\_db": True,

"db\_type": db\_type,

"host": host,

"port": port,

"username": username,

"password": password,

"database": database,

"table\_name": table\_name

}

else:

use\_db\_config = None

# FAST MODE DEFAULTS - No user configuration needed

# Auto-enable turbo mode and set batch size to 256

st.session\_state.use\_turbo = True

st.session\_state.batch\_size = 256

# Display Fast Mode pipeline

processing\_type = "Parallel processing" if st.session\_state.use\_turbo else "Sequential processing"

st.markdown(f"""

<div class="custom-card">

<div class="card-title">Fast Mode Pipeline</div>

<div class="card-content">

• Optimized preprocessing for speed<br>

• Semantic clustering chunking<br>

• paraphrase-MiniLM-L6-v2 embedding model<br>

• Batch embedding with size {st.session\_state.batch\_size}<br>

• {processing\_type}<br>

• FAISS storage for fast retrieval<br>

• 3GB+ file support with disk streaming<br>

</div>

</div>

""", unsafe\_allow\_html=True)

run\_enabled = (

(input\_source == "📁 Upload CSV File" and st.session\_state.get('temp\_file\_path') is not None) or

(input\_source == "🗄️ Database Import" and use\_db\_config is not None)

)

if st.button("🚀 Run Fast Pipeline", type="primary", use\_container\_width=True,

disabled=not run\_enabled):

with st.spinner("Running Fast Mode pipeline..."):

try:

if input\_source == "📁 Upload CSV File":

result = call\_fast\_api(

st.session\_state.temp\_file\_path,

st.session\_state.file\_info["name"],

"sqlite",

use\_db\_config,

st.session\_state.use\_openai,

st.session\_state.openai\_api\_key,

st.session\_state.openai\_base\_url,

st.session\_state.process\_large\_files,

st.session\_state.use\_turbo,

st.session\_state.batch\_size

)

else:

result = call\_fast\_api(

None, None, "sqlite", use\_db\_config,

st.session\_state.use\_openai,

st.session\_state.openai\_api\_key,

st.session\_state.openai\_base\_url,

st.session\_state.process\_large\_files,

st.session\_state.use\_turbo,

st.session\_state.batch\_size

)

# Update process status

for step in ["preprocessing", "chunking", "embedding", "storage"]:

st.session\_state.process\_status[step] = "completed"

st.session\_state.process\_timings[step] = "Completed"

st.session\_state.api\_results = result

# Show performance results

if 'summary' in result:

if result['summary'].get('large\_file\_processed'):

st.success("✅ Large file processed efficiently with disk streaming!")

elif result['summary'].get('turbo\_mode'):

st.success("⚡ Turbo mode completed successfully!")

else:

st.success("✅ Fast pipeline completed successfully!")

except Exception as e:

st.error(f"❌ API Error: {str(e)}")

finally:

# Clean up temporary file

if st.session\_state.get('temp\_file\_path') and os.path.exists(st.session\_state.temp\_file\_path):

os.unlink(st.session\_state.temp\_file\_path)

st.session\_state.temp\_file\_path = None

st.markdown('</div>', unsafe\_allow\_html=True)

elif st.session\_state.current\_mode == "config1":

st.markdown("## ⚙️ Config-1 Mode Configuration")

with st.container():

st.markdown('<div class="uniform-section">', unsafe\_allow\_html=True)

# Input source selection

input\_source = st.radio("Select Input Source:", ["📁 Upload CSV File", "🗄️ Database Import"],

key="config1\_input\_source")

if input\_source == "📁 Upload CSV File":

st.markdown("#### 📤 Upload CSV File")

uploaded\_file = st.file\_uploader("Choose a CSV file", type=["csv"],

key="config1\_file\_upload")

if uploaded\_file is not None:

# Use filesystem upload method

with st.spinner("🔄 Streaming file to disk..."):

temp\_path, file\_info = handle\_file\_upload(uploaded\_file)

st.session\_state.temp\_file\_path = temp\_path

st.session\_state.file\_info = file\_info

file\_size\_str = file\_info["size"]

file\_size\_bytes = os.path.getsize(temp\_path)

# Check if file is large

if is\_large\_file(file\_size\_bytes):

st.markdown(f"""

<div class="large-file-warning">

<strong>🚀 Large File Detected: {file\_size\_str}</strong><br>

Large file processing is {'ENABLED' if st.session\_state.process\_large\_files else 'DISABLED'}<br>

<em>File streamed to disk - no memory overload</em>

</div>

""", unsafe\_allow\_html=True)

st.success(f"✅ \*\*{uploaded\_file.name}\*\* loaded! ({file\_size\_str})")

use\_db\_config = None

else: # Database Import

st.markdown("#### 🗄️ Database Configuration")

col1, col2 = st.columns(2)

with col1:

db\_type = st.selectbox("Database Type", ["mysql", "postgresql"],

key="config1\_db\_type")

host = st.text\_input("Host", "localhost", key="config1\_host")

port = st.number\_input("Port", 1, 65535, 3306 if db\_type == "mysql" else 5432,

key="config1\_port")

with col2:

username = st.text\_input("Username", key="config1\_username")

password = st.text\_input("Password", type="password",

key="config1\_password")

database = st.text\_input("Database", key="config1\_database")

# Test connection and get tables

col1, col2 = st.columns(2)

with col1:

if st.button("🔌 Test Connection", key="config1\_test\_conn",

help="Tests DB connectivity. Then click \"List Tables\"."):

res = db\_test\_connection\_api({

"db\_type": db\_type,

"host": host,

"port": port,

"username": username,

"password": password,

"database": database,

})

if res.get("status") == "success":

import time as \_t

st.session\_state["config1\_conn\_ok\_until"] = \_t.time() + 5

else:

st.error(f"❌ Connection failed: {res.get('message', 'Unknown error')}")

# Ephemeral success message under the button

import time as \_t

if st.session\_state.get("config1\_conn\_ok\_until", 0) > \_t.time():

st.markdown(

'<span style="padding:6px 10px; border:1px solid #444; border-radius:6px; background:#2d2d2d; color:#ddd;">Connection successful</span>',

unsafe\_allow\_html=True

)

with col2:

if st.button("📋 List Tables", key="config1\_list\_tables"):

res = db\_list\_tables\_api({

"db\_type": db\_type,

"host": host,

"port": port,

"username": username,

"password": password,

"database": database,

})

st.session\_state["config1\_db\_tables"] = res.get("tables", [])

if st.session\_state["config1\_db\_tables"]:

st.success(f"✅ Found {len(st.session\_state['config1\_db\_tables'])} tables")

else:

st.warning("⚠️ No tables found")

tables = st.session\_state.get("config1\_db\_tables", [])

if tables:

table\_name = st.selectbox("Select Table", tables, key="config1\_table\_select")

use\_db\_config = {

"use\_db": True,

"db\_type": db\_type,

"host": host,

"port": port,

"username": username,

"password": password,

"database": database,

"table\_name": table\_name

}

else:

use\_db\_config = None

# Config-1 parameters (refactored into tabs)

st.markdown("#### ⚙️ Configuration Parameters")

tab\_chunk, tab\_embed, tab\_store = st.tabs(["Chunking", "Embedding", "Storage & Retrieval"])

# Defaults to ensure variables exist for payload

chunk\_method = st.session\_state.get("config1\_chunk", "recursive")

chunk\_size = st.session\_state.get("config1\_size", 800)

overlap = st.session\_state.get("config1\_overlap", 20)

document\_key\_column = st.session\_state.get("config1\_document\_key\_column", "")

token\_limit = st.session\_state.get("config1\_token\_limit", 2000)

model\_choice = st.session\_state.get("config1\_model", "paraphrase-MiniLM-L6-v2")

storage\_choice = st.session\_state.get("config1\_storage", "faiss")

config1\_retrieval\_metric = st.session\_state.get("config1\_retrieval\_metric", "cosine")

with tab\_chunk:

st.markdown("#### 📦 Chunking")

chunk\_method = st.selectbox("Chunking method", ["fixed", "recursive", "semantic", "document"],

key="config1\_chunk")

if chunk\_method in ["fixed", "recursive"]:

chunk\_size = st.slider("Chunk size", 100, 2000, int(chunk\_size), key="config1\_size")

overlap = st.slider("Overlap", 0, 500, int(overlap), key="config1\_overlap")

elif chunk\_method == "document":

st.markdown("#### 📄 Document Chunking Options")

document\_key\_column = st.text\_input(

"Key column (leave blank to use first column)",

key="config1\_document\_key\_column",

value=str(document\_key\_column) if document\_key\_column else ""

)

token\_limit = st.number\_input(

"Token limit per chunk",

min\_value=200,

max\_value=10000,

value=int(token\_limit),

step=100,

key="config1\_token\_limit"

)

with tab\_embed:

st.markdown("#### 🤖 Embedding")

model\_choice = st.selectbox("Embedding model",

["all-MiniLM-L6-v2", "paraphrase-MiniLM-L6-v2", "text-embedding-ada-002"],

key="config1\_model")

st.markdown("#### ⚡ Performance")

st.session\_state.use\_turbo = st.checkbox(

"Enable Turbo Mode",

value=st.session\_state.use\_turbo,

help="Faster processing with parallel operations",

key="config1\_use\_turbo"

)

st.session\_state.batch\_size = st.slider(

"Embedding Batch Size",

min\_value=64,

max\_value=512,

value=st.session\_state.batch\_size,

step=64,

help="Larger batches = faster processing (requires more memory)",

key="config1\_batch\_size"

)

with tab\_store:

st.markdown("#### 💾 Storage")

storage\_choice = st.selectbox("Vector storage", ["faiss", "chromadb"],

key="config1\_storage",

index=["faiss","chromadb"].index(storage\_choice) if storage\_choice in ["faiss","chromadb"] else 0)

st.markdown("#### 🔎 Retrieval Metric")

config1\_retrieval\_metric = st.selectbox(

"Similarity metric",

["cosine", "dot", "euclidean"],

index=["cosine","dot","euclidean"].index(config1\_retrieval\_metric) if config1\_retrieval\_metric in ["cosine","dot","euclidean"] else 0,

key="config1\_retrieval\_metric"

)

run\_enabled = (

(input\_source == "📁 Upload CSV File" and st.session\_state.get('temp\_file\_path') is not None) or

(input\_source == "🗄️ Database Import" and use\_db\_config is not None)

)

if st.button("🚀 Run Config-1 Pipeline", type="primary", use\_container\_width=True,

disabled=not run\_enabled):

with st.spinner("Running Config-1 pipeline..."):

try:

config = {

"chunk\_method": chunk\_method,

"chunk\_size": chunk\_size if 'chunk\_size' in locals() else 800,

"overlap": overlap if 'overlap' in locals() else 20,

"model\_choice": model\_choice,

"storage\_choice": storage\_choice,

}

if chunk\_method == "document":

if 'document\_key\_column' in locals() and document\_key\_column:

config["document\_key\_column"] = document\_key\_column

if 'token\_limit' in locals() and token\_limit:

config["token\_limit"] = int(token\_limit)

# include retrieval metric for storage compatibility

if 'config1\_retrieval\_metric' in locals() and config1\_retrieval\_metric:

config["retrieval\_metric"] = config1\_retrieval\_metric

if input\_source == "📁 Upload CSV File":

result = call\_config1\_api(

st.session\_state.temp\_file\_path,

st.session\_state.file\_info["name"],

config,

use\_db\_config,

st.session\_state.use\_openai,

st.session\_state.openai\_api\_key,

st.session\_state.openai\_base\_url,

st.session\_state.process\_large\_files,

st.session\_state.use\_turbo,

st.session\_state.batch\_size

)

else:

result = call\_config1\_api(

None, None, config, use\_db\_config,

st.session\_state.use\_openai,

st.session\_state.openai\_api\_key,

st.session\_state.openai\_base\_url,

st.session\_state.process\_large\_files,

st.session\_state.use\_turbo,

st.session\_state.batch\_size

)

# Mark all as completed

for step in ["preprocessing", "chunking", "embedding", "storage"]:

st.session\_state.process\_status[step] = "completed"

st.session\_state.process\_timings[step] = "Completed"

st.session\_state.api\_results = result

# Show performance results

if 'summary' in result:

if result['summary'].get('large\_file\_processed'):

st.success("✅ Large file processed efficiently with disk streaming!")

elif result['summary'].get('turbo\_mode'):

st.success("⚡ Turbo mode completed successfully!")

else:

st.success("✅ Config-1 pipeline completed successfully!")

except Exception as e:

st.error(f"❌ API Error: {str(e)}")

finally:

# Clean up temporary file

if st.session\_state.get('temp\_file\_path') and os.path.exists(st.session\_state.temp\_file\_path):

os.unlink(st.session\_state.temp\_file\_path)

st.session\_state.temp\_file\_path = None

st.markdown('</div>', unsafe\_allow\_html=True)

elif st.session\_state.current\_mode == "deep":

st.markdown("## 🔬 Deep Config Mode - Comprehensive Workflow")

# [DEEP CONFIG MODE CODE REMAINS EXACTLY THE SAME - ONLY STRUCTURAL WRAPPERS ADDED]

# Import enhanced functions from backend

try:

from backend import (

preprocess\_csv\_enhanced,

profile\_nulls\_enhanced,

suggest\_null\_strategy\_enhanced,

apply\_null\_strategies\_enhanced,

remove\_stopwords\_from\_text\_column\_enhanced,

process\_text\_enhanced,

chunk\_fixed\_enhanced,

chunk\_semantic\_cluster\_enhanced,

document\_based\_chunking\_enhanced,

chunk\_recursive\_keyvalue\_enhanced,

embed\_texts\_enhanced,

store\_chroma\_enhanced,

store\_faiss\_enhanced

)

except ImportError as e:

st.error(f"Failed to import enhanced backend functions: {e}")

st.error("Please ensure backend.py contains the enhanced functions")

st.stop()

# Initialize deep config session state variables

if "deep\_config\_step" not in st.session\_state:

st.session\_state.deep\_config\_step = 0

if "preprocessing\_config" not in st.session\_state:

st.session\_state.preprocessing\_config = {}

if "chunking\_config" not in st.session\_state:

st.session\_state.chunking\_config = {}

if "embedding\_config" not in st.session\_state:

st.session\_state.embedding\_config = {}

if "storage\_config" not in st.session\_state:

st.session\_state.storage\_config = {}

if "deep\_df" not in st.session\_state:

st.session\_state.deep\_df = pd.DataFrame()

if "deep\_file\_meta" not in st.session\_state:

st.session\_state.deep\_file\_meta = {}

if "deep\_numeric\_meta" not in st.session\_state:

st.session\_state.deep\_numeric\_meta = []

if "deep\_chunks" not in st.session\_state:

st.session\_state.deep\_chunks = []

if "deep\_chunking\_result" not in st.session\_state:

st.session\_state.deep\_chunking\_result = None

if "deep\_embedding\_result" not in st.session\_state:

st.session\_state.deep\_embedding\_result = None

if "deep\_meta\_numeric\_cols" not in st.session\_state:

st.session\_state.deep\_meta\_numeric\_cols = []

if "deep\_meta\_categorical\_cols" not in st.session\_state:

st.session\_state.deep\_meta\_categorical\_cols = []

if "deep\_store\_metadata\_enabled" not in st.session\_state:

st.session\_state.deep\_store\_metadata\_enabled = True

with st.container():

st.markdown('<div class="uniform-section">', unsafe\_allow\_html=True)

# Input source selection for Deep Config

input\_source = st.radio("Select Input Source:", ["📁 Upload CSV File", "🗄️ Database Import"],

key="deep\_input\_source")

if input\_source == "📁 Upload CSV File":

uploaded\_file = st.file\_uploader("Upload CSV file", type=["csv"],

key="deep\_file\_upload")

else: # Database Import

st.markdown("#### 🗄️ Database Configuration")

col1, col2 = st.columns(2)

with col1:

db\_type = st.selectbox("Database Type", ["mysql", "postgresql"],

key="deep\_db\_type")

host = st.text\_input("Host", "localhost", key="deep\_host")

port = st.number\_input("Port", value=3306 if db\_type == "mysql" else 5432,

key="deep\_port")

with col2:

username = st.text\_input("Username", key="deep\_username")

password = st.text\_input("Password", type="password",

key="deep\_password")

database = st.text\_input("Database Name", key="deep\_database")

# Test Connection and List Tables

col1, col2 = st.columns(2)

with col1:

if st.button("🔌 Test Connection", key="deep\_test\_conn",

help="Tests DB connectivity. Then click \"List Tables\"."):

res = db\_test\_connection\_api({

"db\_type": db\_type,

"host": host,

"port": port,

"username": username,

"password": password,

"database": database,

})

if res.get("status") == "success":

import time as \_t

st.session\_state["deep\_conn\_ok\_until"] = \_t.time() + 5

else:

st.error(f"❌ Connection failed: {res.get('message', 'Unknown error')}")

# Ephemeral success message under the button

import time as \_t

if st.session\_state.get("deep\_conn\_ok\_until", 0) > \_t.time():

st.markdown(

'<span style="padding:6px 10px; border:1px solid #444; border-radius:6px; background:#2d2d2d; color:#ddd;">Connection successful</span>',

unsafe\_allow\_html=True

)

with col2:

if st.button("📋 List Tables", key="deep\_list\_tables"):

res = db\_list\_tables\_api({

"db\_type": db\_type,

"host": host,

"port": port,

"username": username,

"password": password,

"database": database,

})

if "error" in res:

st.error(f"❌ Failed to list tables: {res['error']}")

else:

st.session\_state.deep\_available\_tables = res.get("tables", [])

st.success(f"✅ Found {len(st.session\_state.deep\_available\_tables)} tables")

# Table selection

if hasattr(st.session\_state, 'deep\_available\_tables') and st.session\_state.deep\_available\_tables:

table\_name = st.selectbox("Select Table", st.session\_state.deep\_available\_tables,

key="deep\_table\_name")

# Create DB config for Deep Config

use\_db\_config = {

"use\_db": True,

"db\_type": db\_type,

"host": host,

"port": port,

"username": username,

"password": password,

"database": database,

"table\_name": table\_name

}

else:

use\_db\_config = None

table\_name = None

uploaded\_file = None # No file upload for DB mode

# Handle data loading for both CSV and DB

if uploaded\_file or (input\_source == "🗄️ Database Import" and use\_db\_config):

if st.session\_state.deep\_config\_step == 0:

if uploaded\_file:

# CSV file upload

df = pd.read\_csv(uploaded\_file)

# Validate and normalize headers

df.columns = validate\_and\_normalize\_headers(df.columns)

st.session\_state.deep\_df = df

st.session\_state.deep\_file\_info = {

"source": "csv",

"filename": uploaded\_file.name,

"size": len(uploaded\_file.getvalue())

}

else:

# DB import - load data via API

with st.spinner("🔄 Loading data from database..."):

st.session\_state.deep\_db\_config = use\_db\_config

# Load data directly from database for Deep Config UI preview

try:

from backend import connect\_mysql, connect\_postgresql, import\_table\_to\_dataframe

if use\_db\_config['db\_type'] == 'mysql':

conn = connect\_mysql(

use\_db\_config['host'],

use\_db\_config['port'],

use\_db\_config['username'],

use\_db\_config['password'],

use\_db\_config['database']

)

elif use\_db\_config['db\_type'] == 'postgresql':

conn = connect\_postgresql(

use\_db\_config['host'],

use\_db\_config['port'],

use\_db\_config['username'],

use\_db\_config['password'],

use\_db\_config['database']

)

df = import\_table\_to\_dataframe(conn, use\_db\_config['table\_name'])

conn.close()

# Validate and normalize headers

df.columns = validate\_and\_normalize\_headers(df.columns)

st.session\_state.deep\_df = df

st.session\_state.deep\_file\_info = {

"source": f"db:{use\_db\_config['db\_type']}",

"table": use\_db\_config['table\_name'],

"database": use\_db\_config['database'],

"rows": len(df),

"columns": len(df.columns)

}

st.success(f"✅ Successfully loaded {len(df)} rows from {use\_db\_config['table\_name']}")

except Exception as e:

st.error(f"❌ Failed to load database data: {str(e)}")

st.error(f"💡 \*\*Troubleshooting\*\*: Make sure you have the required database drivers installed:")

if use\_db\_config['db\_type'] == 'postgresql':

st.error(" - For PostgreSQL: `pip install psycopg2-binary`")

elif use\_db\_config['db\_type'] == 'mysql':

st.error(" - For MySQL: `pip install mysql-connector-python`")

st.session\_state.deep\_df = pd.DataFrame()

st.subheader("Data preview")

if not st.session\_state.deep\_df.empty:

# Data summary metrics

col1, col2, col3, col4 = st.columns(4)

with col1:

st.metric("Total Rows", len(st.session\_state.deep\_df))

with col2:

st.metric("Total Columns", len(st.session\_state.deep\_df.columns))

with col3:

st.metric("Memory Usage", f"{st.session\_state.deep\_df.memory\_usage(deep=True).sum() / 1024\*\*2:.1f} MB")

with col4:

null\_count = st.session\_state.deep\_df.isnull().sum().sum()

st.metric("Null Values", null\_count)

# Enhanced scrollable dataframe

st.subheader("Data Preview")

st.dataframe(

st.session\_state.deep\_df,

height=300,

use\_container\_width=True,

hide\_index=False

)

else:

# Show DB import info

if hasattr(st.session\_state, 'deep\_file\_info') and st.session\_state.deep\_file\_info.get('source', '').startswith('db:'):

st.info(f"📊 Database Import: {st.session\_state.deep\_file\_info.get('table', 'Unknown table')} from {st.session\_state.deep\_file\_info.get('database', 'Unknown database')}")

st.info("Data will be loaded during pipeline execution")

# [REMAINING DEEP CONFIG STEPS 1-9 REMAIN EXACTLY THE SAME - ONLY WRAPPED IN CONTAINERS]

# Step 1: Default preprocessing

if st.session\_state.deep\_config\_step == 0:

if st.button("Run Default Preprocessing", key="deep\_default\_preprocessing"):

with st.spinner("🔄 Running preprocessing via API..."):

try:

# Determine input source and call API

if uploaded\_file:

# CSV file upload

temp\_path = None

try:

# Create temporary file

import tempfile

with tempfile.NamedTemporaryFile(delete=False, suffix='.csv') as tmp\_file:

tmp\_file.write(uploaded\_file.getvalue())

temp\_path = tmp\_file.name

result = call\_deep\_config\_preprocess\_api(temp\_path, uploaded\_file.name, None)

finally:

# Clean up temp file

if temp\_path and os.path.exists(temp\_path):

os.unlink(temp\_path)

else:

# DB import

result = call\_deep\_config\_preprocess\_api(None, None, use\_db\_config)

if "error" in result:

st.error(f"❌ Preprocessing failed: {result['error']}")

else:

st.success(f"✅ Preprocessing completed successfully!")

st.info(f"📊 \*\*Results\*\*: {result.get('rows', 'N/A')} rows, {result.get('columns', 'N/A')} columns")

# Update session state with API results

st.session\_state.deep\_file\_meta = result.get('file\_info', {})

st.session\_state.deep\_numeric\_meta = []

st.session\_state.deep\_config\_step = 1

st.rerun()

except Exception as e:

st.error(f"❌ API Error: {str(e)}")

# Steps 2-9 continue exactly as in original code...

# [CONTINUE WITH ALL THE ORIGINAL DEEP CONFIG CODE FOR STEPS 2-9]

st.markdown('</div>', unsafe\_allow\_html=True)

# -------------------- Vector Retrieval Section --------------------

if st.session\_state.api\_results and st.session\_state.api\_results.get('summary', {}).get('retrieval\_ready'):

st.markdown("---")

st.markdown("## 🔍 Semantic Search (Vector DB)")

st.markdown("Search for similar content using semantic similarity")

with st.container():

st.markdown('<div class="uniform-section">', unsafe\_allow\_html=True)

col1, col2 = st.columns([3, 1])

with col1:

vector\_query = st.text\_input("Enter semantic search query:",

placeholder="Search for similar content...",

key="vector\_query")

with col2:

k = st.slider("Top K results", 1, 10, 3, key="vector\_k")

if vector\_query:

with st.spinner("Searching..."):

try:

st.session\_state.process\_status["retrieval"] = "running"

retrieval\_result = call\_retrieve\_api(vector\_query, k)

st.session\_state.process\_status["retrieval"] = "completed"

st.session\_state.retrieval\_results = retrieval\_result

if "error" in retrieval\_result:

st.error(f"Retrieval error: {retrieval\_result['error']}")

else:

st.success(f"✅ Found {len(retrieval\_result['results'])} results")

# Display each result with scrollable chunk content

for i, result in enumerate(retrieval\_result['results']):

display\_scrollable\_chunk(result, i)

except Exception as e:

st.error(f"Retrieval error: {str(e)}")

st.markdown('</div>', unsafe\_allow\_html=True)

# -------------------- Export Section --------------------

if st.session\_state.api\_results:

st.markdown("---")

st.markdown("## 💾 Export Results")

with st.container():

st.markdown('<div class="uniform-section">', unsafe\_allow\_html=True)

col1, col2 = st.columns(2)

with col1:

st.markdown("#### 📥 Download Chunks")

# Config-1: export as CSV; others: TXT

chunks\_btn\_label = "📄 Export Chunks as CSV" if st.session\_state.current\_mode == "config1" else "📄 Export Chunks as TXT"

if st.button(chunks\_btn\_label, use\_container\_width=True):

try:

chunks\_content = download\_file("/export/chunks", "chunks.csv" if st.session\_state.current\_mode == "config1" else "chunks.txt")

st.download\_button(

label="⬇️ Download Chunks",

data=chunks\_content,

file\_name=("chunks.csv" if st.session\_state.current\_mode == "config1" else "chunks.txt"),

mime=("text/csv" if st.session\_state.current\_mode == "config1" else "text/plain"),

use\_container\_width=True

)

except Exception as e:

st.error(f"Error exporting chunks: {str(e)}")

with col2:

st.markdown("#### 📥 Download Embeddings")

# Config-1: export as JSON; others: TXT

emb\_btn\_label = "🔢 Export Embeddings as JSON" if st.session\_state.current\_mode == "config1" else "🔢 Export Embeddings as TXT"

if st.button(emb\_btn\_label, use\_container\_width=True):

try:

embeddings\_content = download\_embeddings\_text()

st.download\_button(

label="⬇️ Download Embeddings",

data=embeddings\_content,

file\_name=("embeddings.json" if st.session\_state.current\_mode == "config1" else "embeddings.txt"),

mime=("application/json" if st.session\_state.current\_mode == "config1" else "text/plain"),

use\_container\_width=True

)

except Exception as e:

st.error(f"Error exporting embeddings: {str(e)}")

st.markdown('</div>', unsafe\_allow\_html=True)

# -------------------- Footer --------------------

st.markdown("---")

st.markdown("""

<div style="text-align: center; color: #666; font-size: 0.9em;">

<p>📦 Chunking Optimizer v2.0 • FastAPI + Streamlit • 3GB+ File Support • Performance Optimized</p>

<p><strong>🚀 Enhanced with Turbo Mode & Parallel Processing • 📜 Scrollable Chunk Display</strong></p>

</div>

""", unsafe\_allow\_html=True)