LDR streamlit App (Kesha Dharmesh Shah)

This notebook contains the full code for a Streamlit-based legal document redaction app using a BERT-NER model.

APP Script(front-end)

""" Lessons Learnt:

1. Learned how to use Streamlit for building interactive web applications, including custom layouts, widgets, and file handling.
2. Understood how to connect and integrate with Hugging Face models for NLP tasks, using both local and remote model files.
3. Practiced connecting and pushing code to GitHub repositories, including branch management and collaboration workflows.
4. Improved skills in handling document formats (PDF, DOCX, TXT) and extracting text for processing.
5. Enhanced understanding of deploying AI-powered applications for legal document redaction and privacy protection.
6. Learned to use Python libraries such as pypdf, python-docx, and Streamlit for end-to-end app development.
7. Gained experience in error handling, user interface design, and providing user feedback in web apps.
8. Explored best practices for organizing code, separating inference logic, and maintaining modularity.
9. Incorporated user feedback to improve usability and functionality of the application.
10. Addressed security and privacy considerations when handling sensitive legal documents and user data.
11. Optimized performance for processing large documents and ensured scalability for future enhancements.
12. Overcame challenges related to model integration, file compatibility, and deployment, learning effective troubleshooting strategies.
    """

```
"""
Streamlit App for Legal Document Redaction using BERT-NER

This app allows users to upload or paste text from PDF, Word, or Text files and redact sensitive information using a NER model.
"""
import streamlit as st
from pypdf import PdfReader
import re
from docx import Document
import io
import os

# Import inference script from the same folder
from inference_script import redact_text, REDACTED_ENTITIES, HF_MODEL_NAME

# ----------------------------
# Streamlit Page Config
# ----------------------------
st.set_page_config(
    page_title="PDF Text Redaction (BERT-NER)",
    layout="wide"
)

# Custom CSS for better styling
st.markdown("""
    <style>
        /* Root and main styling */
        * {
            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
        }

        /* Full page background - vibrant gradient */
        .stApp {
            background: linear-gradient(135deg, #f0f4ff 0%, #e8f5ff 50%, #e8f9ff 100%);
            min-height: 100vh;
        }

        /* Sidebar with gradient */
        [data-testid="stSidebar"] {
            background: linear-gradient(180deg, #1e40af 0%, #1e3a8a 100%) !important;
            padding: 0 !important;
        }
```

```
[data-testid="stSidebar"] > div:first-child {
    padding: 20px;
}

[data-testid="stSidebar"] [data-testid="stVerticalBlockBorderWrapper"] {
    background: rgba(255, 255, 255, 0.95);
    border-radius: 12px;
    margin: 15px 10px;
    padding: 15px;
    border: 2px solid rgba(255, 255, 255, 0.3);
    box-shadow: 0 8px 32px rgba(30, 64, 175, 0.15);
}

[data-testid="stSidebar"] h1,
[data-testid="stSidebar"] h2,
[data-testid="stSidebar"] h3 {
    color: #ffffff;
    margin-top: 0;
    font-weight: 700;
    border-bottom: 2px solid rgba(255, 255, 255, 0.3);
    padding-bottom: 10px;
}

[data-testid="stSidebar"] label {
    color: #ffffff !important;
    font-weight: 600 !important;
    font-size: 0.95em !important;
}

[data-testid="stSidebar"] .stCheckbox label {
    color: #ffffff !important;
}

[data-testid="stSidebar"] .stCheckbox label span {
    color: #ffffff !important;
}

[data-testid="stSidebar"] div[role="radiogroup"] label {
    color: #ffffff !important;
}

[data-testid="stSidebar"] p {
    color: #ffffff !important;
}

[data-testid="stSidebar"] span {
    color: #ffffff !important;
}

[data-testid="stSidebar"] div {
    color: #ffffff !important;
}

/* Main content padding and styling */
.main > div:first-child {
    padding-top: 30px;
    padding-bottom: 30px;
    padding-left: 40px;
    padding-right: 40px;
}

/* Title styling - eye-catching */
h1 {
    background: linear-gradient(135deg, #1e40af 0%, #0ea5e9 100%);
    -webkit-background-clip: text;
    -webkit-text-fill-color: transparent;
    background-clip: text;
    font-size: 3.5em;
    margin-bottom: 10px;
    margin-top: 20px;
    font-weight: 900;
    text-align: center;
    letter-spacing: -1px;
    text-shadow: 0 2px 4px rgba(0,0,0,0.1);
}

/* Subtitle styling */
```

```
.subtitle {
    text-align: center;
    color: #475569;
    font-size: 1.15em;
    margin-bottom: 30px;
    font-weight: 600;
    letter-spacing: 0.5px;
}

/* Status badge styling */
.stSuccess, .stInfo {
    background: linear-gradient(135deg, #ecfdf5 0%, #d1fae5 100%);
    border-left: 5px solid #10b981;
    border-radius: 10px;
    color: #047857;
    padding: 15px;
    font-size: 0.98em;
    font-weight: 500;
    box-shadow: 0 4px 15px rgba(16, 185, 129, 0.1);
}

/* Section headers */
h2 {
    color: #1e40af;
    font-size: 1.8em;
    margin-top: 30px;
    margin-bottom: 20px;
    font-weight: 800;
    position: relative;
    padding-bottom: 15px;
}

h2::after {
    content: '';
    position: absolute;
    bottom: 0;
    left: 0;
    width: 60px;
    height: 4px;
    background: linear-gradient(90deg, #1e40af, #0ea5e9);
    border-radius: 2px;
}

h3 {
    color: #1e3a8a;
    font-size: 1.3em;
    margin-top: 20px;
    margin-bottom: 12px;
    font-weight: 700;
}

/* Radio button group - card style */
.stRadio {
    background: linear-gradient(135deg, #ffffff 0%, #f0f9ff 100%);
    padding: 20px;
    border-radius: 12px;
    border: 2px solid #bfdbfe;
    margin: 15px 0;
    box-shadow: 0 4px 15px rgba(30, 64, 175, 0.08);
}

.stRadio label {
    color: #1e293b !important;
    font-weight: 600;
    font-size: 1em;
}

/* Input fields - styled cards */
textarea, input[type="text"], input[type="email"], input[type="password"] {
    border: 2px solid #bfdbfe !important;
    border-radius: 10px !important;
    background: linear-gradient(135deg, #f8fafc 0%, #f0f9ff 100%) !important;
    color: #0f172a !important;
    font-size: 0.95em !important;
    padding: 12px !important;
    box-shadow: 0 2px 8px rgba(30, 64, 175, 0.05) !important;
}
```

```
textarea:focus, input:focus {
    border-color: #0ea5e9 !important;
    background: #ffffff !important;
    box-shadow: 0 0 0 4px rgba(14, 165, 233, 0.15) !important;
}

/* Buttons - vibrant and interactive */
.stButton > button {
    background: linear-gradient(135deg, #1e40af 0%, #0ea5e9 100%);
    color: white;
    border: none;
    border-radius: 10px;
    padding: 14px 28px;
    font-weight: 700;
    font-size: 1em;
    transition: all 0.3s ease;
    width: 100%;
    cursor: pointer;
    box-shadow: 0 6px 20px rgba(30, 64, 175, 0.3);
    text-transform: uppercase;
    letter-spacing: 0.5px;
}

.stButton > button:hover {
    transform: translateY(-3px);
    box-shadow: 0 10px 30px rgba(30, 64, 175, 0.4);
    background: linear-gradient(135deg, #0ea5e9 0%, #1e40af 100%);
}

.stButton > button:active {
    transform: translateY(-1px);
}

/* Download buttons - success color */
.stDownloadButton > button {
    background: linear-gradient(135deg, #10b981 0%, #059669 100%);
    color: white;
    border: none;
    border-radius: 10px;
    padding: 14px 28px;
    font-weight: 700;
    font-size: 1em;
    width: 100%;
    box-shadow: 0 6px 20px rgba(16, 185, 129, 0.3);
    text-transform: uppercase;
    letter-spacing: 0.5px;
}

.stDownloadButton > button:hover {
    transform: translateY(-3px);
    box-shadow: 0 10px 30px rgba(16, 185, 129, 0.4);
}

/* Column containers - with cards */
[data-testid="column"] {
    background: linear-gradient(135deg, #ffffff 0%, #f0f9ff 100%);
    padding: 25px;
    border-radius: 12px;
    border: 2px solid #bfdbfe;
    box-shadow: 0 6px 20px rgba(30, 64, 175, 0.1);
}

/* Expander styling */
.streamlit-expanderHeader {
    background: linear-gradient(135deg, #f0f9ff 0%, #ffffff 100%);
    border-radius: 8px;
    border: 2px solid #dbeafe;
}

.streamlit-expanderHeader p {
    color: #1e40af;
    font-weight: 700;
}

/* General text */
p, label, span, li, div {
    color: #1e293b;
```

```
            }

            /* Dividers - colorful */
            hr {
                border: none;
                height: 2px;
                background: linear-gradient(90deg, transparent, #bfdbfe, transparent);
                margin: 25px 0;
            }

            /* Checkbox styling */
            .stCheckbox {
                margin: 12px 0;
            }
        </style>
    """, unsafe_allow_html=True)

    st.title("🛡️ Legal Document Redaction")

    st.markdown("""
        <div class="subtitle">
        🔒 Secure document redaction using LDR-NER AI • Redacts sensitive PII and structured data
        </div>
    """, unsafe_allow_html=True)

    st.markdown("---")

    # Load NER model name and redaction function from inference script
    st.info(f"🤖 Model: **{HF_MODEL_NAME}** | Architecture: Token Classification (LDR-NER)")

    # ------------------------------
    # Text Extraction Functions
    # ------------------------------
    def extract_text_from_pdf(uploaded_file):
        """
        Extract text from a PDF file using PdfReader.
        Args:
            uploaded_file: Uploaded PDF file object
        Returns:
            Extracted text as a string
        """
        reader = PdfReader(uploaded_file)
        text = ""
        for page in reader.pages:
            if page.extract_text():
                text += page.extract_text() + "\n"
        return text.strip()

    def extract_text_from_docx(uploaded_file):
        """
        Extract text from a DOCX file using python-docx.
        Args:
            uploaded_file: Uploaded DOCX file object
        Returns:
            Extracted text as a string
        """
        doc = Document(uploaded_file)
        text = ""
        for para in doc.paragraphs:
            text += para.text + "\n"
        return text.strip()

    def extract_text_from_txt(uploaded_file):
        """
        Extract text from a TXT file.
        Args:
            uploaded_file: Uploaded TXT file object
        Returns:
            Extracted text as a string
        """
        return uploaded_file.read().decode("utf-8").strip()

    def extract_text_from_file(uploaded_file):
        """Extract text from any supported file type"""
        """
        Extract text from any supported file type (PDF, DOCX, TXT).
        Args:
            uploaded_file: Uploaded file object
```

```
        uploaded_file: Uploaded file object
    Returns:
        Extracted text as a string, or None if unsupported or error
    """
    file_extension = uploaded_file.name.split('.')[-1].lower()

    try:
        if file_extension == "pdf":
            return extract_text_from_pdf(uploaded_file)
        elif file_extension in ["docx", "doc"]:
            return extract_text_from_docx(uploaded_file)
        elif file_extension == "txt":
            return extract_text_from_txt(uploaded_file)
        else:
            return None
    except Exception as e:
        st.error(f"Error extracting text: {str(e)}")
        return None

# ------------------------------
# Sidebar Controls
# ------------------------------
st.sidebar.header("Redaction Settings")

# Display what entities are being redacted with colors
st.sidebar.markdown("### 🎯 Select Entities to Redact")

# Define colors for each entity type
entity_colors = {
    "PER": "#FF6B6B",    # Red for Person
    "ORG": "#95E1D3",    # Light green for Organization
    "LOC": "#4ECDC4",    # Teal for Location
    "CODE": "#FFD93D",   # Yellow for Code
    "DATETIME": "#6BCB77",  # Green for Date/Time
    "DEM": "#FF6B9D",    # Pink for Demographic
    "MISC": "#A8D8EA",   # Light blue for Miscellaneous
    "QUANTITY": "#AA96DA"  # Purple for Quantity
}

# Multi-select for entity types
entity_options = list(REDACTED_ENTITIES.keys())
entity_display = [f"{code} - {REDACTED_ENTITIES[code]}" for code in entity_options]
selected_display = st.sidebar.multiselect(
    "Choose which entities to redact:",
    entity_display,
    default=entity_display,  # All selected by default
    key="entity_selector"
)

# Extract selected entity codes - ensure always list
selected_entities = []
for display in selected_display:
    code = display.split(" - ")[0].strip()
    if code:
        selected_entities.append(code)

# Fallback to all entities if nothing selected
if not selected_entities:
    selected_entities = list(REDACTED_ENTITIES.keys())

st.sidebar.markdown("---")

# ------------------------------
# Input Options
# ------------------------------
input_method = st.radio(
    "Choose input method:",
    ["Upload File", "Paste Text"],
    horizontal=True
)

extracted_text = None

if input_method == "Upload File":
    uploaded_file = st.file_uploader(
        "Upload a file (PDF, Word, or Text)",
        type=["pdf", "docx", "doc", "txt"]
    )
```

```
        if uploaded_file:
            with st.spinner("Extracting text from file..."):
                extracted_text = extract_text_from_file(uploaded_file)

            if not extracted_text:
                st.error("No readable text found in the file.")
                st.stop()

    else:
        extracted_text = st.text_area(
            "Paste your text here:",
            height=200,
            placeholder="Enter or paste the text you want to redact..."
        )

    if extracted_text:
        col1, col2 = st.columns(2)

        with col1:
            st.subheader("Original Text")
            st.text_area(
                label="Original Text",
                value=extracted_text,
                height=500,
                disabled=True,
                label_visibility="collapsed"
            )

        # Apply redaction using inference script logic
        with st.spinner("Running LDR-NER redaction..."):
            try:
                # Use the imported inference script's redaction function
                processed_text = redact_text(extracted_text, selected_entities)
            except Exception as e:
                st.error(f"Error during redaction: {str(e)}")
                processed_text = extracted_text

        with col2:
            st.subheader("Redacted Text")
            st.text_area(
                label="Redacted Text",
                value=processed_text,
                height=500,
                disabled=True,
                label_visibility="collapsed"
            )

        # Download option
        st.download_button(
            label="⬇️ Download Redacted Text",
            data=processed_text,
            file_name="redacted_output.txt",
            mime="text/plain"
        )

    else:
        st.info("Upload a file (PDF, Word, Text) or paste text to start redaction.")
=======
"""
Streamlit App for Legal Document Redaction using BERT-NER

This app allows users to upload or paste text from PDF, Word, or Text files and redact sensitive information using a NER model.
"""

import streamlit as st
from pypdf import PdfReader
import re
from docx import Document
import io
import os

# Import inference script from the same folder
from inference_script import redact_text, REDACTED_ENTITIES, HF_MODEL_NAME

# ------------------------------
# Streamlit Page Config
# ------------------------------
```

```
# ------------------------------
st.set_page_config(
    page_title="PDF Text Redaction (BERT-NER)",
    layout="wide"
)

# Custom CSS for better styling
st.markdown("""
    <style>
        /* Root and main styling */
        * {
            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
        }

        /* Full page background - vibrant gradient */
        .stApp {
            background: linear-gradient(135deg, #f0f4ff 0%, #e8f5ff 50%, #e8f9ff 100%);
            min-height: 100vh;
        }

        /* Sidebar with gradient */
        [data-testid="stSidebar"] {
            background: linear-gradient(180deg, #1e40af 0%, #1e3a8a 100%) !important;
            padding: 0 !important;
        }

        [data-testid="stSidebar"] > div:first-child {
            padding: 20px;
        }

        [data-testid="stSidebar"] [data-testid="stVerticalBlockBorderWrapper"] {
            background: rgba(255, 255, 255, 0.95);
            border-radius: 12px;
            margin: 15px 10px;
            padding: 15px;
            border: 2px solid rgba(255, 255, 255, 0.3);
            box-shadow: 0 8px 32px rgba(30, 64, 175, 0.15);
        }

        [data-testid="stSidebar"] h1,
        [data-testid="stSidebar"] h2,
        [data-testid="stSidebar"] h3 {
            color: #ffffff;
            margin-top: 0;
            font-weight: 700;
            border-bottom: 2px solid rgba(255, 255, 255, 0.3);
            padding-bottom: 10px;
        }

        [data-testid="stSidebar"] label {
            color: #ffffff !important;
            font-weight: 600 !important;
            font-size: 0.95em !important;
        }

        [data-testid="stSidebar"] .stCheckbox label {
            color: #ffffff !important;
        }

        [data-testid="stSidebar"] .stCheckbox label span {
            color: #ffffff !important;
        }

        [data-testid="stSidebar"] div[role="radiogroup"] label {
            color: #ffffff !important;
        }

        [data-testid="stSidebar"] p {
            color: #ffffff !important;
        }

        [data-testid="stSidebar"] span {
            color: #ffffff !important;
        }

        [data-testid="stSidebar"] div {
            color: #ffffff !important;
        }
```

```
/* Main content padding and styling */
.main > div:first-child {
    padding-top: 30px;
    padding-bottom: 30px;
    padding-left: 40px;
    padding-right: 40px;
}

/* Title styling - eye-catching */
h1 {
    background: linear-gradient(135deg, #1e40af 0%, #0ea5e9 100%);
    -webkit-background-clip: text;
    -webkit-text-fill-color: transparent;
    background-clip: text;
    font-size: 3.5em;
    margin-bottom: 10px;
    margin-top: 20px;
    font-weight: 900;
    text-align: center;
    letter-spacing: -1px;
    text-shadow: 0 2px 4px rgba(0,0,0,0.1);
}

/* Subtitle styling */
.subtitle {
    text-align: center;
    color: #475569;
    font-size: 1.15em;
    margin-bottom: 30px;
    font-weight: 600;
    letter-spacing: 0.5px;
}

/* Status badge styling */
.stSuccess, .stInfo {
    background: linear-gradient(135deg, #ecfdf5 0%, #d1fae5 100%);
    border-left: 5px solid #10b981;
    border-radius: 10px;
    color: #047857;
    padding: 15px;
    font-size: 0.98em;
    font-weight: 500;
    box-shadow: 0 4px 15px rgba(16, 185, 129, 0.1);
}

/* Section headers */
h2 {
    color: #1e40af;
    font-size: 1.8em;
    margin-top: 30px;
    margin-bottom: 20px;
    font-weight: 800;
    position: relative;
    padding-bottom: 15px;
}

h2::after {
    content: '';
    position: absolute;
    bottom: 0;
    left: 0;
    width: 60px;
    height: 4px;
    background: linear-gradient(90deg, #1e40af, #0ea5e9);
    border-radius: 2px;
}

h3 {
    color: #1e3a8a;
    font-size: 1.3em;
    margin-top: 20px;
    margin-bottom: 12px;
    font-weight: 700;
}

/* Radio button group - card style */
```

```
.stRadio {
    background: linear-gradient(135deg, #ffffff 0%, #f0f9ff 100%);
    padding: 20px;
    border-radius: 12px;
    border: 2px solid #bfdbfe;
    margin: 15px 0;
    box-shadow: 0 4px 15px rgba(30, 64, 175, 0.08);
}

.stRadio label {
    color: #1e293b !important;
    font-weight: 600;
    font-size: 1em;
}

/* Input fields - styled cards */
textarea, input[type="text"], input[type="email"], input[type="password"] {
    border: 2px solid #bfdbfe !important;
    border-radius: 10px !important;
    background: linear-gradient(135deg, #f8fafc 0%, #f0f9ff 100%) !important;
    color: #0f172a !important;
    font-size: 0.95em !important;
    padding: 12px !important;
    box-shadow: 0 2px 8px rgba(30, 64, 175, 0.05) !important;
}

textarea:focus, input:focus {
    border-color: #0ea5e9 !important;
    background: #ffffff !important;
    box-shadow: 0 0 0 4px rgba(14, 165, 233, 0.15) !important;
}

/* Buttons - vibrant and interactive */
.stButton > button {
    background: linear-gradient(135deg, #1e40af 0%, #0ea5e9 100%);
    color: white;
    border: none;
    border-radius: 10px;
    padding: 14px 28px;
    font-weight: 700;
    font-size: 1em;
    transition: all 0.3s ease;
    width: 100%;
    cursor: pointer;
    box-shadow: 0 6px 20px rgba(30, 64, 175, 0.3);
    text-transform: uppercase;
    letter-spacing: 0.5px;
}

.stButton > button:hover {
    transform: translateY(-3px);
    box-shadow: 0 10px 30px rgba(30, 64, 175, 0.4);
    background: linear-gradient(135deg, #0ea5e9 0%, #1e40af 100%);
}

.stButton > button:active {
    transform: translateY(-1px);
}

/* Download buttons - success color */
.stDownloadButton > button {
    background: linear-gradient(135deg, #10b981 0%, #059669 100%);
    color: white;
    border: none;
    border-radius: 10px;
    padding: 14px 28px;
    font-weight: 700;
    font-size: 1em;
    width: 100%;
    box-shadow: 0 6px 20px rgba(16, 185, 129, 0.3);
    text-transform: uppercase;
    letter-spacing: 0.5px;
}

.stDownloadButton > button:hover {
    transform: translateY(-3px);
    box-shadow: 0 10px 30px rgba(16, 185, 129, 0.4);
}
```

```
        /* Column containers - with cards */
        [data-testid="column"] {
            background: linear-gradient(135deg, #ffffff 0%, #f0f9ff 100%);
            padding: 25px;
            border-radius: 12px;
            border: 2px solid #bfdbfe;
            box-shadow: 0 6px 20px rgba(30, 64, 175, 0.1);
        }

        /* Expander styling */
        .streamlit-expanderHeader {
            background: linear-gradient(135deg, #f0f9ff 0%, #ffffff 100%);
            border-radius: 8px;
            border: 2px solid #dbeafe;
        }

        .streamlit-expanderHeader p {
            color: #1e40af;
            font-weight: 700;
        }

        /* General text */
        p, label, span, li, div {
            color: #1e293b;
        }

        /* Dividers - colorful */
        hr {
            border: none;
            height: 2px;
            background: linear-gradient(90deg, transparent, #bfdbfe, transparent);
            margin: 25px 0;
        }

        /* Checkbox styling */
        .stCheckbox {
            margin: 12px 0;
        }
    </style>
""", unsafe_allow_html=True)

st.title("🛡️ Legal Document Redaction")

st.markdown("""
    <div class="subtitle">
    🔒 Secure document redaction using LDR-NER AI • Redacts sensitive PII and structured data
    </div>
""", unsafe_allow_html=True)

st.markdown("---")

# Load NER model name and redaction function from inference script
st.info(f"🤖 Model: **{HF_MODEL_NAME}** | Architecture: Token Classification (LDR-NER)")

# ------------------------------
# Text Extraction Functions
# ------------------------------
def extract_text_from_pdf(uploaded_file):
    """
    Extract text from a PDF file using PdfReader.
    Args:
        uploaded_file: Uploaded PDF file object
    Returns:
        Extracted text as a string
    """
    reader = PdfReader(uploaded_file)
    text = ""
    for page in reader.pages:
        if page.extract_text():
            text += page.extract_text() + "\n"
    return text.strip()

def extract_text_from_docx(uploaded_file):
    """
    Extract text from a DOCX file using python-docx.
    Args:
```

```
            uploaded_file: Uploaded DOCX file object
        Returns:
            Extracted text as a string
        """
        doc = Document(uploaded_file)
        text = ""
        for para in doc.paragraphs:
            text += para.text + "\n"
        return text.strip()

    def extract_text_from_txt(uploaded_file):
        """
        Extract text from a TXT file.
        Args:
            uploaded_file: Uploaded TXT file object
        Returns:
            Extracted text as a string
        """
        return uploaded_file.read().decode("utf-8").strip()

    def extract_text_from_file(uploaded_file):
        """Extract text from any supported file type"""
        """
        Extract text from any supported file type (PDF, DOCX, TXT).
        Args:
            uploaded_file: Uploaded file object
        Returns:
            Extracted text as a string, or None if unsupported or error
        """
        file_extension = uploaded_file.name.split('.')[-1].lower()

        try:
            if file_extension == "pdf":
                return extract_text_from_pdf(uploaded_file)
            elif file_extension in ["docx", "doc"]:
                return extract_text_from_docx(uploaded_file)
            elif file_extension == "txt":
                return extract_text_from_txt(uploaded_file)
            else:
                return None
        except Exception as e:
            st.error(f"Error extracting text: {str(e)}")
            return None

    # ------------------------------
    # Sidebar Controls
    # ------------------------------
    st.sidebar.header("Redaction Settings")

    # Display what entities are being redacted with colors
    st.sidebar.markdown("### 🎯 Select Entities to Redact")

    # Define colors for each entity type
    entity_colors = {
        "PER": "#FF6B6B",    # Red for Person
        "ORG": "#95E1D3",    # Light green for Organization
        "LOC": "#4ECDC4",    # Teal for Location
        "CODE": "#FFD93D",   # Yellow for Code
        "DATETIME": "#6BCB77",  # Green for Date/Time
        "DEM": "#FF6B9D",    # Pink for Demographic
        "MISC": "#A8D8EA",   # Light blue for Miscellaneous
        "QUANTITY": "#AA96DA"  # Purple for Quantity
    }

    # Multi-select for entity types
    entity_options = list(REDACTED_ENTITIES.keys())
    entity_display = [f"{code} - {REDACTED_ENTITIES[code]}" for code in entity_options]
    selected_display = st.sidebar.multiselect(
        "Choose which entities to redact:",
        entity_display,
        default=entity_display,  # All selected by default
        key="entity_selector"
    )

    # Extract selected entity codes - ensure always list
    selected_entities = []
    for display in selected_display:
        code = display.split(" - ")[0].strip()
```

```
        if code:
            selected_entities.append(code)

    # Fallback to all entities if nothing selected
    if not selected_entities:
        selected_entities = list(REDACTED_ENTITIES.keys())

    st.sidebar.markdown("---")

    # -----------------------------
    # Input Options
    # -----------------------------
    input_method = st.radio(
        "Choose input method:",
        ["Upload File", "Paste Text"],
        horizontal=True
    )

    extracted_text = None

    if input_method == "Upload File":
        uploaded_file = st.file_uploader(
            "Upload a file (PDF, Word, or Text)",
            type=["pdf", "docx", "doc", "txt"]
        )

        if uploaded_file:
            with st.spinner("Extracting text from file..."):
                extracted_text = extract_text_from_file(uploaded_file)

            if not extracted_text:
                st.error("No readable text found in the file.")
                st.stop()

    else:
        extracted_text = st.text_area(
            "Paste your text here:",
            height=200,
            placeholder="Enter or paste the text you want to redact..."
        )

    if extracted_text:
        col1, col2 = st.columns(2)

        with col1:
            st.subheader("Original Text")
            st.text_area(
                label="Original Text",
                value=extracted_text,
                height=500,
                disabled=True,
                label_visibility="collapsed"
            )

        # Apply redaction using inference script logic
        with st.spinner("Running LDR-NER redaction..."):
            try:
                # Use the imported inference script's redaction function
                processed_text = redact_text(extracted_text, selected_entities)
            except Exception as e:
                st.error(f"Error during redaction: {str(e)}")
                processed_text = extracted_text

        with col2:
            st.subheader("Redacted Text")
            st.text_area(
                label="Redacted Text",
                value=processed_text,
                height=500,
                disabled=True,
                label_visibility="collapsed"
            )

        # Download option
        st.download_button(
            label="⬇️ Download Redacted Text",
            data=processed_text,
```

```
            file_name="redacted_output.txt",
            mime="text/plain"
        )

    else:
        st.info("Upload a file (PDF, Word, Text) or paste text to start redaction.")
```

Inference Script

```
"""
Inference module for legal document redaction using BERT-NER

This module provides the redaction functionality for the Streamlit app. It loads a Hugging Face NER model and exposes functions t
"""
from transformers import pipeline, AutoTokenizer, AutoModelForTokenClassification
import os

# Model configuration
HF_MODEL_NAME = "sayeedlearnstech/LDR-NER"
model_path = "ldr_ner_model"

# Entity types that can be redacted
REDACTED_ENTITIES = {
    "PER": "Person Names",
    "ORG": "Organizations",
    "LOC": "Locations",
    "CODE": "Code",
    "DATETIME": "Date/Time",
    "DEM": "Demographic Info",
    "MISC": "Miscellaneous",
    "QUANTITY": "Quantities"
}

def download_model():
    """
    Download the model and tokenizer from Hugging Face if not available locally.
    Returns:
        True if download and save successful, False otherwise.
    """
    print(f"Model not found locally. Downloading from Hugging Face...")
    print(f"Model: {HF_MODEL_NAME}")
    print("This may take a few minutes...")
    try:
        os.makedirs(model_path, exist_ok=True)
        print("Downloading tokenizer...")
        tokenizer = AutoTokenizer.from_pretrained(HF_MODEL_NAME)
        print("Downloading model...")
        model = AutoModelForTokenClassification.from_pretrained(HF_MODEL_NAME)
        print(f"Saving to {model_path}...")
        tokenizer.save_pretrained(model_path)
        model.save_pretrained(model_path)
        print(f"✅ Model successfully downloaded and saved!")
        return True
    except Exception as e:
        print(f"❌ Error downloading model: {e}")
        return False

# Initialize the NER pipeline
my_ner = None
try:
    # Try to load from local folder first
    if not os.path.exists(model_path):
        print(f"Local model not found at '{model_path}'")
        if download_model():
            print("Attempting to load downloaded model...")
        else:
            raise Exception("Failed to download model")

    my_ner = pipeline(
        "token-classification",
        model=model_path,
        tokenizer=model_path,
        aggregation_strategy="simple"
    )
```

```
        print(f"✅ Model loaded successfully from '{model_path}'")
    except Exception as e:
        print(f"❌ Error loading model: {e}")
        my_ner = None


    def redact_text(text, selected_entities=None):
        """
        Redact sensitive information from text using the NER model.
        Redacts: CODE, DATETIME, DEM, LOC, MISC, ORG, PER, QUANTITY, and O (other tags)

        Args:
            text (str): Input text to redact
            selected_entities (list, optional): List of entity types to redact (e.g., ["PER", "LOC", "ORG"]).
                If None, redacts all entity types.

        Returns:
            str: Redacted text with sensitive info replaced by black lines (█)
        """
        if my_ner is None:
            raise Exception("NER model not loaded. Please check the model path.")
        # Default to all entities if none specified
        if selected_entities is None:
            selected_entities = list(REDACTED_ENTITIES.keys())
        entities = my_ner(text)
        entities = sorted(entities, key=lambda x: x['start'], reverse=True)
        for item in entities:
            if item['entity_group'] in selected_entities:
                start = item['start']
                end = item['end']
                redacted_text = text[start:end]
                # Replace with black line (█) maintaining text length
                text = text[:start] + "█" * len(redacted_text) + text[end:]
        return text


    def get_ner_pipeline():
        """
        Get the NER pipeline for advanced usage.
        Returns:
            transformers.Pipeline: The loaded NER pipeline, or None if not loaded.
        """
        return my_ner
```