# ==========================

# Section 0: Import libraries

# ==========================

import streamlit as st

import os

from dotenv import load\_dotenv

from typing import Dict, List, Tuple

import pandas as pd

from urllib.parse import urlparse

import requests

from bs4 import BeautifulSoup

import re

from get\_url import get\_privacy\_policy\_url

from summarize\_text import get\_summary\_for\_tos

from check\_cache import LLMCacheTool

import openai

from transformers import AutoTokenizer, AutoModelForSequenceClassification, pipeline

import torch

import numpy as np

from textblob import TextBlob

# Global variables

BERT\_MODEL = None

MAX\_TEXT\_LENGTH = 10000

CACHE\_DAYS = 7

MIN\_INPUT\_LENGTH = 3

def check\_requirements():

try:

import transformers

import torch

import textblob

except ImportError as e:

st.error(f"Missing required package: {str(e)}")

st.info("Please install required packages: pip install transformers torch textblob")

st.stop()

def configure():

"""Load environment variables and initialize API keys"""

load\_dotenv()

openai.api\_key = os.getenv("OPENAI\_API\_KEY")

def get\_bert\_model():

"""Initialize or retrieve cached BERT model"""

global BERT\_MODEL

if BERT\_MODEL is None:

try:

BERT\_MODEL = pipeline(

"text-classification",

model="nlptown/bert-base-multilingual-uncased-sentiment",

tokenizer="nlptown/bert-base-multilingual-uncased-sentiment",

device=-1 # Use CPU if GPU not available

)

except Exception as e:

print(f"Error loading BERT model: {str(e)}")

return None

return BERT\_MODEL

def chunk\_text(text: str, max\_length: int = 512) -> List[str]:

"""Split text into manageable chunks"""

sentences = text.split('.')

chunks = []

current\_chunk = ""

for sentence in sentences:

if len(current\_chunk) + len(sentence) < max\_length:

current\_chunk += sentence + "."

else:

if current\_chunk:

chunks.append(current\_chunk)

current\_chunk = sentence + "."

if current\_chunk:

chunks.append(current\_chunk)

return chunks

# ==========================

# Section 1: Configure page

# ==========================

st.set\_page\_config(

page\_title="LegalLens - Privacy Policy Analyzer",

page\_icon="⚖️",

layout="wide"

)

# ✅ Unified CSS for compact layout and alignment

st.markdown("""

<style>

/\* Shrink block container padding \*/

.block-container {

padding-top: 1rem !important;

padding-bottom: 1rem !important;

padding-left: 1rem !important;

padding-right: 1rem !important;

}

/\* Tighten column content spacing \*/

.stColumn > div {

padding-top: 0.5rem;

padding-bottom: 0.5rem;

}

/\* Reduce margin above/below headings \*/

h1, h2, h3, h4 {

margin-top: 0.25rem;

margin-bottom: 0.25rem;

}

/\* Reduce space in expanders \*/

details summary {

font-size: 16px;

margin-bottom: 0;

}

/\* Optional: Compact metrics \*/

div[data-testid="metric-container"] {

padding: 0.25rem;

margin: 0;

}

/\* Align elements inside columns to the top \*/

.stColumn {

vertical-align: top;

}

/\* Fix expander height so both columns align \*/

.stExpander {

min-height: 180px;

}

/\* Optional: Prevent large gaps inside markdown boxes \*/

.stMarkdown {

margin-bottom: 0.5rem;

}

</style>

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

# Custom CSS for risk colors

st.markdown("""

<style>

.high-risk { color: #FF0000; }

.medium-risk { color: #FFA500; }

.low-risk { color: #008000; }

.risk-header-high { color: #FF0000; font-size: 20px; font-weight: bold; }

.risk-header-medium { color: #FFA500; font-size: 20px; font-weight: bold; }

.risk-header-low { color: #008000; font-size: 20px; font-weight: bold; }

.risk-box {

padding: 10px;

border-radius: 5px;

margin: 5px;

background-color: rgba(255, 255, 255, 0.1);

}

.summary-high { color: #FF0000; }

.summary-medium { color: #FFA500; }

.summary-low { color: #008000; }

.progress-bar {

height: 20px;

background-color: #f0f0f0;

border-radius: 10px;

margin: 10px 0;

}

.progress-fill {

height: 100%;

border-radius: 10px;

transition: width 0.3s ease;

}

</style>

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

# ==========================

# Section 2: Helper functions

# ==========================

def get\_root\_url(url: str) -> str:

"""Extract the root URL from a given URL"""

parsed = urlparse(url)

return f"{parsed.scheme}://{parsed.netloc}"

def scrape\_text\_direct(url: str) -> str:

"""Directly scrape text from a given URL"""

try:

headers = {

'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36'

}

response = requests.get(url, headers=headers, timeout=10)

response.raise\_for\_status()

soup = BeautifulSoup(response.text, 'html.parser')

# Remove script and style elements

for script in soup(["script", "style"]):

script.decompose()

# Find main content (usually in specific tags)

main\_content = soup.find('main') or soup.find('article') or soup.find('div', class\_='content') or soup

text = main\_content.get\_text()

lines = (line.strip() for line in text.splitlines())

chunks = (phrase.strip() for line in lines for phrase in line.split(" "))

text = ' '.join(chunk for chunk in chunks if chunk)

return text

except Exception as e:

raise Exception(f"Error scraping {url}: {str(e)}")

def scrape\_text(url: str) -> str:

"""Main function to scrape text from URL with improved error handling"""

try:

# Try direct URL first

text = scrape\_text\_direct(url)

if len(text) > MIN\_INPUT\_LENGTH:

return text

# Try root URL if direct URL fails

root\_url = get\_root\_url(url)

if root\_url != url:

text = scrape\_text\_direct(root\_url)

if len(text) > MIN\_INPUT\_LENGTH:

return text

# Try common privacy policy paths

common\_paths = [

'/privacy',

'/privacy-policy',

'/privacy\_policy',

'/legal/privacy',

'/about/privacy'

]

for path in common\_paths:

try:

policy\_url = f"{root\_url}{path}"

text = scrape\_text\_direct(policy\_url)

if len(text) > MIN\_INPUT\_LENGTH:

return text

except:

continue

raise Exception("Could not find privacy policy text")

except Exception as e:

raise Exception(f"Error scraping text: {str(e)}")

def calculate\_risk\_score(risks: Dict[str, List[str]]) -> int:

"""Calculate overall risk score based on findings"""

score = 100

# Deduct points for each risk

score -= len([r for r in risks['high'] if not r.startswith('No')]) \* 15

score -= len([r for r in risks['medium'] if not r.startswith('No')]) \* 7

score -= len([r for r in risks['low'] if not r.startswith('No')]) \* 3

return max(0, min(score, 100))

def generate\_recommendations(risks: Dict[str, List[str]], score: int) -> List[str]:

"""Generate contextual recommendations based on risk keywords"""

recommendations = []

# 🚨 High-risk triggers

if any("sell" in r.lower() or "share" in r.lower() for r in risks['high']):

recommendations.append("🚨 Consider limiting data sharing by checking your privacy settings.")

if any("track" in r.lower() or "location" in r.lower() for r in risks['high']):

recommendations.append("📍 Disable location tracking in your account and device settings.")

if any("biometric" in r.lower() or "facial recognition" in r.lower() for r in risks['high']):

recommendations.append("🧬 Reconsider using services requiring biometric data without consent.")

# ⚠️ Medium-risk triggers

if any("cookie" in r.lower() or "ads" in r.lower() for r in risks['medium']):

recommendations.append("🍪 Install a cookie manager browser extension to reduce tracking.")

if any("analytics" in r.lower() for r in risks['medium']):

recommendations.append("📊 Use privacy-focused browsers or analytics blockers like uBlock Origin.")

# 📉 Score-based

if score < 50:

recommendations.append("⚠️ Overall privacy score is low. Consider using an alternative service.")

# ✅ Default fallback

if not recommendations:

recommendations.append("✅ This privacy policy seems reasonable. No immediate action needed.")

return recommendations

# ==========================

# Section 3: Analysis Functions

# ==========================

def analyze\_privacy\_policy\_with\_bert(text: str) -> Dict[str, List[str]]:

"""

Analyze privacy policy text using BERT with improved error handling and progress tracking

"""

try:

# Get or initialize BERT model

classifier = get\_bert\_model()

if classifier is None:

return analyze\_privacy\_policy\_with\_textblob(text)

# Define comprehensive risk patterns

risk\_patterns = {

'high': [

'sell your data', 'share your data', 'third party sharing',

'track location', 'biometric data', 'facial recognition',

'sell personal information', 'share personal information',

'data broker', 'surveillance', 'data mining',

'right to publish', 'use your content',

'without consent', 'without explanation',

'reject without notice', 'remove without notice',

'share with third parties', 'transfer your data',

'unlimited rights', 'perpetual license',

'misuse of personal data', 'unauthorized access'

],

'medium': [

'cookie tracking', 'analytics', 'advertising',

'marketing email', 'promotional content', 'user profiling',

'targeted ads', 'tracking pixels', 'social media integration',

'collect information', 'store information', 'process data',

'usage statistics', 'behavioral data'

],

'low': [

'contact information', 'email address', 'basic account info',

'security measure', 'encryption', 'necessary data',

'customer service', 'basic analytics', 'technical data',

'improve service', 'maintain service', 'essential cookies'

]

}

# Add context words that increase risk level

risk\_amplifiers = [

'without', 'unlimited', 'any purpose', 'all rights',

'may', 'could', 'might', 'reserve the right',

'at our discretion', 'not responsible', 'no liability'

]

# Initialize risks dictionary

risks = {

'high': [],

'medium': [],

'low': []

}

# Split text into manageable chunks

chunks = chunk\_text(text)

total\_chunks = len(chunks)

# Initialize progress tracking

progress\_bar = st.progress(0)

progress\_text = st.empty()

# Analyze each chunk

for i, chunk in enumerate(chunks):

try:

# Update progress

progress = (i + 1) / total\_chunks

progress\_bar.progress(progress)

progress\_text.text(f"Analyzing section {i+1} of {total\_chunks}")

# Get BERT sentiment

result = classifier(chunk[:512])[0]

score = int(result['label'][0])

# Analyze sentences within chunk

sentences = [s.strip() for s in chunk.split('.') if len(s.strip()) > 20]

for sentence in sentences:

sentence\_lower = sentence.lower()

# Check for risk patterns

for risk\_level, patterns in risk\_patterns.items():

for pattern in patterns:

if pattern.lower() in sentence\_lower:

# Check for risk amplifiers

has\_amplifier = any(amp in sentence\_lower for amp in risk\_amplifiers)

# Determine final risk level

final\_risk\_level = risk\_level

if has\_amplifier:

if risk\_level == 'low':

final\_risk\_level = 'medium'

elif risk\_level == 'medium':

final\_risk\_level = 'high'

# Add sentiment indicators

formatted\_sentence = sentence.strip()

if score <= 2: # Negative sentiment

formatted\_sentence = "⚠️ " + formatted\_sentence

elif score >= 4: # Positive sentiment

formatted\_sentence = "✓ " + formatted\_sentence

risks[final\_risk\_level].append(formatted\_sentence)

except Exception as e:

print(f"Error processing chunk {i}: {str(e)}")

continue

# Clean up progress indicators

progress\_bar.empty()

progress\_text.empty()

# Process and clean up results

for risk\_level in risks:

# Remove duplicates while preserving order

seen = set()

risks[risk\_level] = [x for x in risks[risk\_level]

if not (x in seen or seen.add(x))]

# Limit length and clean up

risks[risk\_level] = [

(s[:150] + '...') if len(s) > 150 else s

for s in risks[risk\_level]

]

# Add default message if no risks found

if not risks[risk\_level]:

risks[risk\_level].append(f"No {risk\_level}-risk items identified")

return risks

except Exception as e:

print(f"BERT analysis failed, falling back to TextBlob: {str(e)}")

return analyze\_privacy\_policy\_with\_textblob(text)

def explain\_risk\_categorization():

"""Explain how risks are categorized"""

st.markdown("""

### How Risks Are Categorized

🔴 \*\*High Risk Items\*\*:

- Data selling or sharing with third parties

- Collection of sensitive data (biometric, location)

- Unlimited rights to user content

- Lack of user consent or notification

- Potential for data misuse or unauthorized access

🟡 \*\*Medium Risk Items\*\*:

- Cookie tracking and analytics

- Marketing and advertising practices

- User profiling and behavioral tracking

- Data collection for non-essential purposes

- Social media integration

🟢 \*\*Low Risk Items\*\*:

- Basic account information

- Essential service functionality

- Security measures and encryption

- Customer service communication

- Technical maintenance data

\*\*Note\*\*: Risk levels may be elevated if the policy:

- Uses vague or permissive language ("may", "could", "at our discretion")

- Lacks clear user controls or consent mechanisms

- Contains broad or unlimited rights claims

""")

def analyze\_privacy\_policy\_with\_textblob(text: str) -> Dict[str, List[str]]:

"""

Fallback analysis using TextBlob with improved pattern matching

"""

try:

# Define comprehensive risk patterns (same as BERT)

risk\_patterns = {

'high': [

'sell your data', 'share your data', 'third party',

'track location', 'biometric data', 'facial recognition',

'sell personal information', 'share personal information',

'data broker', 'surveillance', 'data mining', 'behavioral tracking'

],

'medium': [

'cookie', 'analytic', 'advertising', 'marketing email',

'promotional', 'profiling', 'targeted ads', 'tracking pixel',

'social media', 'device information', 'usage data'

],

'low': [

'contact information', 'email address', 'basic account',

'security measure', 'encryption', 'necessary data',

'customer service', 'basic analytics', 'technical data'

]

}

risks = {

'high': [],

'medium': [],

'low': []

}

# Create TextBlob object and analyze

blob = TextBlob(text.lower())

# Show progress

progress\_bar = st.progress(0)

progress\_text = st.empty()

total\_sentences = len(blob.sentences)

for i, sentence in enumerate(blob.sentences):

# Update progress

progress = (i + 1) / total\_sentences

progress\_bar.progress(progress)

progress\_text.text(f"Analyzing sentence {i+1} of {total\_sentences}")

sentence\_text = str(sentence)

if len(sentence\_text) < 20:

continue

sentiment = sentence.sentiment.polarity

for risk\_level, patterns in risk\_patterns.items():

for pattern in patterns:

if pattern.lower() in sentence\_text.lower():

risk\_entry = sentence\_text.strip()

if sentiment < -0.1: # Negative sentiment

risk\_entry = "⚠️ " + risk\_entry

elif sentiment > 0.1: # Positive sentiment

risk\_entry = "✓ " + risk\_entry

risks[risk\_level].append(risk\_entry)

# Clean up progress indicators

progress\_bar.empty()

progress\_text.empty()

# Clean up results

for risk\_level in risks:

risks[risk\_level] = list(set(risks[risk\_level]))

risks[risk\_level] = [

(s[:150] + '...') if len(s) > 150 else s

for s in risks[risk\_level]

]

if not risks[risk\_level]:

risks[risk\_level].append(f"No {risk\_level}-risk items identified")

return risks

except Exception as e:

print(f"Error in TextBlob analysis: {str(e)}")

return {

'high': ['Analysis Error - Please try again'],

'medium': ['Analysis Error - Please try again'],

'low': ['Analysis Error - Please try again']

}

def analyze\_privacy\_policy(text: str) -> Dict[str, List[str]]:

"""

Main analysis function with improved error handling

"""

try:

with st.spinner("Analyzing with BERT..."):

return analyze\_privacy\_policy\_with\_bert(text)

except Exception as e:

st.warning("BERT analysis failed, falling back to TextBlob")

with st.spinner("Analyzing with TextBlob..."):

return analyze\_privacy\_policy\_with\_textblob(text)

# ==========================

# Section 4: Display Functions

# ==========================

def format\_summary\_with\_risks(summary: str, risks: Dict[str, List[str]]) -> str:

"""Enhanced summary formatting with better risk highlighting and context"""

formatted\_summary = summary

# Create a risk dictionary for quick lookup

risk\_terms = {}

for risk\_level, items in risks.items():

for item in items:

# Remove the warning symbols if present

clean\_item = item.replace('⚠️ ', '').replace('✓ ', '')

# Take only the first part if there's an ellipsis

clean\_item = clean\_item.split('...')[0]

risk\_terms[clean\_item.lower()] = risk\_level

# Split summary into sentences

sentences = formatted\_summary.split('. ')

formatted\_sentences = []

for sentence in sentences:

sentence\_lower = sentence.lower()

risk\_level = None

# Check if sentence contains any risk terms

for term, level in risk\_terms.items():

if term in sentence\_lower:

risk\_level = level

break

# Format sentence based on risk level

if risk\_level:

formatted\_sentences.append(

f"<span class='summary-{risk\_level}'>{sentence}</span>"

)

else:

formatted\_sentences.append(sentence)

return '. '.join(formatted\_sentences)

def display\_risk\_meter(score: int, risks: Dict[str, List[str]]):

"""Display an enhanced visual risk meter with override for high-risk flags"""

st.markdown("### Overall Privacy Risk Score")

has\_high\_risk = any(r for r in risks['high'] if not r.startswith('No'))

if has\_high\_risk:

color = "red"

message = "High Risk"

context = "This policy contains high-risk practices that may significantly impact your privacy."

elif score >= 70:

color = "green"

message = "Low Risk"

context = "This policy appears to have good privacy practices."

elif score >= 40:

color = "orange"

message = "Medium Risk"

context = "This policy has some concerning elements that should be reviewed."

else:

color = "red"

message = "High Risk"

context = "This policy has significant privacy concerns that require attention."

# Visual elements

col1, col2 = st.columns([2, 3])

with col1:

st.progress(score / 100)

st.markdown(

f"<h2 style='color: {color}; text-align: center;'>{score}/100</h2>",

unsafe\_allow\_html=True

)

with col2:

st.markdown(f"<h3 style='color: {color};'>{message}</h3>", unsafe\_allow\_html=True)

st.write(context)

def display\_analysis(risks: Dict[str, List[str]], text: str):

"""Enhanced display with comprehensive risk analysis and recommendations"""

try:

# Calculate and display risk score

score = calculate\_risk\_score(risks)

display\_risk\_meter(score, risks)

# Display categorized risks

st.markdown("### Detailed Risk Analysis")

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

with col1:

st.markdown("<div class='risk-header-high'>🔴 High Risk Items</div>",

unsafe\_allow\_html=True)

high\_risks = [r for r in risks['high'] if not r.startswith('No')]

if high\_risks:

for item in high\_risks:

st.markdown(f"<div class='risk-box high-risk'>• {item}</div>",

unsafe\_allow\_html=True)

else:

st.markdown("<div class='risk-box'>✓ No high-risk items found</div>",

unsafe\_allow\_html=True)

with col2:

st.markdown("<div class='risk-header-medium'>🟡 Medium Risk Items</div>",

unsafe\_allow\_html=True)

medium\_risks = [r for r in risks['medium'] if not r.startswith('No')]

if medium\_risks:

for item in medium\_risks:

st.markdown(f"<div class='risk-box medium-risk'>• {item}</div>",

unsafe\_allow\_html=True)

else:

st.markdown("<div class='risk-box'>✓ No medium-risk items found</div>",

unsafe\_allow\_html=True)

with col3:

st.markdown("<div class='risk-header-low'>🟢 Low Risk Items</div>",

unsafe\_allow\_html=True)

low\_risks = [r for r in risks['low'] if not r.startswith('No')]

if low\_risks:

for item in low\_risks:

st.markdown(f"<div class='risk-box low-risk'>• {item}</div>",

unsafe\_allow\_html=True)

else:

st.markdown("<div class='risk-box'>✓ No low-risk items found</div>",

unsafe\_allow\_html=True)

# # Add recommendations

# st.markdown("### Recommendations")

# recommendations = generate\_recommendations(risks, score)

# for rec in recommendations:

# st.info(rec)

# Add recommendations inside an expander for alignment and compact view

with st.expander("📌 Recommendations"):

# Add a wrapper div to enforce consistent height for alignment

st.markdown('<div style="min-height: 150px;">', unsafe\_allow\_html=True)

recommendations = generate\_recommendations(risks, score)

if recommendations:

for rec in recommendations:

st.info(rec)

else:

st.success("✅ No additional recommendations found.")

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

# Add privacy impact summary

st.markdown("### Privacy Impact Summary")

impact\_summary = generate\_impact\_summary(risks, score)

st.write(impact\_summary)

except Exception as e:

st.error(f"Error displaying analysis: {str(e)}")

st.info("Falling back to simple display")

# Fallback display

for risk\_level, items in risks.items():

st.markdown(f"### {risk\_level.title()} Risk Items")

for item in items:

st.write(f"• {item}")

def generate\_impact\_summary(risks: Dict[str, List[str]], score: int) -> str:

"""Generate a comprehensive privacy impact summary"""

high\_count = len([r for r in risks['high'] if not r.startswith('No')])

medium\_count = len([r for r in risks['medium'] if not r.startswith('No')])

low\_count = len([r for r in risks['low'] if not r.startswith('No')])

summary = []

if score >= 70:

summary.append("🟢 This privacy policy demonstrates good privacy practices overall.")

elif score >= 40:

summary.append("🟡 This privacy policy has some areas that could be improved.")

else:

summary.append("🔴 This privacy policy raises significant privacy concerns.")

if high\_count > 0:

summary.append(f"Found {high\_count} high-risk practices that could significantly impact your privacy.")

if medium\_count > 0:

summary.append(f"Identified {medium\_count} medium-risk items that warrant attention.")

if low\_count > 0:

summary.append(f"Noted {low\_count} low-risk items that are generally acceptable but should be monitored.")

return "\n\n".join(summary)

# ==========================

# Section 5: Main App

# ==========================

def main():

st.title("Welcome to LegalLens! ⚖️")

st.subheader("Your AI-Powered Legal Assistant")

st.write("This app provides easy-to-understand summaries of "

"privacy policies and terms of service (ToS) agreements.")

# Sidebar with metrics and information

with st.sidebar:

st.subheader("Analytics")

# st.metric("Documents Analyzed", "100+")

# st.metric("Average Privacy Score", "75/100")

with st.expander("Learn About Your Privacy Rights"):

st.markdown("""

### Key Privacy Rights

- Right to Access Your Data

- Right to Delete Your Data

- Right to Opt-Out

- Right to Data Portability

### Privacy Best Practices

- Regularly review privacy settings

- Use strong, unique passwords

- Enable two-factor authentication

- Be cautious with third-party integrations

""")

with st.expander("About Risk Levels"):

st.markdown("""

🔴 \*\*High Risk\*\*: Practices that could significantly impact your privacy

- Data selling

- Extensive tracking

- Broad data sharing

🟡 \*\*Medium Risk\*\*: Practices that warrant attention

- Cookie usage

- Analytics

- Marketing

🟢 \*\*Low Risk\*\*: Generally acceptable practices

- Basic account info

- Essential services

- Security measures

""")

with st.expander("How Are Risks Categorized?"):

explain\_risk\_categorization()

# Main content

tab1, tab2 = st.tabs(["Single Analysis", "Compare Policies"])

with tab1:

input\_type = st.selectbox(

"To get started, select your preferred input method:",

["Company Name", "URL"],

index=None,

placeholder="Select input type",

)

if input\_type:

input\_text = st.text\_input(f"Enter {input\_type}:").strip()

if input\_text:

if len(input\_text) < MIN\_INPUT\_LENGTH:

st.error(f"Please enter at least {MIN\_INPUT\_LENGTH} characters")

return

try:

with st.spinner("Fetching policy..."):

if input\_type == "Company Name":

privacy\_policy\_url = get\_privacy\_policy\_url(os.getenv("API\_KEY"), input\_text)

else:

privacy\_policy\_url = input\_text

if privacy\_policy\_url:

st.write("✅ URL found:", privacy\_policy\_url)

text = scrape\_text(privacy\_policy\_url)

if text:

with st.spinner("Analyzing risks..."):

cache = LLMCacheTool()

summary, risks, from\_cache = get\_summary\_for\_tos(privacy\_policy\_url, text, cache)

if from\_cache:

st.info("✅ Retrieved analysis from cache")

else:

st.success("✅ Analysis complete")

# Display risk analysis

display\_analysis(risks, text)

# Display color-coded summary

with st.expander("📄 Full Summary"):

formatted\_summary = format\_summary\_with\_risks(summary, risks)

st.markdown(formatted\_summary, unsafe\_allow\_html=True)

with st.expander("📄 Full Policy Text"):

st.write(text)

except Exception as e:

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

with tab2:

col1, col2 = st.columns(2)

# Initialize variables

risks1 = None

risks2 = None

text1 = None

text2 = None

score1 = 0

score2 = 0

with col1:

company1 = st.text\_input("First Company/URL")

if company1:

try:

with st.spinner("Analyzing first policy..."):

url1 = get\_privacy\_policy\_url(os.getenv("API\_KEY"), company1) if not company1.startswith('http') else company1

text1 = scrape\_text(url1)

cache = LLMCacheTool()

summary1, risks1, from\_cache1 = get\_summary\_for\_tos(url1, text1, cache)

risks1 = analyze\_privacy\_policy(text1)

st.success("✅ First policy analyzed")

except Exception as e:

st.error(f"Error analyzing first policy: {str(e)}")

with col2:

company2 = st.text\_input("Second Company/URL")

if company2:

try:

with st.spinner("Analyzing second policy..."):

url2 = get\_privacy\_policy\_url(os.getenv("API\_KEY"), company2) if not company2.startswith('http') else company2

text2 = scrape\_text(url2)

cache = LLMCacheTool()

summary2, risks2, from\_cache2 = get\_summary\_for\_tos(url2, text2, cache)

risks2 = analyze\_privacy\_policy(text2)

st.success("✅ Second policy analyzed")

except Exception as e:

st.error(f"Error analyzing second policy: {str(e)}")

if company1 and company2 and risks1 and risks2:

st.subheader("Comparison Results")

comparison\_cols = st.columns(2)

with comparison\_cols[0]:

st.markdown(f"### 🔍 [{company1}]({url1})")

display\_analysis(risks1, text1)

with comparison\_cols[1]:

st.markdown(f"### 🔍 [{company2}]({url2})")

display\_analysis(risks2, text2)

# Key Differences

st.markdown("### Key Differences")

diff\_col1, diff\_col2, diff\_col3 = st.columns(3)

with diff\_col1:

diff\_high = len(risks1['high']) - len(risks2['high'])

st.metric("High Risk Items", f"{len(risks1['high'])} vs {len(risks2['high'])}", diff\_high, delta\_color="inverse")

with diff\_col2:

diff\_medium = len(risks1['medium']) - len(risks2['medium'])

st.metric("Medium Risk Items", f"{len(risks1['medium'])} vs {len(risks2['medium'])}", diff\_medium, delta\_color="inverse")

with diff\_col3:

diff\_low = len(risks1['low']) - len(risks2['low'])

st.metric("Low Risk Items", f"{len(risks1['low'])} vs {len(risks2['low'])}", diff\_low, delta\_color="inverse")

# Overall Conclusion

st.markdown("### Overall Comparison")

if score1 > score2:

st.success(f"🏆 {company1} has better privacy practices (Score: {score1} vs {score2})")

elif score2 > score1:

st.success(f"🏆 {company2} has better privacy practices (Score: {score2} vs {score1})")

else:

st.info("Both policies have similar privacy practices.")

if \_\_name\_\_ == "\_\_main\_\_":

try:

check\_requirements()

configure()

main()

except Exception as e:

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

st.info("Please refresh the page and try again")