LABORATORY REPORT

Application Development Lab (CS33002)

B.Tech Program in ECSc

Submitted By

Name:- SAHIL KUMAR

Roll No: 2230189

Kalinga Institute of Industrial Technology
(Deemed to be University)
Bhubaneswar, India

Spring 2024-2025

Table of Content

Experiment Number	5
Experiment Title	Web Scraper using LLMs
Date of Experiment	09|02|2025
Date of Submission	19|02|2025

1. **Objective:-**

- To create a web scraper application integrated with LLMs for processing scraped data.

2. **Procedure:- (Steps Followed)**

- Use Python libraries like BeautifulSoup and Requests to scrape web data.
- You can also use LlamaIndex for Web Scraping and Ollama for open ended LLMs.
- Integrate LLMs to process and summarize the scraped information.
- Develop a Flask backend for handling scraping tasks and queries.
- Create an HTML/CSS frontend to initiate scraping (like the web page to scrape) and display results.
- You can also take a topic and search the web for a web page and then scrape it.

3. **Code:-**

Index.html file:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
            <meta   name="viewport"   content="width=device-width,
initial-scale=1.0" />
    <title>Intelligent Web Scraper</title>
    <link rel="stylesheet" href="styles/styles.css" />
    <link
      rel="stylesheet"

href="https://fonts.googleapis.com/css2?family=Inter:wght@400;500
;600;700;800&display=swap"
    />
```

```html
  </head>
  <body>
    <div class="container">
      <div class="header">
                <h1><span  class="title-gradient">Intelligent  Web
Scraper</span></h1>
        <p class="subtitle">
               Extract  insights  from  any  webpage  with  AI-powered
analysis
        </p>
      </div>

      <div class="search-container">
        <div class="input-wrapper">
          <div class="input-group">
                <label  for="urlInput"  class="input-label">Website
URL</label>
            <input
              type="text"
              id="urlInput"
              placeholder="Enter any website URL to analyze..."
              autocomplete="off"
            />
          </div>
          <div class="input-group">
                <label  for="topicInput"  class="input-label">Topic
(Optional)</label>
            <input
              type="text"
              id="topicInput"
                 placeholder="Enter  a  specific  topic  to  focus  the
analysis"
              autocomplete="off"
            />
          </div>
          <div class="input-group">
                  <label  for="modelSelect"  class="input-label">AI
Model</label>
            <select id="modelSelect" class="model-select">
              <option value="deepseek">Deepseek (Local)</option>
              <option value="gemini">Gemini</option>
              <option value="groq">Groq</option>
            </select>
          </div>
          <button onclick="scrapeWebsite()" class="analyze-btn">
            <span class="btn-text">Analyze Website</span>
            <span class="btn-icon">→</span>
          </button>
        </div>
      </div>

      <div class="loading" id="loading">
        <div class="loader"></div>
        <p>Analyzing website content...</p>
      </div>

      <div class="error" id="error"></div>

      <div class="results" id="results">
        <div class="summary-container" id="summary"></div>
      </div>
    </div>
    <script src="styles/script.js"></script>
  </body>
</html>
```

Styles.css file:

```css
:root {
  --primary: #4f46e5;
  --primary-dark: #4338ca;
  --primary-light: #eef2ff;
  --background: #f9fafb;
  --text: #1f2937;
  --text-light: #6b7280;
  --error: #ef4444;
  --success: #10b981;
  --code-bg: #f3f4f6;
  --input-border: #e5e7eb;
  --card-shadow: 0 4px 6px -1px rgba(0, 0, 0, 0.1),
    0 2px 4px -1px rgba(0, 0, 0.06);
}
```

```css
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

body {
  font-family: "Inter", sans-serif;
  background: var(--background);
  color: var(--text);
  line-height: 1.6;
  min-height: 100vh;
  display: flex;
  align-items: center;
  justify-content: center;
  padding: 2rem;
  background: linear-gradient(135deg, #f9fafb 0%, #eef2ff 100%);
}

.container {
  max-width: 800px;
  width: 100%;
  margin: 0 auto;
  opacity: 0;
  transform: translateY(20px);
  animation: fadeIn 0.8s cubic-bezier(0.4, 0, 0.2, 1) forwards;
}

.header {
  text-align: center;
  margin-bottom: 3.5rem;
  padding: 0 1rem;
}

h1 {
  font-size: 2.75rem;
  font-weight: 800;
  margin-bottom: 0.75rem;
  letter-spacing: -0.025em;
  color: var(--text);
}
```

```css
.title-gradient {
  background-size: 200% auto;
  background-clip: text;
  -webkit-background-clip: text;
  -webkit-text-fill-color: transparent;
  animation: rgbText 8s linear infinite;
}

@keyframes rgbText {
  0% {
    background-image: linear-gradient(
      92deg,
      rgba(59, 130, 246, 0.9),
      rgba(16, 185, 129, 0.7),
      rgba(99, 102, 241, 0.9),
      rgba(59, 130, 246, 0.9)
    );
    background-position: 0% center;
  }
  50% {
    background-image: linear-gradient(
      92deg,
      rgba(59, 130, 246, 0.9),
      rgba(16, 185, 129, 0.7),
      rgba(99, 102, 241, 0.9),
      rgba(59, 130, 246, 0.9)
    );
    background-position: 100% center;
  }
  100% {
    background-image: linear-gradient(
      92deg,
      rgba(59, 130, 246, 0.9),
      rgba(16, 185, 129, 0.7),
      rgba(99, 102, 241, 0.9),
      rgba(59, 130, 246, 0.9)
    );
    background-position: 0% center;
  }
}

.subtitle {
  color: var(--text-light);
  font-size: 1.125rem;
  font-weight: 500;
}

.search-container {
  margin-bottom: 2.5rem;
  padding: 0 1rem;
}

.input-wrapper {
  background: white;
  border-radius: 16px;
  padding: 1rem;
  box-shadow: var(--card-shadow);
  display: flex;
  flex-direction: column;
  gap: 1rem;
  transition: all 0.3s cubic-bezier(0.4, 0, 0.2, 1);
}

.input-wrapper:focus-within {
  transform: translateY(-2px);
  box-shadow: 0 10px 15px -3px rgba(0, 0, 0, 0.1),
    0 4px 6px -2px rgba(0, 0, 0, 0.05);
}

.input-group {
  display: flex;
  flex-direction: column;
  gap: 0.5rem;
}

.input-label {
  font-size: 0.875rem;
  font-weight: 500;
  color: var(--text);
}

#urlInput,
#topicInput,
.model-select {
  width: 100%;
  border: 1px solid var(--input-border);
  padding: 0.75rem 1rem;
  font-size: 1rem;
  outline: none;
  color: var(--text);
  border-radius: 8px;
  transition: all 0.2s ease;
}

#urlInput:focus,
#topicInput:focus,
.model-select:focus {
  border-color: var(--primary);
  box-shadow: 0 0 0 3px var(--primary-light);
}

.model-select {
  background-color: white;
  cursor: pointer;
  appearance: none;
  background-image: url("data:image/svg+xml,%3Csvg xmlns='http://www.w3.org/2000/svg' fill='none' viewBox='0 0 24 24' stroke='%236b7280'%3E%3Cpath stroke-linecap='round' stroke-linejoin='round' stroke-width='2' d='M19 9l-7 7-7-7'%3E%3C/path%3E%3C/svg%3E");
  background-repeat: no-repeat;
  background-position: right 1rem center;
  background-size: 1.5em 1.5em;
  padding-right: 2.5rem;
}

.model-select:hover {
  border-color: var(--primary);
}

.analyze-btn {
  background: var(--primary);
  color: white;
  border: none;
  padding: 1rem 1.5rem;
  border-radius: 8px;
  font-weight: 600;
  font-size: 1rem;
  cursor: pointer;
```

```css
  display: flex;
  align-items: center;
  justify-content: center;
  gap: 0.5rem;
  transition: all 0.3s cubic-bezier(0.4, 0, 0.2, 1);
  margin-top: 0.5rem;
}

.analyze-btn:hover {
  background: var(--primary-dark);
  transform: translateY(-1px);
}

.btn-icon {
  transition: transform 0.3s cubic-bezier(0.4, 0, 0.2, 1);
}

.analyze-btn:hover .btn-icon {
  transform: translateX(4px);
}

.loading {
  display: none;
  align-items: center;
  justify-content: center;
  gap: 1rem;
  margin: 2rem 0;
}

.loader {
  width: 24px;
  height: 24px;
  border: 3px solid #ddd;
  border-top-color: var(--primary);
  border-radius: 50%;
  animation: spin 0.8s linear infinite;
}

.error {
  display: none;
  color: var(--error);
  background: rgba(239, 68, 68, 0.1);
  padding: 1rem 1.5rem;
  border-radius: 12px;
  margin: 1rem;
  text-align: center;
  animation: shake 0.4s cubic-bezier(0.36, 0, 0.66, -0.56);
  font-weight: 500;
}

.results {
  display: none;
  opacity: 0;
  transform: translateY(20px);
  padding: 0 1rem;
}

.results.visible {
  display: block;
  animation: fadeIn 0.6s cubic-bezier(0.4, 0, 0.2, 1) forwards;
}

.summary-container {
  background: white;
  padding: 2rem;
  border-radius: 16px;
  box-shadow: var(--card-shadow);
}

.markdown-content {
  line-height: 1.7;
  color: var(--text);
}

.markdown-content h1,
.markdown-content h2,
.markdown-content h3 {
  margin: 1.5rem 0 1rem;
  color: var(--text);
  font-weight: 700;
  line-height: 1.3;
}

.markdown-content h1 {
  font-size: 1.875rem;
}
```

```css
.markdown-content h2 {
  font-size: 1.5rem;
}

.markdown-content h3 {
  font-size: 1.25rem;
}

.markdown-content p {
  margin-bottom: 1.25rem;
}

.markdown-content ul {
  margin: 1.25rem 0;
  padding-left: 1.5rem;
}

.markdown-content li {
  margin-bottom: 0.75rem;
  position: relative;
}

.markdown-content li::before {
  content: "•";
  color: var(--primary);
  position: absolute;
  left: -1rem;
}

.markdown-content a {
  color: var(--primary);
  text-decoration: none;
  transition: all 0.2s ease;
  border-bottom: 1px solid transparent;
}

.markdown-content a:hover {
  border-bottom-color: var(--primary);
}

.markdown-content code {
  background: var(--code-bg);
  padding: 0.2rem 0.4rem;
  border-radius: 4px;
    font-family:  ui-monospace,  SFMono-Regular,  Menlo,  Monaco,
Consolas, monospace;
  font-size: 0.875em;
  color: var(--primary-dark);
}

.markdown-content pre {
  background: var(--code-bg);
  padding: 1.25rem;
  border-radius: 8px;
  overflow-x: auto;
  margin: 1.25rem 0;
}

.markdown-content pre code {
  background: none;
  padding: 0;
  color: var(--text);
}

.markdown-content strong {
  font-weight: 600;
  color: var(--text);
}

.markdown-content em {
  font-style: italic;
}

@keyframes fadeIn {
  from {
    opacity: 0;
    transform: translateY(20px);
  }
  to {
    opacity: 1;
    transform: translateY(0);
  }
}

@keyframes spin {
```

```css
  to {
    transform: rotate(360deg);
  }
}

@keyframes shake {
  0%,
  100% {
    transform: translateX(0);
  }
  25% {
    transform: translateX(-5px);
  }
  75% {
    transform: translateX(5px);
  }
}

@media (max-width: 640px) {
  body {
    padding: 1rem;
  }

  h1 {
    font-size: 2rem;
  }

  .input-wrapper {
    padding: 1rem;
  }

  .analyze-btn {
    width: 100%;
  }

  .summary-container {
    padding: 1.5rem;
  }
}
```

script.js file

```
document.addEventListener("DOMContentLoaded", () => {
  const inputs = ["urlInput", "topicInput"];

  inputs.forEach((inputId) => {
    const input = document.getElementById(inputId);
    input.addEventListener("focus", () => {
      input.parentElement.classList.add("focused");
    });
    input.addEventListener("blur", () => {
      input.parentElement.classList.remove("focused");
    });

    input.addEventListener("keypress", (e) => {
      if (e.key === "Enter") {
        scrapeWebsite();
      }
    });
  });
});

function parseMarkdown(text) {
                text        =        text.replace(/```([^`]+)```/g,
"<pre><code>$1</code></pre>");

  text = text.replace(/`([^`]+)`/g, "<code>$1</code>");

  text = text.replace(/^### (.*$)/gm, "<h3>$1</h3>");
  text = text.replace(/^## (.*$)/gm, "<h2>$1</h2>");
  text = text.replace(/^# (.*$)/gm, "<h1>$1</h1>");

  text = text.replace(/\*\*(.*?)\*\*/g, "<strong>$1</strong>");

  text = text.replace(/\*(.*?)\*/g, "<em>$1</em>");

  text = text.replace(
    /\[([^\]]+)\]\(([^)]+)\)/g,
    '<a href="$2" target="_blank">$1</a>'
  );

  text = text.replace(/^\s*-\s(.+)/gm, "<li>$1</li>");
```

```
  text = text.replace(/(<li>.*<\/li>)/s, "<ul>$1</ul>");

  text = text.replace(/\n\n/g, "</p><p>");
  text = "<p>" + text + "</p>";

  return text;
}

async function scrapeWebsite() {
  const urlInput = document.getElementById("urlInput");
  const topicInput = document.getElementById("topicInput");
  const modelSelect = document.getElementById("modelSelect");
  const loading = document.getElementById("loading");
  const error = document.getElementById("error");
  const results = document.getElementById("results");
  const summary = document.getElementById("summary");

  error.style.display = "none";
  results.classList.remove("visible");
  summary.innerHTML = "";

  const url = urlInput.value.trim();

  const topic = topicInput.value.trim();
  const model = modelSelect.value;

  try {
    loading.style.display = "flex";

    const response = await fetch("/scrape", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({ url, topic, model }),
    });

    const data = await response.json();

    if (response.ok) {
      let headerText = topic
        ? `Analysis of "${topic}" using ${model}`
        : `Website Analysis using ${model}`;
      if (data.fallback_url) {
          headerText = `Analysis of "${topic}" from Google Search
Result using ${model}`;
      }

      summary.innerHTML = `
                        <h2 style="margin-bottom: 1.5rem; color:
var(--text);         font-size:        1.5rem;        font-weight:
700;">${headerText}</h2>
                <div class="markdown-content">${parseMarkdown(
                  data.summary
                )}</div>
            `;

      results.classList.add("visible");
    } else {
      showError(data.error || "Failed to analyze the website");
    }
  } catch (err) {
    showError("Error processing request. Please try again.");
  } finally {
    loading.style.display = "none";
  }
}

function showError(message) {
  const error = document.getElementById("error");
  error.textContent = message;
  error.style.display = "block";
}
```

app.py:-

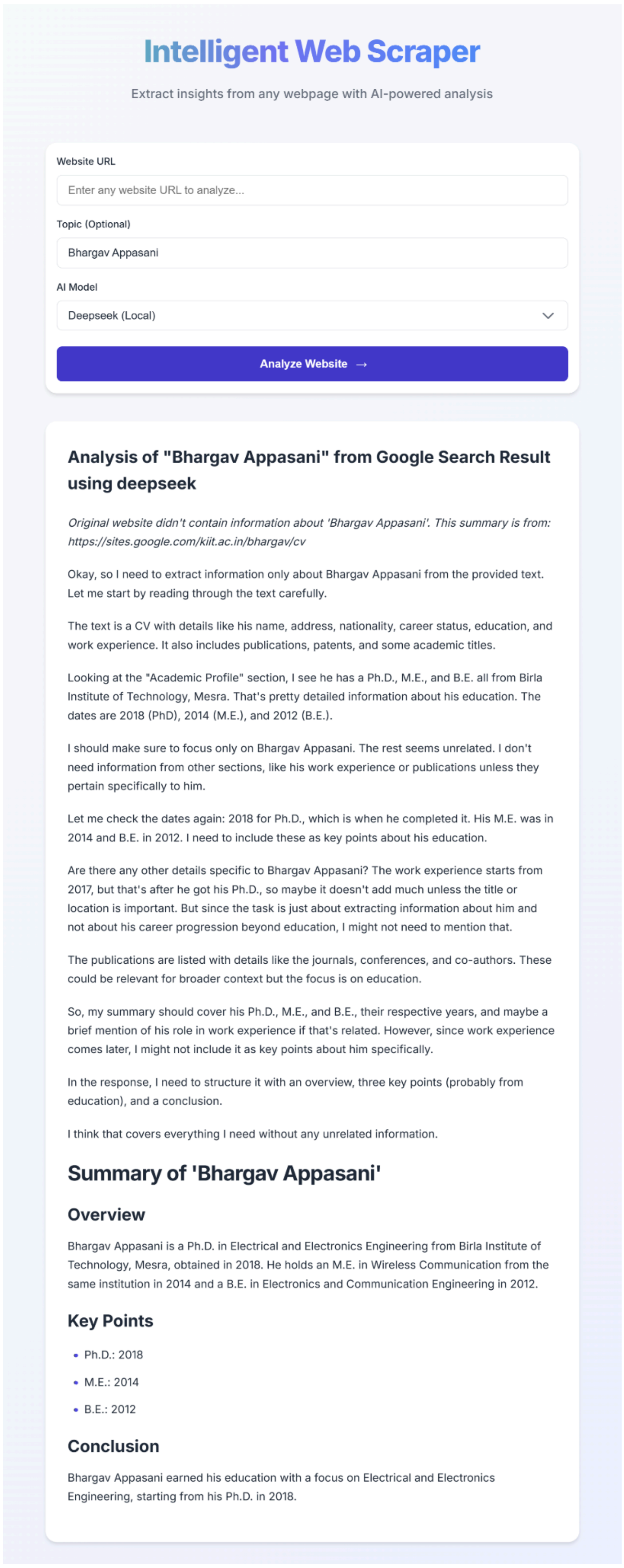
```
from flask import Flask, render_template, request, jsonify
from bs4 import BeautifulSoup
import requests
import validators
from dotenv import load_dotenv
import os
import google.generativeai as genai
from groq import Groq
import re
from urllib.parse import quote_plus

load_dotenv('.env.local')

app = Flask(__name__, static_folder='styles',
template_folder='.')

def init_gemini():
    gemini_api_key = os.getenv('GEMINI_API_KEY')
    if not gemini_api_key:
        raise ValueError("GEMINI_API_KEY not found in environment
variables")
    genai.configure(api_key=gemini_api_key)
    return genai.GenerativeModel('gemini-pro')

def init_groq():
    groq_api_key = os.getenv('GROQ_API_KEY')
    if not groq_api_key:
        raise ValueError("GROQ_API_KEY not found in environment
variables")
    return Groq(api_key=groq_api_key)

OLLAMA_API_URL = "http://localhost:11434/api/generate"

def get_ollama_response(prompt):
    payload = {
        "model": "deepseek-r1:1.5b",
        "prompt": prompt,
        "stream": False
    }
    try:
        response = requests.post(OLLAMA_API_URL, json=payload)
        response.raise_for_status()
        return response.json()["response"]
    except requests.exceptions.ConnectionError:
        raise Exception("Could not connect to Ollama. Make sure
it's running (ollama run deepseek-r1:1.5b)")
    except Exception as e:
        raise Exception(f"Ollama API error: {str(e)}")
```

```
try:
    gemini_model = init_gemini()
    groq_client = init_groq()
except Exception as e:
    print(f"Error initializing AI models: {str(e)}")

def scrape_webpage(url):
    try:
        headers = {
            'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64;
x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124
Safari/537.36'
        }
        response = requests.get(url, headers=headers, timeout=10)
        response.raise_for_status()
        soup = BeautifulSoup(response.text, 'html.parser')

        content = []
            for tag in soup.find_all(['p', 'h1', 'h2', 'h3',
'article']):
            if tag.text.strip():
                content.append(tag.text.strip())

        return ' '.join(content)
    except Exception as e:
        return f"Error scraping webpage: {str(e)}"

def duckduckgo_search(query):
    try:
                                                  search_url =
f"https://html.duckduckgo.com/html/?q={quote_plus(query)}"
        headers = {
            'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64;
x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124
Safari/537.36',
        }
            response = requests.get(search_url, headers=headers,
timeout=10)
        response.raise_for_status()

                  print(f"DuckDuckGo search status code:
{response.status_code}")
```

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

                                                                    results       =
soup.select('.results_links_deep:not(.results-sponsored)')

        valid_links = []
        for result in results:
            link = result.select_one('.result__title .result__a')
            snippet = result.select_one('.result__snippet')

            if link and snippet:
                href = link.get('href', '')
                title = link.get_text()
                snippet_text = snippet.get_text()

                            if  any(ad_indicator  in  href.lower()  for
ad_indicator in [
                      'ad_provider', 'sponsored', 'advertisement',
'pdffiller',
                    'click_metadata', 'utm_source=bing'
                ]):
                    continue

                if href.startswith('http'):
                    url = href
                elif href.startswith('/'):
                    try:
                                                                            url   =
requests.utils.unquote(href.split('?uddg=')[1].split('&')[0])
                    except:
                        continue
                else:
                    continue

                valid_links.append({
                    'url': url,
                    'title': title,
                    'snippet': snippet_text
                })
                print(f"Found DuckDuckGo result: {url}")

                if len(valid_links) >= 5:
                    break
        if valid_links:
            news_domains = ['news', 'times', 'bbc', 'cnn', 'reuters', 'theweek', 'firstpost', 'ndtv', 'msn']
            for link in valid_links:
                if any(domain in link['url'].lower() for domain in news_domains):
                    return link['url']

            return valid_links[0]['url']

            print("No valid links found in DuckDuckGo search results.")
        return None

    except Exception as e:
        print(f"Error during DuckDuckGo search: {str(e)}")
        return None

def create_prompt(text, topic=None):
    if topic:
        prompt = f"""TASK: Extract and summarize ONLY information about '{topic}' from the provided text.

        TEXT: {text[:3000]}

        INSTRUCTIONS:
        1. Focus exclusively on '{topic}' - ignore all unrelated content
        2. If '{topic}' is mentioned, provide:
           - A brief overview of how '{topic}' is discussed
           - 3-5 key points specifically about '{topic}'
           - A short conclusion about '{topic}'
        3. If '{topic}' is not mentioned at all, simply respond: "The topic '{topic}' was not found in the content."
        4. Do not include any information unrelated to '{topic}'
        5. Do not mention or summarize other topics or the general content

        FORMAT YOUR RESPONSE AS FOLLOWS (only if the topic is found):
        # Summary of '{topic}'
        ## Overview
        [Brief overview of how '{topic}' appears in the text]

        ## Key Points
        - [Point 1 about '{topic}']
        - [Point 2 about '{topic}']
        - [Point 3 about '{topic}']

        ## Conclusion
        [Brief conclusion specifically about '{topic}']"""
    else:
         prompt = f"""TASK: Provide a comprehensive summary of the
entire text.

        TEXT: {text[:3000]}

        INSTRUCTIONS:
        1. Identify the main topic or themes
        2. Extract the most important information
        3. Organize the summary in a clear structure
        4. Focus on substance over style

        FORMAT YOUR RESPONSE AS FOLLOWS:
        # Summary

        ## Main Topic
        [Brief description of what the text is primarily about]

        ## Key Points
        - [Most important point 1]
        - [Most important point 2]
        - [Most important point 3]
        - [Most important point 4]
        - [Most important point 5]

        ## Conclusion
        [Overall takeaway or significance of the content]"""

    return prompt

def check_if_topic_not_found(result, topic):
    if not topic:
        return False
        if re.search(rf"(?:topic|'{topic}'|the topic '{topic}')\s+(?:was|is)\s+not\s+found", result, re.IGNORECASE):
        return True

        if "therefore, i should respond accordingly without mentioning other topics." in result.lower():
        return True

        if re.search(rf"no\s+information\s+(?:about|on|regarding)\s+(?:the topic\s+)?['\"]?{topic}['\"]?", result, re.IGNORECASE):
        return True

    if not (re.search(r"## Overview", result, re.IGNORECASE) and re.search(r"## Key Points", result, re.IGNORECASE)):
        if re.search(rf"(?:couldn't|could not|didn't|did not|no|none)\s+find", result, re.IGNORECASE):
            return True

    return False

def process_with_llm(text, topic=None, model="deepseek"):
    prompt = create_prompt(text, topic)

    try:
        if model == "deepseek":
            result = get_ollama_response(prompt)

        elif model == "gemini":
            response = gemini_model.generate_content(prompt)
            result = response.text

        elif model == "groq":
            completion = groq_client.chat.completions.create(
                messages=[
                    {"role": "system", "content": "You are a helpful assistant that analyzes web content and provides well-structured summaries."},
                    {"role": "user", "content": prompt}
                ],
                model="llama3-8b-8192",
            )
```

```python
            result = completion.choices[0].message.content

        else:
            raise ValueError(f"Unsupported model: {model}")

        return result

    except Exception as e:
        raise Exception(f"Error processing with {model}:
{str(e)}")

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/scrape', methods=['POST'])
def scrape():
    data = request.get_json()
    url = data.get('url')
    topic = data.get('topic')
    model = data.get('model', 'deepseek')

    if not url:
        if not topic:
            return jsonify({'error': 'Either a URL or a topic
must be provided'}), 400

        print(f"No URL provided, searching DuckDuckGo for:
{topic}")
        url = duckduckgo_search(topic)

        if not url:
            return jsonify({'error': 'No relevant webpage found
for the topic'}), 404

        print(f"Using found URL: {url}")


    if not validators.url(url):
        return jsonify({'error': 'Invalid URL'}), 400

    scraped_content = scrape_webpage(url)
    try:
        print(f"Processing URL: {url} for topic: {topic} using
model: {model}")
        summary = process_with_llm(scraped_content, topic, model)
        print(f"Initial summary result: {summary[:200]}...")

        if topic and check_if_topic_not_found(summary, topic):
            print(f"Topic '{topic}' not found, attempting
DuckDuckGo search...")
            search_query = f"{topic} information"
            new_url = duckduckgo_search(search_query)
            print(f"DuckDuckGo search returned URL: {new_url}")

            if new_url:
                print(f"Scraping new URL: {new_url}")
                new_content = scrape_webpage(new_url)
                new_summary = process_with_llm(new_content,
topic, model)
                print(f"New summary result:
{new_summary[:200]}...")

                if not check_if_topic_not_found(new_summary,
topic):
                    print(f"Topic found in new URL. Returning
combined summary.")
                    search_info = f"*Original website didn't
contain information about '{topic}'. This summary is from:
{new_url}*\n\n"
                    return jsonify({
                        'summary': search_info + new_summary,
                        'fallback_url': new_url
                    })
                else:
                    print(f"Topic not found in new URL either.
Returning original summary.")

        return jsonify({
            'summary': summary
        })
    except ValueError as e:
        print(f"ValueError: {str(e)}")
        return jsonify({'error': str(e)}), 400
    except Exception as e:
        print(f"Exception: {str(e)}")
            return jsonify({'error': f'Error processing with LLM: 
{str(e)}'}), 500

if __name__ == '__main__':
    app.run(debug=True)
```

4. Results/Output:- Entire Screen Shot including Date & Time


Intelligent Web Scraper
Extract insights from any webpage with AI-powered analysis
Website URL
Enter any website URL to analyze...
Analyze Website →

Topic (Optional): Bhargav Appasani

AI Model: Deepseek (Local)

Analysis of "Bhargav Appasani" from Google Search Result using deepseek

Original website didn't contain information about 'Bhargav Appasani'. This summary is from: https://sites.google.com/kiit.ac.in/bhargav/cv

Okay, so I need to extract information only about Bhargav Appasani from the provided text. Let me start by reading through the text carefully.

The text is a CV with details like his name, address, nationality, career status, education, and work experience. It also includes publications, patents, and some academic titles.

Looking at the "Academic Profile" section, I see he has a Ph.D., M.E., and B.E. all from Birla Institute of Technology, Mesra. That's pretty detailed information about his education. The dates are 2018 (PhD), 2014 (M.E.), and 2012 (B.E.).

I should make sure to focus only on Bhargav Appasani. The rest seems unrelated. I don't need information from other sections, like his work experience or publications unless they pertain specifically to him.

Let me check the dates again: 2018 for Ph.D., which is when he completed it. His M.E. was in 2014 and B.E. in 2012. I need to include these as key points about his education.

Are there any other details specific to Bhargav Appasani? The work experience starts from 2017, but that's after he got his Ph.D., so maybe it doesn't add much unless the title or location is important. But since the task is just about extracting information about him and not about his career progression beyond education, I might not need to mention that.

The publications are listed with details like the journals, conferences, and co-authors. These could be relevant for broader context but the focus is on education.

So, my summary should cover his Ph.D., M.E., and B.E., their respective years, and maybe a brief mention of his role in work experience if that's related. However, since work experience comes later, I might not include it as key points about him specifically.

In the response, I need to structure it with an overview, three key points (probably from education), and a conclusion.

I think that covers everything I need without any unrelated information.

Summary of 'Bhargav Appasani'

Overview

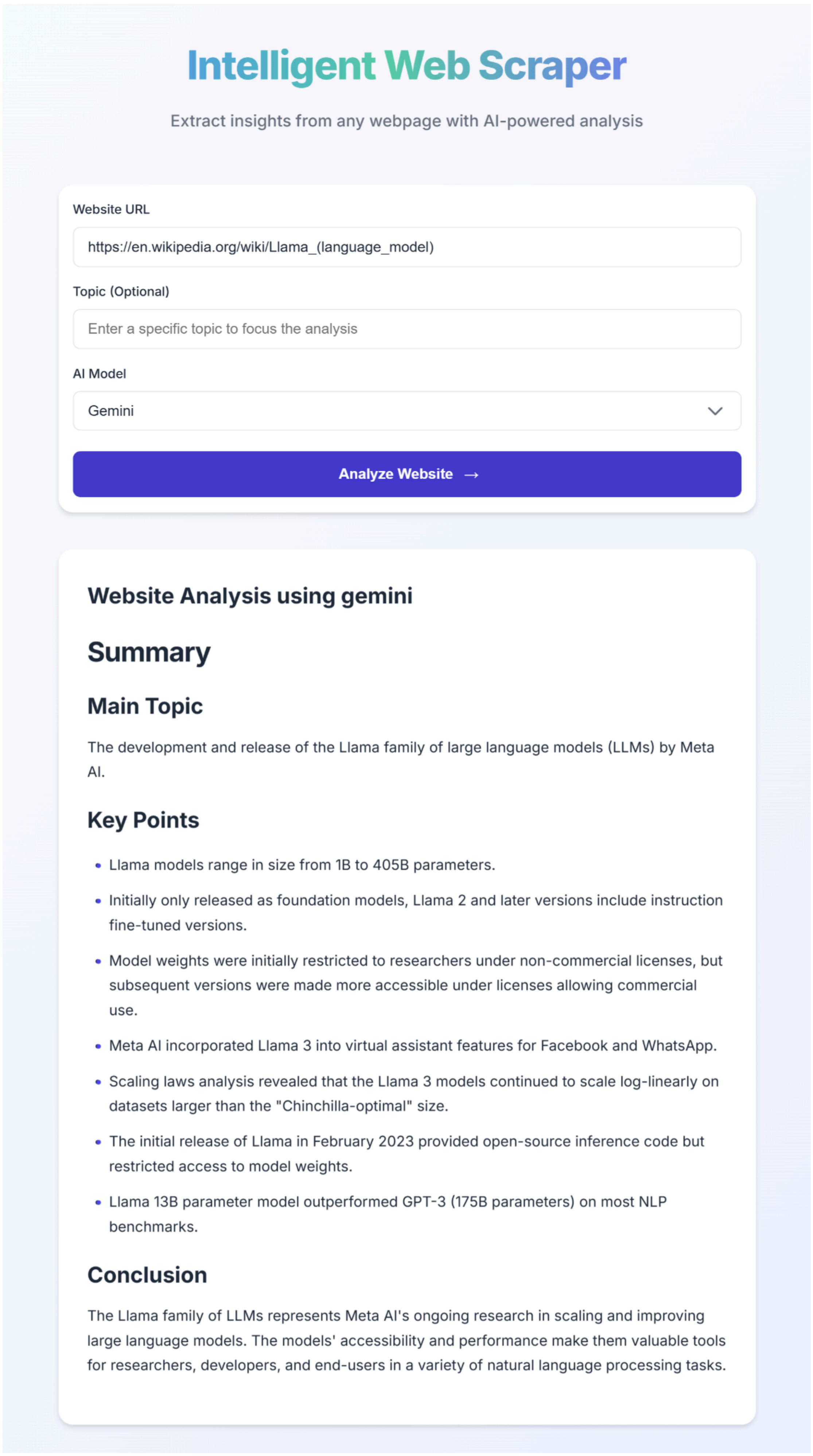
Bhargav Appasani is a Ph.D. in Electrical and Electronics Engineering from Birla Institute of Technology, Mesra, obtained in 2018. He holds an M.E. in Wireless Communication from the same institution in 2014 and a B.E. in Electronics and Communication Engineering in 2012.

Key Points

- Ph.D.: 2018
- M.E.: 2014
- B.E.: 2012

Conclusion

Bhargav Appasani earned his education with a focus on Electrical and Electronics Engineering, starting from his Ph.D. in 2018.



Intelligent Web Scraper

Extract insights from any webpage with AI-powered analysis

Website URL

https://en.wikipedia.org/wiki/Llama_(language_model)

Topic (Optional)

Enter a specific topic to focus the analysis

AI Model

Gemini

Website Analysis using gemini

Summary

Main Topic

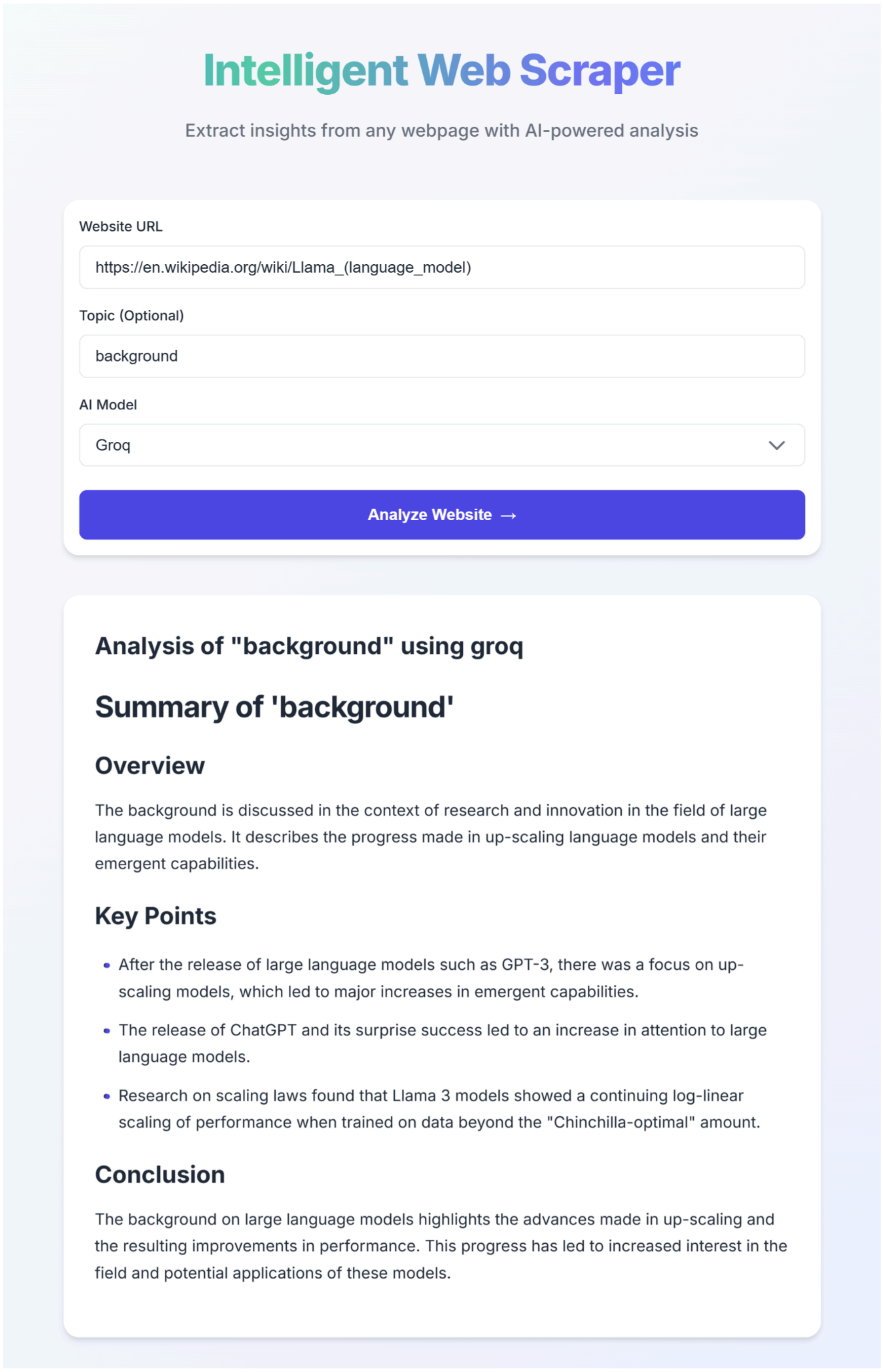
The development and release of the Llama family of large language models (LLMs) by Meta AI.

Key Points

- Llama models range in size from 1B to 405B parameters.
- Initially only released as foundation models, Llama 2 and later versions include instruction fine-tuned versions.
- Model weights were initially restricted to researchers under non-commercial licenses, but subsequent versions were made more accessible under licenses allowing commercial use.
- Meta AI incorporated Llama 3 into virtual assistant features for Facebook and WhatsApp.
- Scaling laws analysis revealed that the Llama 3 models continued to scale log-linearly on datasets larger than the "Chinchilla-optimal" size.
- The initial release of Llama in February 2023 provided open-source inference code but restricted access to model weights.
- Llama 13B parameter model outperformed GPT-3 (175B parameters) on most NLP benchmarks.

Conclusion

The Llama family of LLMs represents Meta AI's ongoing research in scaling and improving large language models. The models' accessibility and performance make them valuable tools for researchers, developers, and end-users in a variety of natural language processing tasks.



Intelligent Web Scraper

Extract insights from any webpage with AI-powered analysis

Website URL

https://en.wikipedia.org/wiki/Llama_(language_model)

Topic (Optional)

background

AI Model

Groq

Analysis of "background" using groq

Summary of 'background'

Overview

The background is discussed in the context of research and innovation in the field of large language models. It describes the progress made in up-scaling language models and their emergent capabilities.

Key Points

- After the release of large language models such as GPT-3, there was a focus on up-scaling models, which led to major increases in emergent capabilities.
- The release of ChatGPT and its surprise success led to an increase in attention to large language models.
- Research on scaling laws found that Llama 3 models showed a continuing log-linear scaling of performance when trained on data beyond the "Chinchilla-optimal" amount.

Conclusion

The background on large language models highlights the advances made in up-scaling and the resulting improvements in performance. This progress has led to increased interest in the field and potential applications of these models.

5. Remarks:-

Git - Github Repo

Signature of the Student	Signature of the Lab Coordinator
____________________	____________________
(Name of the Student)	(Name of the Coordinator)