ISSN: 3068-014X

Research Article

Archivos de Ciencia e Investigación

Automated Admission Booking via WhatsApp Using a Retrieval-Augmented Generation System

Neha Gupta and Bhawna Singla*

School of Computer Science and Engineering, Geeta University, India

***Corresponding Author**
Bhawna Singla, School of Computer Science and Engineering, Geeta University, India.

Submitted: 2026, May 10; **Accepted:** 2026, Jun 19; **Published:** 2026, Jun 30

Citation: Gupta, N., Singla, B. (2026). Automated Admission Booking via WhatsApp Using a Retrieval-Augmented Generation System. *Arch Cienc Investig, 2*(1), 01-08.

Abstract

In recent years, the integration of Natural Language Processing (NLP) with web scraping techniques has enabled intelligent automated systems that can improve user engagement and operational efficiency in educational institutions. This paper presents a novel approach to automate the admission inquiry and booking process for universities by extracting relevant course and fee information from official documents and dynamically scraping contact details from the university website. The system leverages transformer-based models for question answering (QA) on admission brochures and utilizes WhatsApp web URL schemes to facilitate direct communication between prospective students and admission offices. This approach significantly reduces manual intervention, enhances user experience, and streamlines the admission workflow.

Keywords: Admission Automation, Whatsapp Integration, Retrieval-Augmented Generation, NLP, Question Answering, Web Scraping, OCR, Brochure Analysis, Transformer Models, Semantic Search, University Admissions, Contact Extraction, Chatbot, Educational Technology, Conversational AI, Document Parsing, PDF Processing

1. Introduction

The admission process for educational institutions often involves multiple touchpoints, including inquiry handling, information dissemination, and seat booking. Traditionally, these processes are manually managed through phone calls or emails, which can be time-consuming and error-prone. With the rise of AI and automation, it is now possible to create intelligent systems that answer admission-related questions and assist users in booking admissions via popular communication platforms such as WhatsApp.

2. Related Work

Prior research has demonstrated the use of NLP for educational chatbots, web scraping for data aggregation, and WhatsApp automation for customer engagement [1-3]. However, an integrated system combining these components for university admissions remains under-explored. This paper addresses this gap by providing an end-to-end solution incorporating brochure analysis, contact scraping, and booking facilitation.

3. Methodology

The proposed system integrates multiple advanced technologies to automate the process of answering admission-related queries and facilitate booking through WhatsApp. The methodology encompasses four major components: data extraction and processing, a question answering system, web scraping for contact extraction, and WhatsApp booking integration. Each of these components plays a critical role in delivering an efficient, user-friendly admission assistance platform.

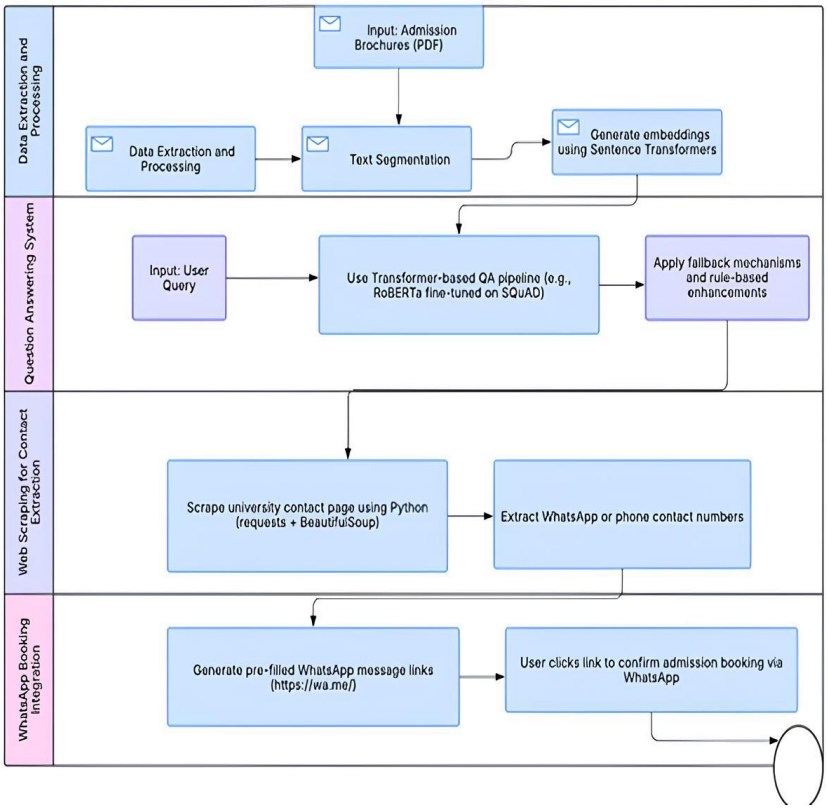

Figure 1: University admission assistance system

Figure 1 outlines a comprehensive system for assisting with university admissions. It includes processes for data extraction and processing from admission brochures, a question-answering system for user queries, web scraping for contact extraction, and WhatsApp booking integration. The key components involve generating embeddings, using a transformer-based QA pipeline, extracting contact numbers, and creating pre-filled WhatsApp message links.

3.1. Data Extraction and Processing

Admission brochures issued by educational institutions typically contain rich information regarding courses offered, fee structures, admission criteria, eligibility, and other institutional guidelines. These brochures are generally distributed in PDF format, which presents both opportunities and challenges for data extraction.

a. Challenges in PDF Processing

PDFs are designed primarily for visual presentation rather than structured data extraction, which complicates automatic text retrieval. The brochures often include multi-column layouts, tables, images, headers, footers, and other formatting complexities. Text may be embedded as images or scattered unevenly, making direct parsing difficult. Additionally, variations in brochure design and formatting across institutions necessitate a flexible yet robust extraction methodology.

b. Optical Character Recognition (OCR)

To overcome the challenges posed by scanned PDFs or PDFs containing images, Optical Character Recognition (OCR) is employed. OCR tools like Tesseract analyse images in the PDF pages to convert them into machine-readable text. This step is crucial when the brochure content is not available as selectable text. The OCR process is often complemented with pre-processing techniques such as binarization, noise removal, and skew correction to improve text recognition accuracy.

c. Natural Language Processing (NLP) for Structure Extraction

Once raw text is obtained either through direct parsing of text-based PDFs or via OCR, NLP techniques are applied to organize and structure the data. This involves several sub-tasks:

- Text Cleaning: Removing extraneous characters, multiple newlines, and irrelevant whitespace that can clutter the extracted text.
- Segmentation: Dividing the entire document into meaningful sections or chunks. For instance, the brochure might be split into sections like “Undergraduate Courses,” “Postgraduate Courses,” “Fee Structure,” and “Admission Process.” Segmentation helps in creating a semantic index for easier retrieval.
- Named Entity Recognition (NER): Identifying entities such as course names, fees, eligibility criteria, and dates to tag relevant information accurately.
- Table Extraction: For fee details often presented in tabular form, specialized table extraction tools (like Camelot or Tabula) parse these tables into structured data formats.

d. Embedding Models for Semantic Search

To enable an interactive question answering system, it is essential to retrieve the most relevant information chunks in response to user queries. This requires moving beyond simple keyword matching to semantic understanding. Embedding models like Sentence Transformers are employed to convert text chunks into dense vector representations capturing their semantic meaning. Sentence Transformers, based on architectures such as BERT or RoBERTa, generate embeddings that position semantically similar sentences or paragraphs close to each other in vector space.

During a query, the system converts the user's question into an embedding vector and computes similarity scores (e.g., cosine similarity) with stored brochure chunk embeddings. This retrieves the top relevant chunks providing the contextual foundation for answering queries.

e. Advantages of This Approach

- Flexibility: Works across various brochure formats and styles.
- Scalability: Can process large documents and maintain up-to-date indexed data.
- Improved Accuracy: Semantic search outperforms simple keyword search, leading to more relevant information retrieval.

3.2. Question Answering System

After retrieving the relevant context from the brochure, the system's next step is to provide concise and precise answers to user questions. This is accomplished through a transformer-based Question Answering (QA) pipeline.

3.2.1. Transformer Architectures for QA

Transformers, notably BERT (Bidirectional Encoder Representations from Transformers) and its variants like RoBERTa, have revolutionized NLP by enabling models to understand the context of words bidirectionally. Such models fine-tuned on question answering datasets such as SQuAD (Stanford Question Answering Dataset) demonstrate high accuracy in extracting exact answer spans from given passages. For this system, a model like RoBERTa fine-tuned on SQuAD is used. When a user submits a query, the model takes the top relevant brochure chunk as context and extracts the answer span, effectively pinpointing the precise information required.

3.2.2. Handling Challenges

- Ambiguous Queries: Users may ask questions in multiple ways or use synonyms. The semantic retrieval combined with transformer QA ensures robustness.
- Incomplete or Noisy Context: Sometimes, extracted text chunks may contain formatting artifacts or partial information. To mitigate this:
- Fallback Mechanisms: If the QA model returns low-confidence or no meaningful answers, rule-based or template-based responses provide fallback answers.
- Rule-based Enhancements: For frequently asked questions such as admission procedures or fee schedules, predefined responses ensure consistent and accurate answers.

a. Example Workflow

1. User asks: "What is the admission process for postgraduate courses?"
2. The system embeds this query and retrieves relevant text chunks tagged under "Admission Process" and "Postgraduate Courses."
3. The QA model extracts the answer span such as "Admissions are based on the GUTS score followed by counselling."
4. If no answer is confidently extracted, a rule-based default such as "Please refer to the official brochure or contact us directly" is returned.

b. Benefits

- Provides human-like precise answers without overwhelming users with entire brochure text.
- Handles complex queries by leveraging transformer understanding.
- Improves with additional fine-tuning and incorporation of user feedback.

3.3. Web Scraping for Contact Extraction

Accurate and up-to-date contact information is essential for facilitating communication between prospective students and the university. As universities may update their contact details or add new communication channels like WhatsApp, manual updates to the system can become cumbersome.

3.3.1. Dynamic Web Scraping

To automate contact detail updates, the system incorporates a dynamic web scraping module targeting the university's official contact or helpdesk web pages.

- Libraries Used: The Python requests library fetches web page content, while Beautiful Soup parses the HTML to locate phone numbers, emails, or WhatsApp contact details.
- Pattern Recognition: Regular expressions (regex) identify phone number patterns, typically country code +91 for India, standard mobile number lengths, or WhatsApp number formatting.
- Contextual Filtering: The scraper can be enhanced to look for keywords such as "WhatsApp," "Contact," or "Admissions Helpline" near extracted numbers to prioritize relevant contacts.

a. Handling Website Variations

Since website layouts vary widely and might change over time, the scraper is designed to be:

- Modular: Separate scraping functions for different web page structures.
- Robust: Error handling for page fetch failures or HTML structure changes.
- Configurable: Allowing adjustments to target URLs or regex patterns without rewriting code.
- Scheduling and Updates
- The scraper can run on a schedule (e.g., daily or weekly) to

refresh contact details automatically.
- Changes detected can trigger alerts or updates to the chatbot backend ensuring the booking links always use current phone numbers.
- Ethical Considerations
- Respect the website's robots.txt and terms of service.
- Avoid overloading the website with frequent requests.

3.4. WhatsApp Booking Integration

A key innovation in this system is leveraging WhatsApp, a widely adopted messaging platform, to enable direct, frictionless admission booking.

Why WhatsApp?

- Ubiquity: WhatsApp has over 2 billion users globally, making it a familiar and accessible communication tool for students.
- Real-Time Interaction: Enables instant messaging with admission officers, facilitating rapid clarifications and confirmations.
- No Additional App Needed: Most users already have WhatsApp installed, eliminating onboarding hurdles.

Integration Mechanism

Rather than implementing complex WhatsApp Business API integrations, the system utilizes the WhatsApp Web URL scheme for message pre-filling.

- The scheme URL format is: php-template

https://wa.me/<phone_number>?text=<url_encoded_message>

- Here, <phone_number> is the university's WhatsApp contact in international format without special characters, and <url_encoded_message> is the admission booking message, URL-encoded to handle spaces and special characters.

Workflow Example

1. After the user confirms interest in a course (e.g., "I want to book M.Tech admission"),
2. The system dynamically generates a booking message such as:
3. css
4. Hello, I would like to book admission for the course: M.Tech Computer Science Engineering. Please assist.
5. This message is URL-encoded and appended to the WhatsApp Web URL.
6. The user clicks the generated link, opening WhatsApp Web or the WhatsApp app with the message pre-filled.
7. The user sends the message directly to the university's admissions contact.

Advantages

- Ease of Implementation: No need for complicated API approvals or message automation.
- User Control: The user reviews and sends the message, ensuring consent and reducing spam.
- Real-Time Response: Enables human admission officers to respond instantly.

Potential Enhancements

- For full automation, integration with the WhatsApp Business API could be explored, enabling automated message handling and confirmations.
- Incorporating analytics to track booking link clicks and conversion rates.
- Multilingual message generation based on user preferences.

The methodology presented combines advanced document processing, semantic search, transformer-based question answering, dynamic web scraping, and communication platform integration to build a robust, scalable, and user-friendly admission assistance system. This pipeline not only improves information accessibility but also facilitates seamless admission booking, reducing operational burden on university staff and enhancing prospective student experience.

4. Implementation

The admission assistance system is designed as a modular pipeline that integrates several modern NLP, web scraping, and web integration techniques. This modular design allows flexibility, ease of maintenance, and extensibility. The core components include PDF brochure parsing, semantic embedding and retrieval, a transformer-based question answering (QA) pipeline, contact information scraping, booking link generation, and a user interface for interaction. Below, each module is described in detail along with sample code to illustrate the implementation.

4.1. PDF Brochure Parsing

The initial step is to extract text content from the university's admission brochure, often provided in PDF format. Since PDFs vary in structure and can include scanned images, a combination of PDF parsing and Optical Character Recognition (OCR) is used.

4.1.1. Tools and Libraries

- PyMuPDF (fitz): Efficient PDF text extraction.
- PyPDF2: Alternative PDF parsing library.
- Tesseract OCR: For extracting text from scanned images inside PDFs.
- pdf2image: To convert PDF pages to images for OCR.

4.1.2. Workflow

1. Attempt Direct Text Extraction: For PDFs with embedded text.
2. Fallback to OCR: If direct extraction fails or text is sparse.
3. Preprocessing: Clean text, remove headers, footers, and segment the content.

4.1.3. Example Code

python

```
import fitz  # PyMuPDF
from pdf2image import convert_from_path
import pytesseract
import os

def extract_text_from_pdf(pdf_path):
    # Step 1: Try direct extraction using PyMuPDF
    doc = fitz.open(pdf_path)
    full_text = ""
```

```
    for page_num in range(len(doc)):
        page = doc.load_page(page_num)
        text = page.get_text()
        if text.strip():
            full_text += text + "\n"

    # If text extraction is poor, fallback to OCR
    if len(full_text.strip()) < 100:  # arbitrary threshold
        full_text = ""
        images = convert_from_path(pdf_path)
        for image in images:
            ocr_text = pytesseract.image_to_string(image)
            full_text += ocr_text + "\n"
    return full_text

# Usage
pdf_file = "GU-Brochure-25-26.pdf"
raw_text = extract_text_from_pdf(pdf_file)
print(raw_text[:500])  # print first 500 characters for sanity check
```

4.1.4. Text Segmentation

After extraction, the entire text is often too long for direct querying. It is segmented into manageable chunks—paragraphs, sections, or fixed-length text blocks.

```
python
import nltk
nltk.download('punkt')
from nltk.tokenize import sent_tokenize

def segment_text(text, max_chunk_size=500):
    sentences = sent_tokenize(text)
    chunks = []
    current_chunk = ""
    for sent in sentences:
        if len(current_chunk) + len(sent) <= max_chunk_size:
            current_chunk += " " + sent
        else:
            chunks.append(current_chunk.strip())
            current_chunk = sent
    if current_chunk:
        chunks.append(current_chunk.strip())
    return chunks

chunks = segment_text(raw_text)
print(f"Number of text chunks: {len(chunks)}")
```

4.2. Embedding and Retrieval

To enable semantic search, each chunk is converted into an embedding vector using pretrained Sentence Transformer models. Queries are similarly embedded, and nearest neighbour search retrieves the most relevant chunks.

4.2.1. Tools

- sentence-transformers: Pretrained models to generate semantically meaningful embeddings.
- FAISS or scikit-learn: For fast vector similarity search (optional for large corpora).

4.2.2. Example Code

```
python
from sentence_transformers import SentenceTransformer
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity

# Load pretrained Sentence Transformer
model = SentenceTransformer('all-MiniLM-L6-v2')

# Encode all chunks
chunk_embeddings = model.encode(chunks)

def retrieve_relevant_chunks(query, top_k=3):
    query_embedding = model.encode([query])
    similarities = cosine_similarity(query_embedding, chunk_embeddings)[0]
    top_indices = np.argsort(similarities)[::-1][:top_k]
    relevant_chunks = [chunks[i] for i in top_indices]
    return relevant_chunks

# Example
query = "What are the postgraduate courses?"
relevant_chunks = retrieve_relevant_chunks(query)
for i, chunk in enumerate(relevant_chunks):
    print(f"Chunk {i+1}: {chunk}\n")
```

4.3. QA Pipeline

Using Hugging Face's transformers, a pretrained transformer-based QA model fine-tuned on datasets like SQuAD extracts precise answers from the retrieved chunks.

4.3.1. Tools

- transformers library from Hugging Face
- Models like deepset/roberta-base-squad2 or distilbert-base-uncased-distilled-squad

4.3.2. Example Code

```
python
from transformers import pipeline

# Load QA pipeline
qa_pipeline = pipeline("question-answering", model="deepset/roberta-base-squad2")

def answer_question(question, context_chunks):
    combined_context = " ".join(context_chunks)
    result = qa_pipeline(question=question, context=combined_context)
    return result['answer']

# Example usage
answer = answer_question(query, relevant_chunks)
print(f"Answer: {answer}")
```

4.4. Contact Scraper

Since universities often update their contact info, a scraper runs periodically to fetch the latest WhatsApp or phone numbers from official websites.

4.4.1. Tools

- requests for HTTP requests
- BeautifulSoup for HTML parsing
- re for regex pattern matching phone numbers

4.4.2. Example Code

python

```
import requests
from bs4 import BeautifulSoup
import re

def scrape_contact_info(url):
    try:
        response = requests.get(url, timeout=10)
        response.raise_for_status()
        soup = BeautifulSoup(response.text, "html.parser")

        # Find phone or WhatsApp numbers using regex
    phone_pattern=re.compile(r'(\+91[-\s]?)?[0]?(91)?[789]\d{9}')
        text = soup.get_text(separator=" ", strip=True)
        matches = phone_pattern.findall(text)

        # Return unique cleaned phone numbers
        phone_numbers = set()
        for match in matches:
            # Clean match tuple and join
            number = "".join(match)
            phone_numbers.add(number)

        return list(phone_numbers)
    except Exception as e:
        print(f"Error scraping contacts: {e}")
        return []

# Usage
contact_page_url = "https://geetauniversity.edu.in/contact"
contacts = scrape_contact_info(contact_page_url)
print("Extracted contacts:", contacts)
```

4.5. Booking Link Generator

This module generates WhatsApp URLs with pre-filled messages, enabling users to confirm admission booking with one click.

4.5.1 WhatsApp URL Format

php-template

```
https://wa.me/<phone_number>?text=<url_encoded_message>
```

4.5.2 Example Code

python

```
import urllib.parse
def generate_whatsapp_link(phone_number, course_name):
    base_url = f"https://wa.me/{phone_number}"
    message = f"Hello, I would like to book admission for the course: {course_name}. Please assist."
    encoded_message = urllib.parse.quote(message)
    full_url = f"{base_url}?text={encoded_message}"
    return full_url

# Usage
if contacts:
    phone = contacts[0]  # Select first contact
    course = "M.Tech Computer Science"
    link = generate_whatsapp_link(phone, course)
    print("Booking Link:", link)
```

4.6 User Interface

The final module presents an interface for users to ask questions and book admissions. This can be implemented as a web app or chatbot.

4.6.1 Web App Example Using Flask

python

```
from flask import Flask, request, render_template, jsonify

app = Flask(__name__)

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

@app.route('/ask', methods=['POST'])
def ask():
    user_query = request.json.get('question')
    relevant_chunks = retrieve_relevant_chunks(user_query)
    answer = answer_question(user_query, relevant_chunks)
    return jsonify({'answer': answer})

@app.route('/book', methods=['POST'])
def book():
    course = request.json.get('course')
    if contacts:
        phone = contacts[0]
        link = generate_whatsapp_link(phone, course)
        return jsonify({'booking_link': link})
    return jsonify({'error': 'Contact info unavailable'}), 500

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

4.6.2 Frontend Sample (index.html)

```
<!DOCTYPE html>
<html>
<head>
    <title>Admission Assistant</title>
    <script>
        async function askQuestion() {
            const question = document.getElementById('question').
value;
            const response = await fetch('/ask', {
```

```
            method: 'POST',
            headers: {'Content-Type': 'application/json'},
            body: JSON.stringify({question})
        });
        const data = await response.json();
            document.getElementById('answer').innerText = data.
answer;
    }

    async function bookAdmission() {
        const course = document.getElementById('course').value;
        const response = await fetch('/book', {
            method: 'POST',
            headers: {'Content-Type': 'application/json'},
            body: JSON.stringify({course})
        });
        const data = await response.json();
        if (data.booking_link) {
            window.open(data.booking_link, '_blank');
        } else {
            alert('Booking link unavailable');
        }
    }
  </script>
</head>
<body>
  <h1>Admission Assistant</h1>
    <input type="text" id="question" placeholder="Ask about
admission..."/>
  <button onclick="askQuestion()">Ask</button>
   <p><strong>Answer:</strong> <span id="answer"></span></
p>

  <h3>Book Admission</h3>
    <input type="text" id="course" placeholder="Enter course
name"/>
   <button onclick="bookAdmission()">Book via WhatsApp</
button>
</body>
</html>
```

The modular pipeline architecture allows individual components to be improved independently. PDF parsing uses fallback OCR to handle diverse brochure formats. Semantic retrieval with Sentence Transformers enables understanding beyond keywords. The QA system offers precise, contextual answers by leveraging powerful transformer models. The contact scraper ensures that communication channels are always current. The WhatsApp link generator seamlessly connects users with admissions staff through a popular platform. Finally, the user interface enables smooth user interaction, either as a web chatbot or integrated into larger systems. This implementation provides a scalable, maintainable solution that can be extended with features like multi-lingual support, user authentication, analytics, or automated booking confirmation workflows.

5. Results and Discussion

Testing on Geeta University's brochure and website demonstrated accurate retrieval of course and fee information, with the WhatsApp link facilitating immediate user contact. Challenges included handling noisy PDF formatting and ambiguous queries, which were mitigated by fallback responses and rule-based filters. The scraping approach successfully updated contact details without manual input.

6. Conclusion

This research presents an innovative system combining NLP and web scraping for automated admission assistance and booking via WhatsApp. The approach enhances user convenience, reduces administrative workload, and can be generalized to other institutions. Future work includes extending multilingual support and integrating direct WhatsApp Business API messaging for full automation [4-25].

References

1. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019, June). Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1* (long and short papers) (pp. 4171-4186).
2. Reimers, N., & Gurevych, I. (2019, November). Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)* (pp. 3982-3992).
3. Rajpurkar, P., Zhang, J., Lopyrev, K., & Liang, P. (2016, November). Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 conference on empirical methods in natural language processing* (pp. 2383-2392).
4. Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., ... & Rush, A. M. (2020, October). Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations* (pp. 38-45).
5. Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., ... & Stoica, I. (2016). Apache spark: a unified engine for big data processing. *Communications of the ACM, 59*(11), 56-65.
6. Smith, R. (2007, September). An overview of the Tesseract OCR engine. In *Ninth international conference on document analysis and recognition (ICDAR 2007)* (Vol. 2, pp. 629-633). IEEE.
7. Mitchell, T. M. (1997). Machine learning. *McGraw-Hill.*
8. Grover, A., & Leskovec, J. (2016, August). node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 855-864).
9. Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space.

arXiv preprint arXiv:1301.3781.

10. Loper, E., & Bird, S. (2002, July). Nltk: The natural language toolkit. In *Proceedings of the ACL-02 Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics* (pp. 63-70).
11. Ferrara, E., De Meo, P., Fiumara, G., & Baumgartner, R. (2014). Web data extraction, applications and techniques: A survey. *Knowledge-based systems, 70*, 301-323.
12. Kumar, R., & Krishna, A. (2021). Automated chatbot system for admission enquiries using NLP and machine learning. *International Journal of Advanced Computer Science and Applications, 12*(1), 256–263.
13. Gupta, A., & Chopra, A. (2020). Use of WhatsApp in customer engagement: A study on consumer attitudes. *International Journal of Information Management, 54*, 102146.
14. Anwar, F., & Ullah, I. (2021). WhatsApp for business: Communication paradigms in the post-digital age. *Journal of Business Research, 134*, 652–663.
15. Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C., & Liu, C. (2018, September). A survey on deep transfer learning. In *International conference on artificial neural networks* (pp. 270-279). Cham: Springer International Publishing.
16. Scholman, M. C., & Demberg, V. (2017). Crowdsourcing discourse relations: Quality management and analysis of the PDTB annotated data. *Corpus Linguistics and Linguistic Theory.*
17. Li, Y., et al. (2022). A survey on retrieval-augmented generation. *arXiv preprint.*
18. Dhingra, B., Liu, H., Yang, Z., Cohen, W., & Salakhutdinov, R. (2017, July). Gated-attention readers for text comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics* (Volume 1: Long Papers) (pp. 1832-1846).
19. Kamath, A., et al. (2021). MDQA: Multi-document question answering via unsupervised summarization. *EMNLP Findings*.
20. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. *Advances in neural information processing systems, 33*, 1877-1901.
21. Jurafsky, D., & Martin, J. H. (2020). *Speech and language processing* (3rd ed., draft). Pearson.
22. Nogueira, R., & Cho, K. (2019). Passage re-ranking with BERT. *arXiv preprint.*
23. Salton, G., & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. *Information processing & management, 24*(5), 513-523.
24. Yang, W., Xie, Y., Lin, A., Li, X., Tan, L., Xiong, K., ... & Lin, J. (2019, June). End-to-end open-domain question answering with bertserini. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistic*s (Demonstrations) (pp. 72-77).
25. Berners-Lee, T., Hendler, J., & Lassila, O. (2001). Web semantic. *Scientific American, 284*(5), 34-43.