Brief Report

Simplification and Translation of Medical Reports Using Large Language Models- A Protocol for the Indian Context

Bhavin Jain and Sandeep Reddy *

Posted Date: 13 August 2025

doi: 10.20944/preprints202508.0955.v1

Keywords: medical report simplification; large language model; cloud cognitive services; patient communication; AI in healthcare

 doi:10.20944/preprints202508.0955.v1

Brief Report

Simplification and Translation of Medical Reports Using Large Language Models-A Protocol for the Indian Context

Bhavin Jain [1] and Sandeep Reddy [2,*]

[1] Indian Institute of Technology, Mumbai, India

[2] Centre for Data Science, Queensland University of Technology, Brisbane, Australia

* Correspondence: sandeep.reddy@qut.edu.au

Abstract

Medical reports are often written in complex English, making them difficult for the average patient to understand, especially in regional and rural areas where English proficiency is low. This paper presents a cloud-based solution leveraging Large Language Models (LLMs), specifically OpenAI's GPT-4.1, deployed through Microsoft Azure, to simplify radiology reports. The system features a secure, scalable web interface that allows users to input medical reports and receive simplified outputs. The architecture emphasises security, compliance, and modularity, allowing future enhancements such as translation into Indian languages and OCR for scanned reports. Initial tests show promising results in improving readability. Future efforts include fine-tuning the model with expert-reviewed data, implementing quality checks, and incorporating user feedback to ensure medical accuracy and usefulness. This approach aims to bridge the language gap in Indian healthcare, promote health literacy, and empower patients with accessible medical information.

Keywords: medical report simplification; large language model; cloud cognitive services; patient communication; AI in healthcare

Background

Medical reports pose difficulties for many patients to read worldwide. [1,2] This is because such reports are predominantly written in complex terminology. The medical vocabulary used in such documents is typically tailored for healthcare professionals, making it inaccessible to the average patient. [1,3] This situation applies particularly to India, where a majority of the population resides in regional and rural areas where English is not widely spoken. [2,3] Additionally, a significant portion of the tier-2 and tier-3 population has English as their third or fourth language. [4,5] As per the 2011 Census [5], 129 million Indians (~10.6%) reported speaking English (as a first, second, or third language)

- First language (mother tongue): ~260,000 individuals (~0.02%)
- Second language: ~83 million (~6.8%)
- Third language: ~46 million (~3.8%)

While the number of people who can understand English has increased in India, the language remains a significant barrier to understanding medical reports. Furthermore, the linguistic diversity of India, with its 22 scheduled languages [6] and numerous regional dialects, exacerbates the issue. This communication gap can lead to confusion, anxiety, non-compliance with treatment, and overall reduced patient satisfaction. [1,2] Despite this pressing need, most healthcare providers in India do not currently offer simplified or translated versions of reports. [3,4]

In recent years, significant advancements have been made in the fields of artificial intelligence (AI) and natural language processing (NLP). [7] Due to these developments, we have considered a

 doi:10.20944/preprints202508.0955.v1

feasible AI pathway to bridge the medical communication gap. AI models can be trained to interpret medical content and present it in layperson's terms while preserving the clinical essence. [8,9] Additionally, the integration of cloud platforms enables the scalable deployment of these models with robust security and compliance. [10,11]

Our project aims to leverage cloud services to simplify medical reports and, in future phases, translate them into local Indian languages. This approach ensures that patients from diverse linguistic backgrounds can understand their health information more clearly, resulting in improved health outcomes and increased trust in the healthcare system. [1,2]

Method

Architecture Overview

For the simplification and translation of the medical report process, we will utilise a cloud platform, Microsoft Azure, as it offers a wide range of features. Therefore, our architecture is built upon this platform. The frontend is developed in HTML, CSS, and JavaScript and is hosted on Azure App Service. This interface allows users to input their medical reports and view simplified versions.

The backend is implemented using Flask, a Python-based microframework for web applications. This backend is responsible for receiving user inputs, forwarding them to the Azure OpenAI service, and returning the simplified outputs. To maintain data privacy and secure communication, the backend can be VNet-integrated, enabling communication with Azure OpenAI through a private endpoint. Azure Key Vault is used to manage sensitive information such as API keys and credentials. Access to the vault can be governed by a managed identity assigned to the backend service, ensuring that secrets are never hardcoded or exposed.

This setup is supported by Azure Functions for modular execution and Azure Monitor for performance tracking and diagnostics. Application Insights is also employed for real-time analytics, error logging, and user behaviour monitoring. This comprehensive architecture ensures secure, scalable, and reliable operations.

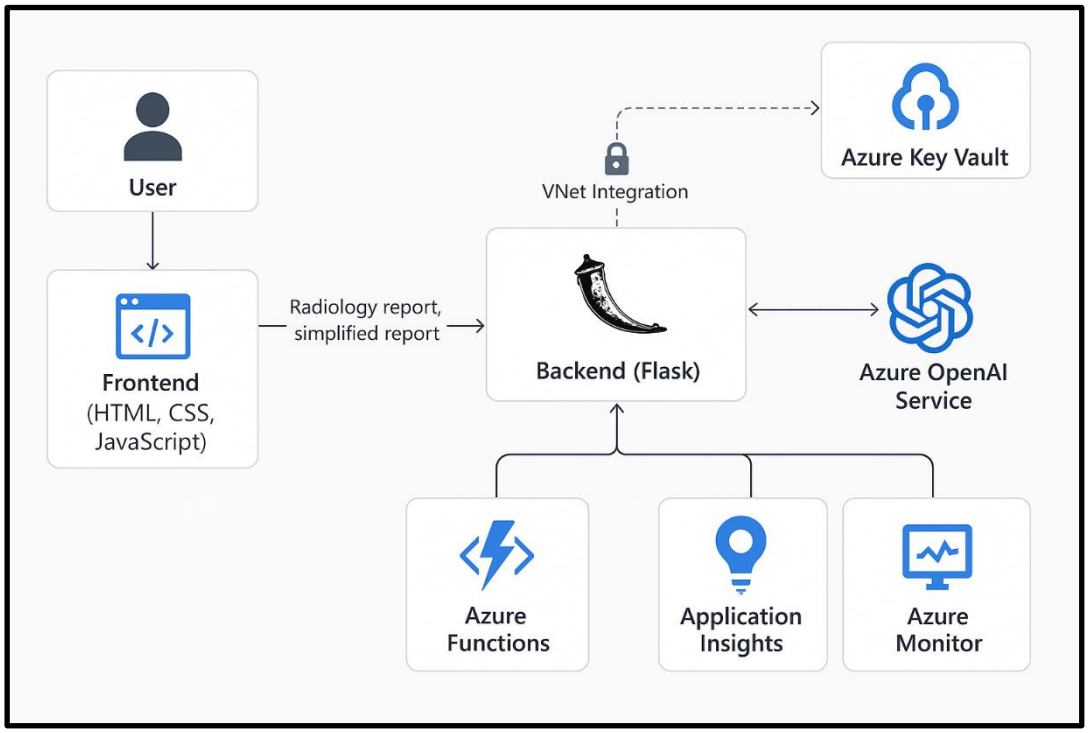

Figure 1. Visual Flow/Diagram of Architecture.

Data Flow

 doi:10.20944/preprints202508.0955.v1

The data flow begins when a user pastes or uploads a medical report through the web application's frontend interface. The input is transmitted over HTTPS to the Flask-based backend hosted on Azure App Service. Upon receiving the report, the backend invokes the relevant Large Language Model (LLM), in this case, the OpenAI GPT-4.1 model, via the Azure SDK, securely accessed through a private endpoint within a virtual network. The backend sends a prompt to the model instructing it to simplify the report using patient-friendly language.

The OpenAI model processes the input and returns a simplified version of the report. The backend parses the model's output, performs any necessary post-processing or content filtering, and sends the cleaned result back to the frontend. The user then views the simplified report directly on the web interface. In the planned future enhancement phase, the output will be routed through Azure Translator for conversion into regional Indian languages. Each step of this pipeline—from input reception to final display—is logged and monitored using Azure Monitor and Application Insights for transparency and debugging purposes.

This data flow architecture ensures that sensitive health data is handled securely and remains compliant with jurisdictional data protection regulations such as those in countries like Australia.

Security & Compliance

Given the sensitive nature of medical data, security and compliance are critical. Our system employs Azure Virtual Network (VNet) integration to isolate the backend service and prevent public exposure of model endpoints. Only the front end is publicly accessible. To protect communication between components, all data transmissions are encrypted via HTTPS. Azure Key Vault stores all secrets, including API keys and tokens. The backend accesses these securely using a managed identity, reducing the risk of credential leakage.

Azure OpenAI model access is further secured using private endpoints, ensuring that data does not traverse the public internet. This configuration is critical, given that some countries' jurisdictions in which the service is hosted have local data privacy laws that prohibit the unrestricted exposure of medical content. We also use Azure's built-in content filtering capabilities to prevent the generation of inappropriate or harmful content. Application-level checks augment this during the development phase and will continue to be monitored in production.

Azure Monitor and Application Insights are integrated to provide real-time visibility into system health, usage, and anomalies. Token usage is tracked to ensure cost-effectiveness and to monitor for any signs of misuse. These measures collectively uphold security, compliance, and ethical standards. Together, these services provide a robust, secure, and scalable cloud-native solution. They allow for automated deployment, seamless integration, and enterprise-grade security. Their modularity also enables straightforward enhancements such as language translation and image OCR, which are part of the planned future roadmap.

Fine Tuning

As per my research, it would be beneficial to fine-tune our LLM model. This can be done by collecting the following dataset:

1. Data from actual de-identified medical reports
2. Simplified Reports Generated by Medical Professionals and Humans to check these reports.

This collection will take time but will help increase the accuracy of our model [4].

Implementation

In the previous section, we detailed an ideal system for achieving this objective. Here, we have detailed how we implemented a system with the limited resources available.

The backend system is developed using Flask, a lightweight Python framework. It contains a RESTful API endpoint /simplify report that accepts POST requests containing medical report text in JSON format. The endpoint processes the request by calling the Azure OpenAI GPT-4.1 model using

 doi:10.20944/preprints202508.0955.v1

the Azure SDK. To deploy this backend, we used GitHub integration with Azure App Service. The codebase, hosted in a private GitHub repository, contains essential files like requirements.txt, startup.sh, and the main Python script. Azure Web App's Deployment Centre automates the pulling and deployment process whenever updates are pushed to the main branch.

Backend Code: For exact code, refer to Appendix A1.

The frontend is a minimal HTML/CSS/JS page that communicates with the backend via JavaScript fetch () calls. It provides a simple interface for inputting medical reports and viewing simplified outputs.

Frontend Code: For exact code, refer to Appendix A2.

Security configurations include the use of HTTPS, as well as environment variable loading and secret management via Azure Key Vault. The startup.sh script ensures that dependencies are installed before launching the application using Gunicorn.

The system also integrates Azure Monitor and Application Insights for operational oversight. Logging is implemented at key checkpoints to facilitate debugging and track user behaviour.

This setup ensures rapid prototyping, secure deployment, and ease of future maintenance. The modularity of the backend allows for quick integration of new features, such as regional language translation or optical character recognition (OCR) for scanned documents.

Results

Initial Test Runs showed an average response time of 10-12 seconds for report simplification. The simplified text, although in English, was much easier to understand and significantly reduced the complexity. It is worth noting that these promising results were observed before the model underwent fine-tuning with a focused dataset. Currently, the system only handles English-only inputs and outputs. However, one can also plan to integrate the Azure Translator to support Indian languages such as Hindi, Marathi, Tamil, Kannada, etc. One can also include OCR for scanned documents using Azure CV, as this will make it easier for the user to upload the reports in various formats. It can be an image, a PDF, etc. As we receive more data, we can refine post-processing to achieve better grammatical clarity.

Figure 2. Image of sample website.

 doi:10.20944/preprints202508.0955.v1

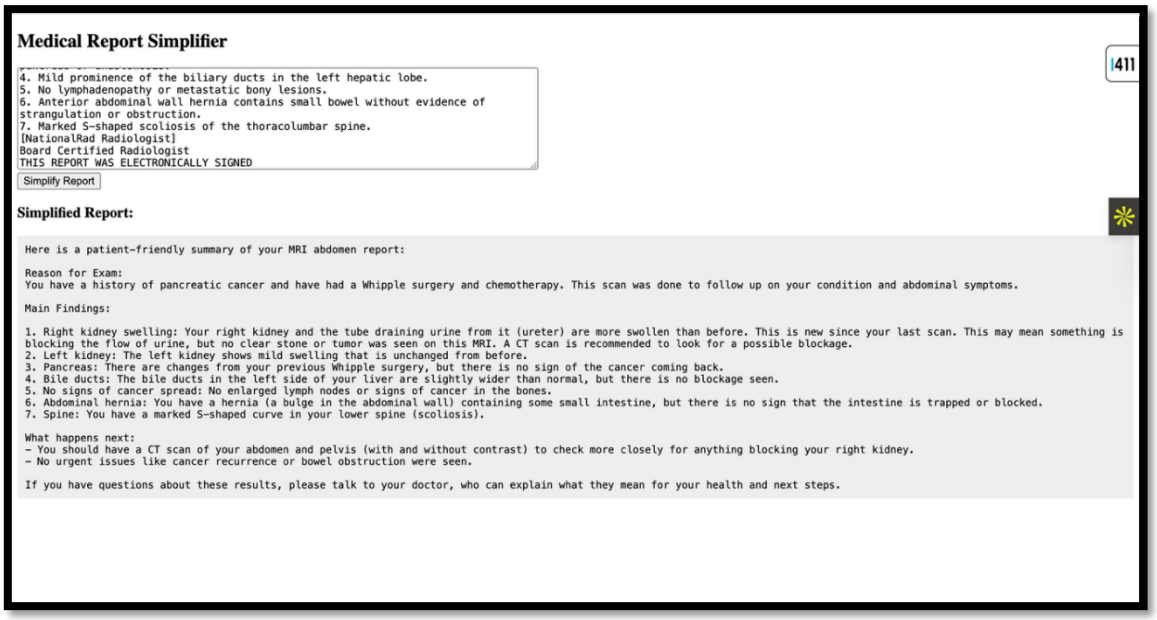

Figure 3. Image of Sample Result Generated on the website.

It is often seen that multiple reports from different tests are combined to diagnose a case. We can develop or fine-tune a model to handle such a batch of reports and give better results to the average user. How can we verify the accuracy of such simplified reports, since models sometimes start hallucinating and give incorrect outputs, which can lead to panic in the user, which would be counterproductive to our purpose.

For this, we have thought of multiple checks:

1. Fine-tuning our model: Once our model is fine-tuned on reports and simplified versions written by medical professionals and tested by humans for simplicity, the model's results will, of course, improve.
2. Random Checks for the first 6 months: For the first 3 or 6 months, one can conduct regular checks on the results being generated to identify any discrepancies. We can also run an external model on the results daily to compare the reports and the output generated, and output a verification score, which will help us flag any discrepancies. As this model is external, it will not malfunction, and the chances of malfunction will be drastically reduced.
3. Other innovative checks, such as user reviews and a dedicated helpline, can also provide users with confidence.

This evaluation approach will ensure that the model's output is not only technically accurate but also valuable and understandable to non-medical users. User validation is a critical component of our development roadmap. We plan to conduct clinician-led evaluations, where radiologists and general practitioners will review the AI-simplified reports for clinical accuracy and relevance.

Additionally, the organisation will need to form focus groups comprising patients from diverse linguistic backgrounds. These participants will review both the original and simplified reports and provide feedback on understandability and perceived usefulness. We can also develop a feedback module within the application to collect user ratings and comments in real-time. This will help us iteratively improve the simplification algorithms and translation accuracy. These user testing efforts should be conducted in collaboration with local hospitals and academic institutions to ensure a broad and representative evaluation.

Discussion

This protocol presents a structured and scalable approach to addressing the longstanding challenge

 doi:10.20944/preprints202508.0955.v1

of medical report comprehension in India.(3,4) Given the dominance of English in clinical documentation and the country's vast linguistic diversity (5), a significant communication gap exists between healthcare providers and patients. Our solution leverages Azure-hosted Large Language Models (LLMs) to simplify radiology reports into patient-friendly language, delivered via a secure and intuitive web interface. The architecture ensures data privacy through virtual networks, private endpoints, and managed identity-based authentication, aligning with international compliance standards. Furthermore, the integration of services like Azure Monitor and Application Insights offers continuous system health tracking and cost control, making the deployment both efficient and enterprise-ready.

The implementation narrative demonstrates that even with limited resources, a working prototype can be built using open-source tools and cloud-native components. Initial results are promising, showing reduced complexity in the simplified outputs and reasonable response times. Looking ahead, the protocol includes fine-tuning the LLM with annotated datasets, incorporating multilingual support using Azure Translator, and expanding input modalities through OCR for scanned documents. To ensure accuracy and user trust, the plan also emphasises post-deployment validation, including clinician-led reviews and real-time patient feedback loops. This project not only proposes a technical solution but also establishes a user-centric and ethically sound framework for transforming how Indian patients interact with their medical information.

Ethical Statement

The patient data used was synthetic/test data. Any real patient data, if used, must be de-identified and used with the patient's consent. Currently, no real patient data was used; hence, ethics approval and patient consent were not required.

Appendix A

Appendix A.1. Backend Code

```
from flask import Flask, request, jsonify, send_from_directory
    from openai import AzureOpenAI
    import os

    app = Flask(__name__)

    # Load environment variables
    endpoint = "https://proje-mbb39o6p-swedencentral.services.ai.azure.com/"

    subscription_key =
    "BU5FjAvMdze1GiTiB1psRRvErzusatjdcfq2qOr22S0KOj581M51JQQJ99BEACfhMk5XJ3w3AAAAACO
    GCtC8"
    api_version = "2024-12-01-preview"
    deployment = "gpt-4.1"

    # Initialize AzureOpenAI client
    client = AzureOpenAI(
        api_version=api_version,
        azure_endpoint=endpoint,
        api_key=subscription_key,
    )
    @app.route('/')
    def serve_index():
```

 doi:10.20944/preprints202508.0955.v1


```
    return send_from_directory('.', 'index.html')



@app.route('/simplify_report', methods=['POST'])
def simplify_report():
    data = request.get_json()
    print("Received data:", data)    # Debug print
    report = data.get('report', '')

    if not report:
        return jsonify({'error': 'No report provided'}), 400

    try:
        response = client.chat.completions.create(
            messages=[
                {
                    "role": "system",
                    "content": "You are a helpful assistant that simplifies radiology reports for patients.",
                },
                {
                    "role": "user",
                    "content": report,
                }
            ],
            max_completion_tokens=500,
            temperature=0.7,
            model=deployment
        )
        simplified_report = response.choices[0].message.content.strip()
        print("Simplified report:", simplified_report)    # Debug print
        return jsonify({"simplified_report": simplified_report})

    except Exception as e:
        print("Error:", e)    # Debug print
        return jsonify({"error": str(e)}), 500

if __name__ == '__main__':
    import os
    port = int(os.environ.get("PORT", 5000))
    app.run(host='0.0.0.0', port=port, debug=True)
```

Appendix A.2. Frontend Code

```
<!DOCTYPE html>
    <html lang="en">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1" />
        <title>Radiology Report Simplifier</title>
```

 doi:10.20944/preprints202508.0955.v1


```
</head>
<body>
    <h2>Radiology Report Simplifier</h2>
    <textarea id="reportInput" rows="8" cols="80" placeholder="Paste radiology report here..."></textarea><br />
    <button onclick="simplifyReport()">Simplify Report</button>
    <h3>Simplified Report:</h3>
    <pre id="resultOutput" style="background:#eee; padding:10px;"></pre>

    <script>
        async function simplifyReport() {
            const reportText = document.getElementById('reportInput').value;
            if (!reportText.trim()) {
                alert('Please enter a radiology report.');
                return;
            }
            document.getElementById('resultOutput').textContent = 'Simplifying...';

            try {
                const response = await fetch('/simplify_report', {
                    method: 'POST',
                    headers: {
                        'Content-Type': 'application/json'
                    },
                    body: JSON.stringify({ report: reportText })
                });

                const data = await response.json();
                if (response.ok) {
                    document.getElementById('resultOutput').textContent = data.simplified_report;
                } else {
                    document.getElementById('resultOutput').textContent = `Error: ${data.error}`;
                }
            } catch (error) {
                document.getElementById('resultOutput').textContent = 'Network error or backend is not running.';
            }
        }
    </script>
</body>
</html>
```

References

1. Haun KH, Patel NR, French DD, Campbell RR, Bradham DD, Lapcevic WA. Health literacy and healthcare outcomes: a systematic review. Patient Educ Couns. 2014;94(1):107-17.
2. Richards T. Health literacy matters. BMJ. 2024;385:q879. doi:10.1136/bmj.q879.
3. Nigam N, Thaha H, Kumar A, Wadhawan R. Tackling non-communicable diseases in India: The role of health literacy. Int J Res Med Sci. 2024;6(2):7-11. doi:10.33545/26648733.2024.v6.i2a.65.

 doi:10.20944/preprints202508.0955.v1

4. Passi R, Kaur M, Lakshmi PVM, Cheng C, Hawkins M, et al. Health literacy strengths and challenges among residents of a resource-poor village in rural India: Epidemiological and cluster analyses. PLOS Glob Public Health. 2023;3(2):e0001595. doi:10.1371/journal.pgph.0001595.
5. Office of the Registrar General & Census Commissioner, India. Census of India 2011: Language. New Delhi: Office of the Registrar General & Census Commissioner, Ministry of Home Affairs, Government of India; 2011.
6. Government of India. Eighth Schedule to the Constitution of India. New Delhi: Ministry of Law and Justice; 2007.
7. Reddy S, Fox J, Purohit MP. Artificial intelligence-enabled healthcare delivery. J R Soc Med. 2019;112(1):22-8. doi:10.1177/0141076818815510.
8. Yang X, Xiao Y, Liu D, et al. Enhancing doctor-patient communication using large language models for pathology report interpretation. BMC Med Inform Decis Mak. 2025;25:36. doi:10.1186/s12911-024-02838-z.
9. Ouyang L, Wu J, Jiang X, et al. Training language models to follow instructions with human feedback. arXiv [Preprint]. 2022. arXiv:2203.02155.
10. Stephan D, Bertsch AS, Schumacher S, Puladi B, Burwinkel M, Al-Nawas B, et al. Improving patient communication by simplifying AI-generated dental radiology reports with ChatGPT: Comparative study. J Med Internet Res. 2025;27:e73337.
11. Almezhghwi K, Hassan MA, Ghadedo A, Belhaj F, Shwehdi R. Medical reports simplification using large language models. In: Abraham A, Bajaj A, Hanne T, Siarry P, editors. Intelligent Systems Design and Applications. ISDA 2023. Lecture Notes in Networks and Systems. Vol. 1046. Cham: Springer; 2024. p. 69-81. doi:10.1007/978-3-031-64813-7_6.