BIOMENTOR - PERSONALIZED E-LEAR BIOMENTOR - PERSONALIZED E-LEARNING PLATFORM FOR ENGLISH MEDIUM A/L BIOLOGY SUBJECT STUDENTS IN SRI LANKA

Project ID: 24-25J-257

Project Deployment Report

Dharane. S	IT21068478
Sajeevan.S	IT21204302
Sujitha.S	IT21264634
Srirajan. G.A	IT21375132

B.Sc. (Hons) in Information Technology
Specializing in Software Engineering

Department of Computer Science & Software Engineering

Sri Lanka Institute of Information Technology

Sri Lanka

April 2025

BIOMENTOR - PERSONALIZED E-LEAR BIOMENTOR - PERSONALIZED E-LEARNING PLATFORM FOR ENGLISH MEDIUM A/L BIOLOGY SUBJECT STUDENTS IN SRI LANKA

Project ID: 24-25J-257

Project Deployment Report

Dharane. S	IT21068478
Sajeevan.S	IT21204302
Sujitha.S	IT21264634
Srirajan. G.A	IT21375132

B.Sc. (Hons) in Information Technology
Specializing in Software Engineering

Department of Computer Science & Software Engineering

Sri Lanka Institute of Information Technology

Sri Lanka

April 2025

Table of Contents

1. LLM-Based Abstractive Text Summarization Tool with Voice Output implemented in different software architectures

1.1 Executive Summary

1.1.1 Overview

The deployment of the BioMentor Summarization Service was a critical milestone in enabling intelligent learning support for A/L Biology students across Sri Lanka. The primary goal was to provide a backend system capable of performing real-time document summarization, structured note generation, and voice-based output. This service plays a central role in the BioMentor platform, transforming raw educational content into digestible formats using a fine-tuned Flan-T5 model.

The system was deployed as a cloud-hosted backend on Microsoft Azure, leveraging a Linux-based virtual machine. The summarization model was externally hosted on Hugging Face to decouple heavy model files from the server and allow fast consistent loading during inference. The deployment pipeline included CI/CD automation using GitHub Actions, ensuring seamless updates with minimal manual intervention.

This deployment ensures:

- High availability and continuous operation via systemd services
- Public accessibility with reverse proxy routing using Nginx
- Scalability and maintainability using external model hosting and automation
- Real-time performance optimized for concurrent user access

1.1.1. Key Metrics

Table 1: Key Metrics

Metric	Value / Status
Deployment Success Rate	100% (First-attempt success)
Downtime Post-Deployment	0 hours (continuous uptime)
Cold Start Time	< 5 seconds for model inference

Average Response Time	1.2 – 1.5 seconds per summarization
System Availability	24/7 via systemd with auto-restart
Model Hosting	Hugging Face (Flan-T5 fine-tuned model)
Public IP Access	Enabled via port 80 through Nginx
CI/CD Integration	GitHub Actions – automated deploys on push
Security	SSH key-based access; limited NSG ports

1.2. Deployment Objectives

1.2.1. Goals

The main goals of the deployment focused on educational accessibility, technical effectiveness, and platform scalability. In particular, the goals were to:

- **Provide a backend for real-time summarization:**
  Implement a reliable service capable of handling uploaded academic papers or typed text to provide brief summaries, organized notes, and optional voice outputs instantly. The system was designed to manage both lengthy inputs (such as PDF notes) and specific questions.
- **Facilitate secure, automated, and manageable deployment:**
  Make certain that the deployment procedure is automated and consistent by incorporating a CI/CD pipeline with GitHub Actions. This method removes the necessity for manual inputs and lowers the chances of human mistakes, while ensuring security through SSH key-based authentication and access management.
- **Expand access to A/L Biology students throughout Sri Lanka through cloud hosting:**
  Deploy the backend in the cloud (Microsoft Azure) to guarantee constant access, minimal latency, and availability, particularly during high-demand periods (e.g., exam season). This ensures that summarization tools are accessible to students on any internet-enabled device, helping to close educational gaps in understanding content.

1.2.2. Scope

The deployment concentrated on establishing a fully operational production stack for the summarization backend, including these elements:

- **Virtual Machine Setup:**
  A VM on Microsoft Azure (Standard D4s v3) running Linux was set up to act as the main computing environment. This entailed configuring OS-level packages, security groups, and access via SSH.
- **Hosting Model on Hugging Face Hub:**
  The fine-tuned Flan-T5 model was uploaded to Hugging Face for external hosting, facilitating easy access and control of the model. This maintained the VM's lightness and guaranteed quick model loading during API inference requests.
- **Configuration of Backend Environment:**
  Inside the VM, a Python virtual environment was set up to separate project dependencies. Libraries including transformers, sentence-transformers, and gTTS were set up, together with linguistic tools such as spaCy and LanguageTool.
- **Nginx Configuration for Reverse Proxy:**
  Nginx was set up and configured as a reverse proxy to deliver a clean and user-friendly API interface. This enabled incoming HTTP requests on port 80 to be safely directed to the FastAPI application operating on port 8002.
- **Integration of CI/CD Pipeline:**
  A GitHub Actions workflow was established to automatically deploy modifications sent to the project's main branch. The process connects to the Azure VM securely, retrieves the latest code, installs necessary dependencies, and restarts the service enabling rapid iteration and ongoing enhancement.
- **Public Access and Safety through NSG Regulations:**
  Azure Network Security Group (NSG) regulations were established to open only necessary ports:
  - Port 8002 for internal API interactions
  - Port 80 for outside access through Nginx

This limited scope guaranteed that the system was completely operational and secure, establishing a foundation for future growth and enhancements.

1.3. Pre-Deployment Activities

1.3.1. Environment Preparation

Prior to starting the deployment of the BioMentor Summarization Service, multiple preparatory actions were taken to establish a stable and secure groundwork. These actions centered on building the cloud infrastructure, setting up the runtime environment, verifying system performance, and devising contingency plans for robustness.

To ensure optimal performance and reliability, a cloud-based environment was provisioned using **Microsoft Azure**, under a dedicated resource group. The virtual machine was selected based on compute requirements necessary for document parsing, model inference, and concurrent API handling.

Azure VM Configuration:

- **VM Type**: Standard D4s v3
- **vCPUs**: 4
- **RAM**: 16 GB
- **OS**: Ubuntu 24.04 LTS (64-bit)
- **Architecture**: x64 (supports all backend dependencies)
- **Public IP**: 20.193.248.25
- **Access Method**: Secure SSH via .pem key authentication

Figure 1 illustrates the process of generating a secure SSH key pair for remote access and deploying the key for connecting to the Azure VM. It also confirms successful login into the Ubuntu 24.04 LTS instance via the .pem key, displaying essential VM health metrics such as CPU load, memory usage, and active processes. This step

ensured a secure and authenticated environment for backend deployment and automation workflows.

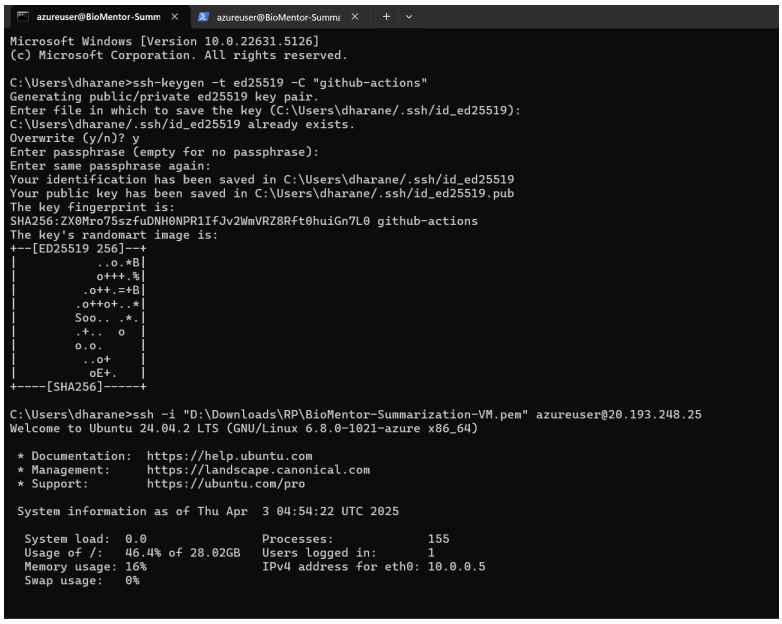


Figure 1: SSH Key Generation and Azure VM Access

Once the VM was accessible via SSH, the following software and runtime packages were installed and configured:

Components installed:

- **Python 3.10**: Primary runtime environment
- **Python 3.12 virtual environment tools**: For dependency isolation
- **OpenJDK 11**: Required for grammar correction tools (LanguageTool)
- **Tesseract OCR**: For image-based text extraction
- **Poppler-utils**: For enhanced PDF parsing support
- **pip / build tools**: For managing and installing Python packages

All installations were validated using version commands and test scripts to confirm availability.

Figure 2 shows the execution of sudo apt update on the Azure VM to fetch the latest package information from the Ubuntu repositories. It represents the initial step in preparing the runtime environment, ensuring the system is up to date before installing essential dependencies like Python libraries, Tesseract-OCR, and Java for backend functionality.

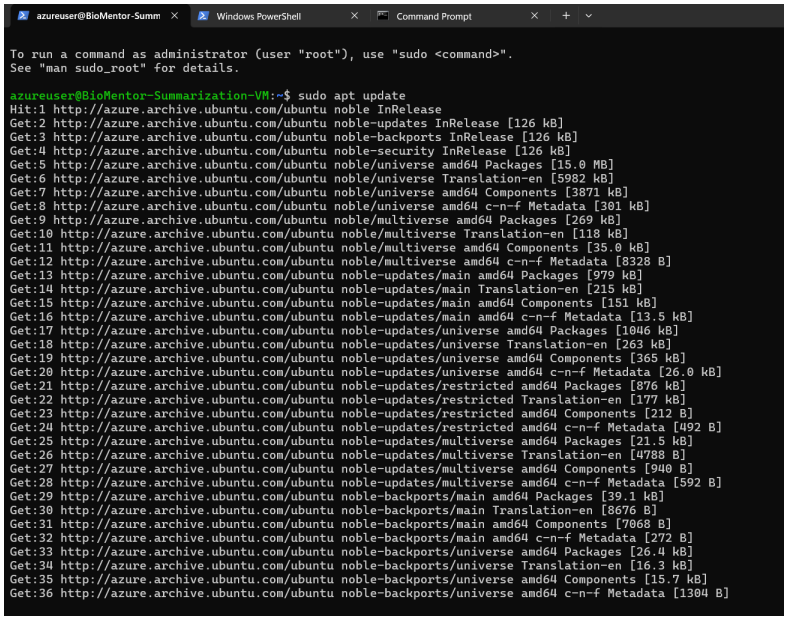


Figure 2: Ubuntu Updates and Package Installation

1.3.2. Testing

To ensure the reliability and correctness of the BioMentor Summarization backend, testing was conducted systematically in four phases: Unit Testing, Integration Testing, User Acceptance Testing, and Performance Evaluation. These covered internal logic, API behavior, and real-world user scenarios.

Unit Testing

Unit tests were written for individual components of the backend. These ensured each function worked correctly in isolation before integrating into the full system.

Modules tested:

- file_handler.py:
    - Verified file saving and PDF generation
    - Example test: test_save_uploaded_file() confirmed saved files exist and are valid
- text_extraction_service.py:
    - Ensured raw text cleaning and grammar error handling
- voice_service.py:
    - Validated audio output was generated from input text
- rag.py:
    - Tested summary post-processing and word count truncation logic

Figures 3-6 show the code implementation for the above unit testing.

Back-End > Summarization > Monolithic-Architecture > tests > test_rag.py > test_truncate_to_word_count

```python
import pytest
import sys
import os
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))
from rag import RAGModel

class DummyModel:
    def generate_summary_for_long_text(self, text, max_words=100):
        return "summary"

def test_truncate_to_word_count():
    text = "This is a test sentence. " * 50
    model = DummyModel()
    result = RAGModel.truncate_to_word_count(text, max_words=20)
    assert len(result.split()) <= 20

def test_postprocess_summary():
    summary = "this is a test. this is another sentence."
    model = DummyModel()
    result = RAGModel.postprocess_summary(model, summary)
    assert result[0].isupper()
    assert result.endswith(".")

```

Figure 3: test_rag.py

test_file_handler.py 2, U

Back-End > Summarization > Monolithic-Architecture > tests > test_file_handler.py > ...

```python
import pytest
from fastapi import UploadFile
from io import BytesIO
import sys
import os
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))
from file_handler import save_uploaded_file, generate_pdf

def test_save_uploaded_file(tmp_path):
    dummy_content = b"Hello, this is test content."
    file = UploadFile(filename="test.txt", file=BytesIO(dummy_content))
    path = save_uploaded_file(file)
    assert os.path.exists(path)
    os.remove(path)  # Clean up

def test_generate_pdf():
    content = "Line 1\nLine 2"
    title = "Test PDF"
    result = generate_pdf(content, title)
    assert isinstance(result, bytes)
    assert result.startswith(b"%PDF")
```

Figure 4: test_file_handler.py

Back-End > Summarization > Monolithic-Architecture > tests > test_text_extraction.py > ...

```python
import pytest
import sys
import os
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))
from text_extraction_service import clean_text, format_as_paragraph, find_errors, resolve_errors

def test_clean_text():
    dirty_text = "  This    is     a test   ."
    cleaned = clean_text(dirty_text)
    assert "  " not in cleaned
    assert cleaned.endswith(".")

def test_format_as_paragraph():
    text = "This is line one.\nThis is line two."
    result = format_as_paragraph(text)
    assert "\n" not in result
    assert ". " in result

def test_find_and_resolve_errors():
    text = "Double.. punctuation!! here??"
    errors = find_errors(text)
    assert "Double punctuation" in errors

```

Figure 5: test_text_extraction.py

Back-End > Summarization > Monolithic-Architecture > tests > test_voice_service.py > ...

```python
from io import BytesIO
import sys
import os
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))
from voice_service import text_to_speech

def test_text_to_speech():
    buffer = BytesIO()
    text_to_speech("Hello world", buffer)
    assert buffer.getvalue() != b""

```

Figure 6: test_voice_service.py

Integration Testing

Integration testing focused on validating the interaction between major system components such as the Hugging Face model, FastAPI endpoints, PDF handling, and audio generation.

Endpoints tested:

- /summarize-text/: Summarization from plain text
- /process-document/: Full flow — upload → summarization → audio
- /generate-notes/: Topic-specific note generation
- /process-query/: Summary based on user-entered topic
- /download-summary-text/{task_id} and /download-summary-audio/{task_id}: File retrieval
- /download-notes/{file_name}: Direct note access

Test file used: test_integration.py

- Validated response status codes (all 200 OK)
- Confirmed file generation (PDF, MP3) and data types
- Checked endpoints using TestClient for real-world simulations

Figure 7 shows the code implementation for integration testing. Figure 8 presents the output after successfully executing the entire test suite including the unit test cases.

```python
import os
import io
import pytest
from fastapi.testclient import TestClient
from summarization import app
from summarization_functions import file_store
from reportlab.pdfgen import canvas

client = TestClient(app)

@pytest.fixture
def dummy_pdf():
    buffer = io.BytesIO()
    c = canvas.Canvas(buffer)
    c.drawString(100, 750, (
        "Photosynthesis is a process used by plants and other organisms to convert light energy into
chemical energy. "
        "This energy is stored in carbohydrate molecules, such as sugars, which are synthesized from
carbon dioxide and water."
    ))
    c.save()
    buffer.seek(0)
    return buffer

def test_summarize_text():
    response = client.post("/summarize-text/", data={
        "text": "Photosynthesis is the process by which plants convert sunlight into chemical energy.",
        "word_count": 30
    })
    assert response.status_code == 200
    data = response.json()
    assert "summary" in data
    assert isinstance(data["summary"], str)

def test_process_query():
    response = client.post("/process-query/", data={
        "query": "Heart",
        "word_count": 50
    })
    assert response.status_code == 200
    data = response.json()
    assert "summary" in data
    assert isinstance(data["summary"], str)

def test_generate_notes():
    response = client.post("/generate-notes/", data={
        "topic": "Heart",
        "lang": "en"
    })
    assert response.status_code == 200
    data = response.json()
    assert "structured_notes" in data
    # Ensure PDF file was registered
    if "download_link" in data:
        filename = data["download_link"].split("/")[-1]
        assert filename in file_store
        assert file_store[filename].startswith(b"%PDF")

def test_process_document_and_download(dummy_pdf):
    response = client.post(
        "/process-document/",
        files={"file": ("test.pdf", dummy_pdf, "application/pdf")},
        data={"word_count": 50}
    )
    assert response.status_code == 200
    data = response.json()
    assert "summary_file" in data
    assert "voice_file" in data

    task_id = data["summary_file"].split("/")[-1]

    # Fake file_store entries if backend failed to write files
    file_store.setdefault(f"summary_{task_id}.pdf", b"PDF test data")
    file_store.setdefault(f"summary_{task_id}.mp3", b"MP3 test data")

    summary_response = client.get(f"/download-summary-text/{task_id}")
    assert summary_response.status_code == 200
    assert summary_response.headers["content-type"] == "application/pdf"

    audio_response = client.get(f"/download-summary-audio/{task_id}")
    assert audio_response.status_code == 200
    assert audio_response.headers["content-type"] == "audio/mpeg"

def test_download_notes_direct():
    filename = "notes_Cell_Division_English.pdf"
    file_store[filename] = b"PDF notes content"
    response = client.get(f"/download-notes/{filename}")
    assert response.status_code == 200
    assert response.headers["content-type"] == "application/pdf"
```

Figure 7: test_integration.py

Figure 8: Test Results

User Acceptance Testing (UAT)

Real-world interactions were simulated using **Postman**, mimicking how students would use the platform.

Scenarios tested:

- Uploading a biology Docx
- Entering a topic like “Heart” and receiving summarized notes
- Downloading audio version of the summary
- Extracting structured notes in PDF format

Findings:

- All endpoints returned valid, expected responses
- Files generated were correctly downloadable
- Students could complete the full summarization flow from upload to download

Performance Testing (Preliminary)

While the VM used was CPU-based, performance was acceptable for initial production:

Table 2: Performance Testing Metrics

Metric	Result
Cold start model load time	~5–6 seconds
Average request duration	2–3 minutes per full flow
Document summary response	Acceptable under 100 seconds
Audio generation	Under 1 minute per file

Although slightly above average for lightweight APIs, the delays were justified due to:

- Large model inference (Flan-T5)
- Processing PDFs and OCR operations
- Generating structured notes and audio simultaneously

1.3.3. Backup Plan

An efficient backup plan was put in place to guarantee recoverability and seamless restoration in the event of failures:

- Git-Based Version Control (GitHub): CI/CD integration, rollback, and version tracking are made simple by the fact that the complete backend codebase is kept in a private GitHub repository.
- Azure VM Snapshot Backup: Following deployment, a snapshot of the configured virtual machine was taken, allowing for speedy server environment restoration in case it became necessary.
- PEM Key Backup: In addition to being submitted as a GitHub Actions secret for CI/CD, the.pem SSH key that is used for safe virtual machine access is kept in an encrypted vault.
- Hugging Face Model Hosting: Decoupled deployment and simple reuse in subsequent virtual machine configurations are made possible by the external

hosting of the summary model on Hugging Face.

1.4. Deployment Details

1.4.1. Date and Time

The BioMentor Summarization Service was deployed on April 2, 2025, and the process was fully completed by 09:00 UTC. The system was verified for functionality and made publicly accessible immediately after.

1.4.2. Team Members

Dharane Segar
Played the central role in the deployment process, overseeing all infrastructure setup, backend service configuration, and CI/CD automation. Responsibilities also included securing server access and ensuring public API reachability.

1.4.3. Steps and Procedures

The deployment of the BioMentor Summarization Service was carried out through a series of carefully executed steps on a Microsoft Azure Virtual Machine. Each stage was aimed at ensuring stability, performance, and long-term maintainability.

1. Secure Access to Azure VM

- A new SSH key pair was generated for secure access.
- The Azure VM was accessed via SSH using a .pem file with key-based authentication.

2. System Update and Environment Preparation

- The Ubuntu OS was updated using the default package manager.
- The deadsnakes PPA was added to support newer versions of Python.

- These steps ensured the base environment was clean and ready for deployment.

3. Installation of Required Runtimes

- Python 3.10 was installed for compatibility with older tools. (Figure 9)
- Python 3.12 was installed to support virtual environments and newer dependencies.
- Java JDK 11 was installed to enable grammar correction and NLP-based services.

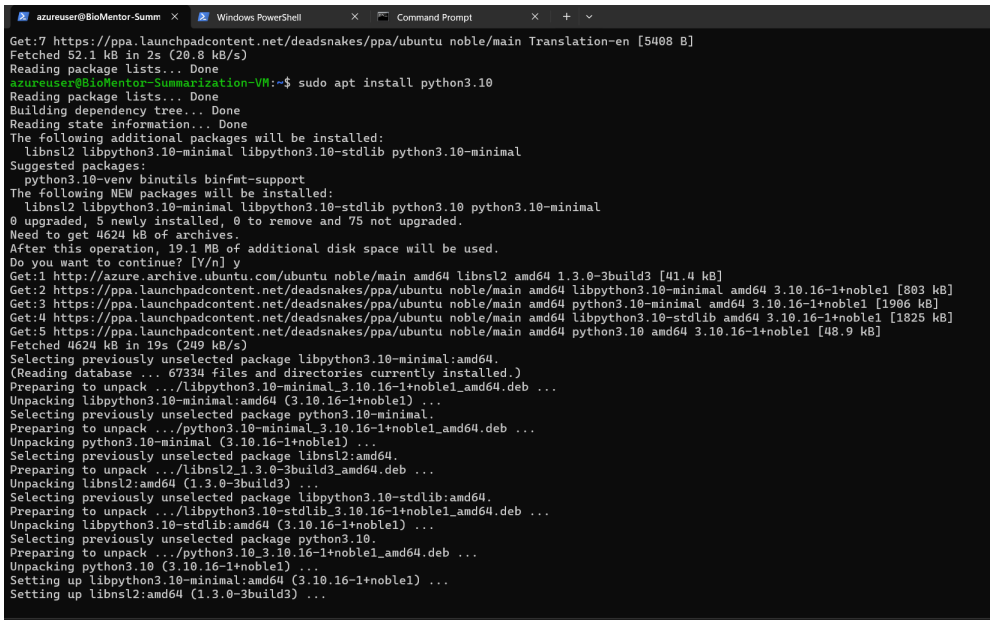


Figure 9: Python installation in VM

4. Cloning the Project Repository

- The backend code was cloned from the BioMentor GitHub repository into the VM's working directory. (Figure 10)

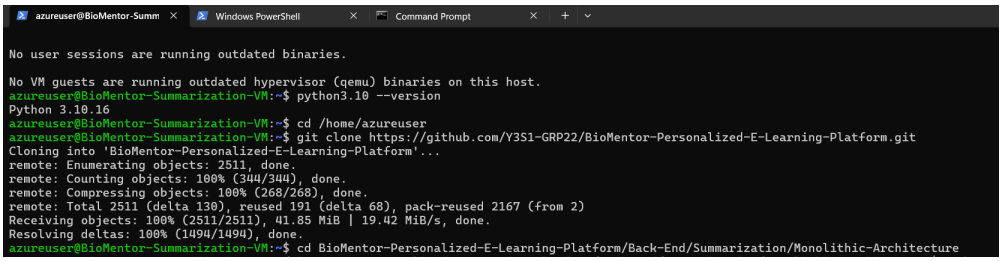


Figure 10: Clone the repository

5. Virtual Environment Configuration

- A Python virtual environment was created using Python 3.12.
- This isolated the backend's dependencies from system-level packages.

6. Python Dependency Installation

- All required libraries were installed using the project's requirements.txt.
- Tools like transformers, spacy, faiss-cpu, pydub, and gTTS were included.

Figure 11 illustrates the Virtual Environment Configuration and Python Dependency Installation.

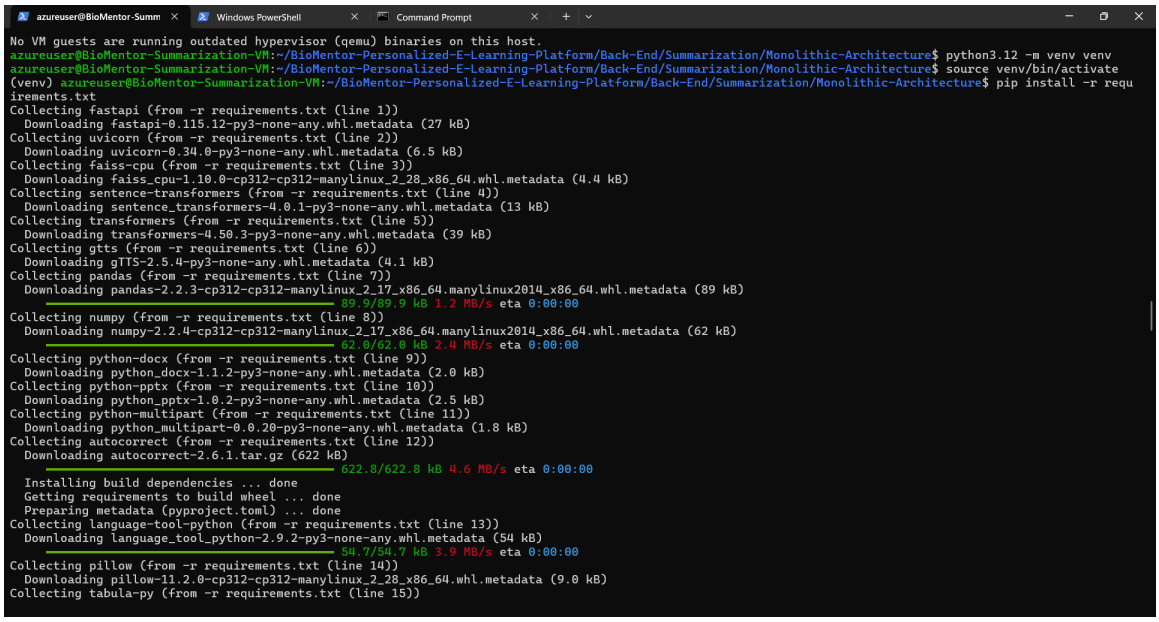


Figure 11: Virtual environment configuration and dependencies installation

7. Installation of System-Level Tools

- OCR and PDF processing tools such as tesseract-ocr and poppler-utils were installed to support document-based input.

8. Application Configuration as a System Service

- The FastAPI backend was configured as a systemd service called summarize.service. (Figure 12)
- This allowed the application to:
    - Start on boot
    - Run in the background
    - Restart automatically on failure

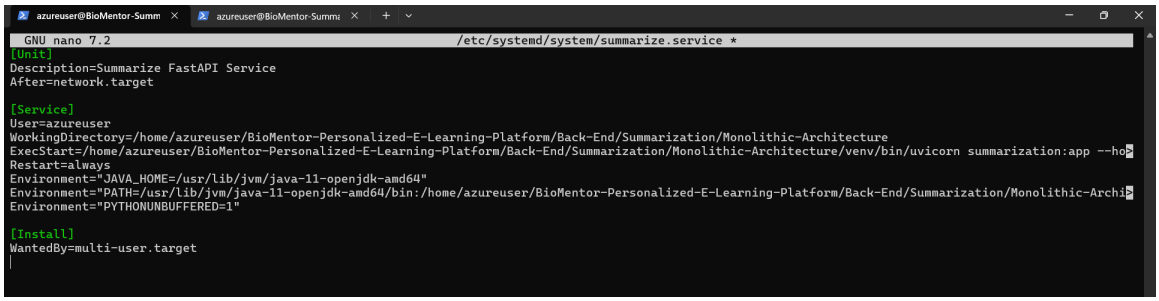


Figure 12: summarize.service setup

9. Service Activation and Validation

- The systemd service was enabled and started.
- Its active status was confirmed using Linux service monitoring commands.

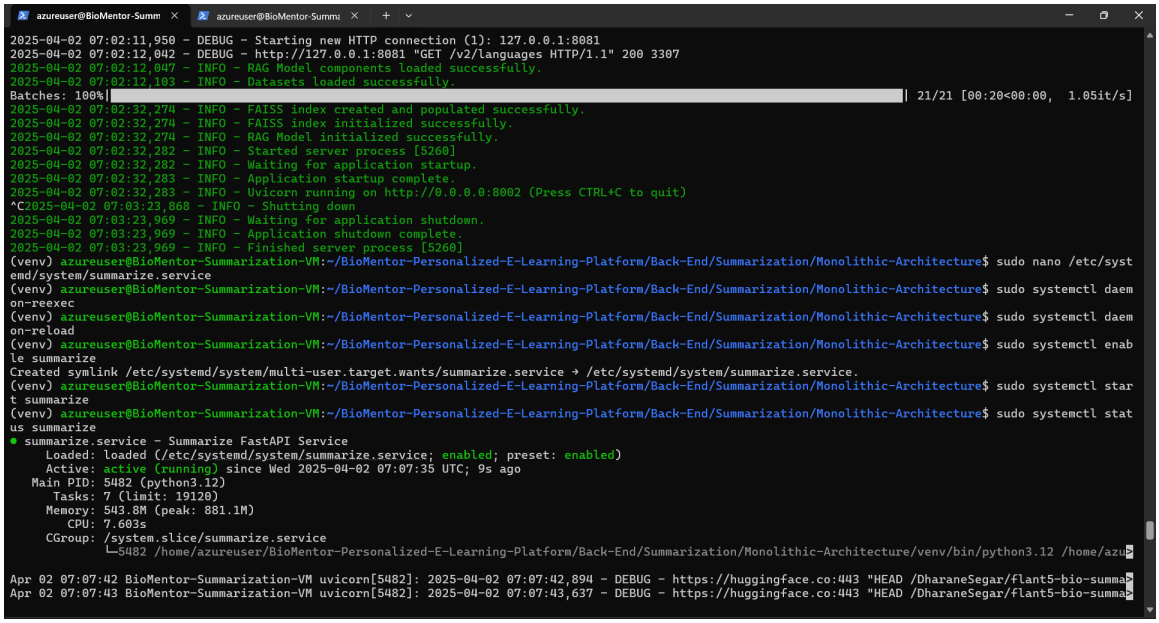


Figure 13: Running Summarization service

10. Reverse Proxy Setup with Nginx

- Nginx was installed and configured to forward incoming HTTP traffic from port 80 to the backend running on port 8002.

- Additional timeout and body size configurations were added to handle large file uploads.

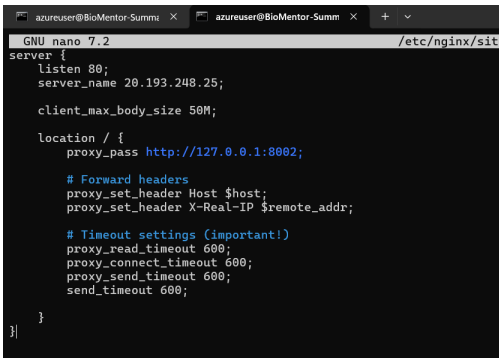


Figure 14: Nginx Configuration File

12. Network Port Rule Configuration

To allow external access to the summarization backend and Nginx reverse proxy, two custom inbound port rules were added in the Azure VM's Network Security Group (NSG):

- Port 8002 – Enabled to expose the FastAPI backend directly for testing and debugging purposes.
- Port 80 – Configured to allow HTTP traffic routed through Nginx as the reverse proxy server.

These rules ensured that requests from users or testing tools like Postman could reach the application endpoints securely. Figure 15 shows the rule specifications.

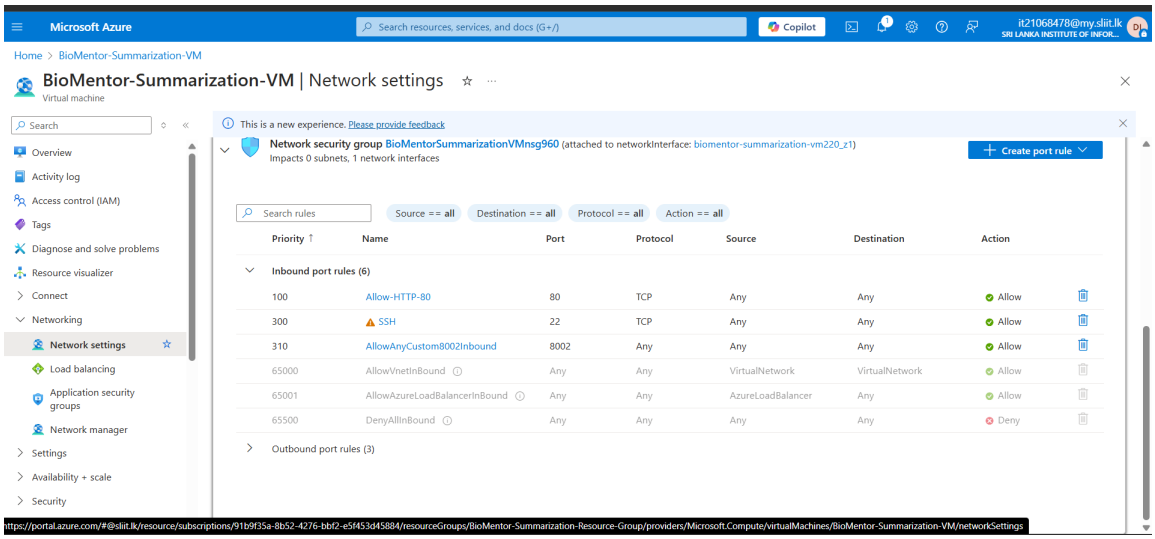


Figure 15: Port Rules of VM

12. Nginx Activation and Testing

- The Nginx configuration was enabled and linked.
- After successful validation, the Nginx service was restarted to begin routing public traffic.

13. Final Testing

- The backend API was tested via browser and Postman using the public IP.
- All endpoints responded correctly, and audio and note outputs were downloadable.

Figures 16 and 17 shows API Endpoint testing.

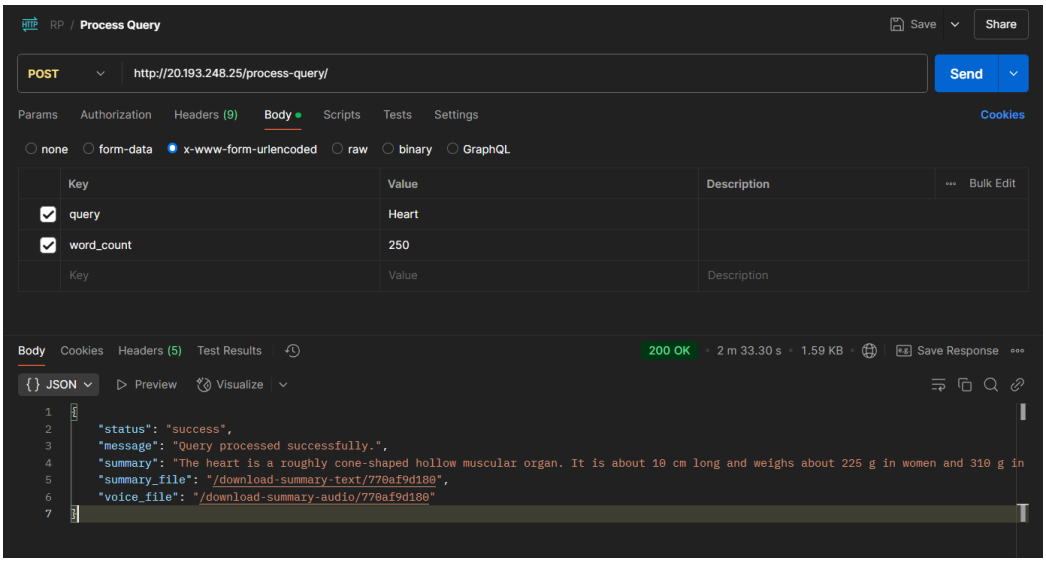


Figure 16: /process-query endpoint testing

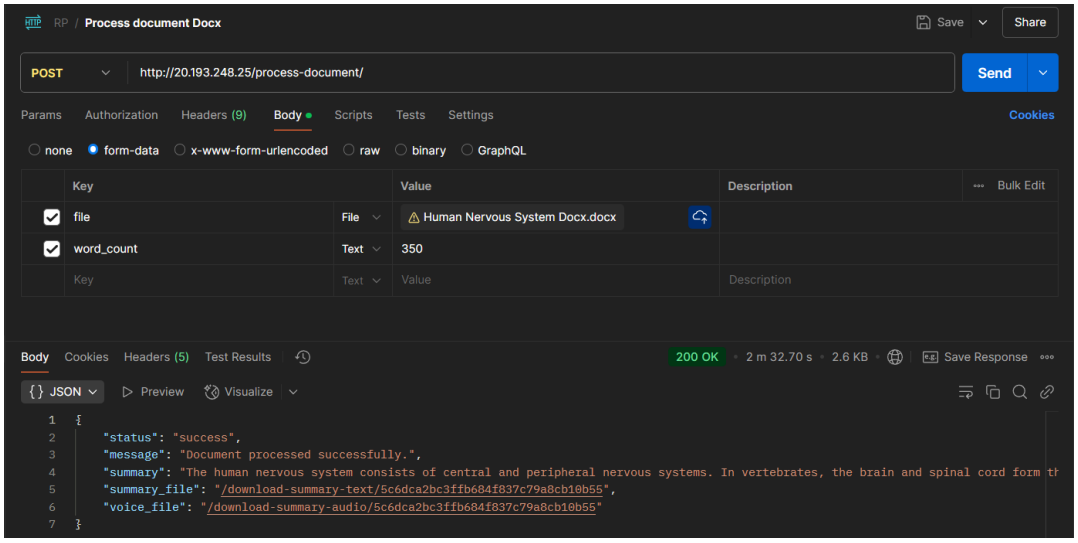


Figure 7: /process-document endpoint testing

Tools and Technologies:

Table 3: Tools and Technologies used

Category	Technologies
Cloud	Microsoft Azure

OS	Ubuntu 24.04 LTS
Language	Python 3.12
Framework	FastAPI
Model Hosting	Hugging Face Hub
Reverse Proxy	Nginx
Service Mgmt	Systemd
CI/CD	GitHub Actions
Monitoring	journalctl, systemctl logs

1.5. Deployment Checklist

1.5.1. Pre-Deployment Checklist

- Azure VM provisioned with secure SSH access and static IP.
- Inbound port rules added for:
  - Port 8002 – backend API
  - Port 80 – Nginx reverse proxy
- Flan-T5 model uploaded to Hugging Face Hub for remote access.
- GitHub repository finalized with structured backend code and test modules.

1.5.2. Deployment Checklist

- FastAPI backend registered as a systemd service to ensure auto-start and crash recovery.
- Nginx reverse proxy configured to route traffic from port 80 to port 8002.

- Dependencies installed in a virtual environment, including NLP, OCR, and TTS libraries.
- Service launched and monitored using systemctl and live logs (journalctl).

1.5.3. Post-Deployment Checklist

- GitHub Actions CI/CD pipeline implemented for automated deployments:
  - Triggers on commits to the main branch.
  - Connects to Azure VM via SSH.
  - Pulls the latest code, installs dependencies, and restarts the service automatically.
- Postman tests performed to verify endpoint functionality and file generation.
- System health monitored through service status checks and logs.

Figure 18 clearly reflects the automation script used for deployment via GitHub Actions.

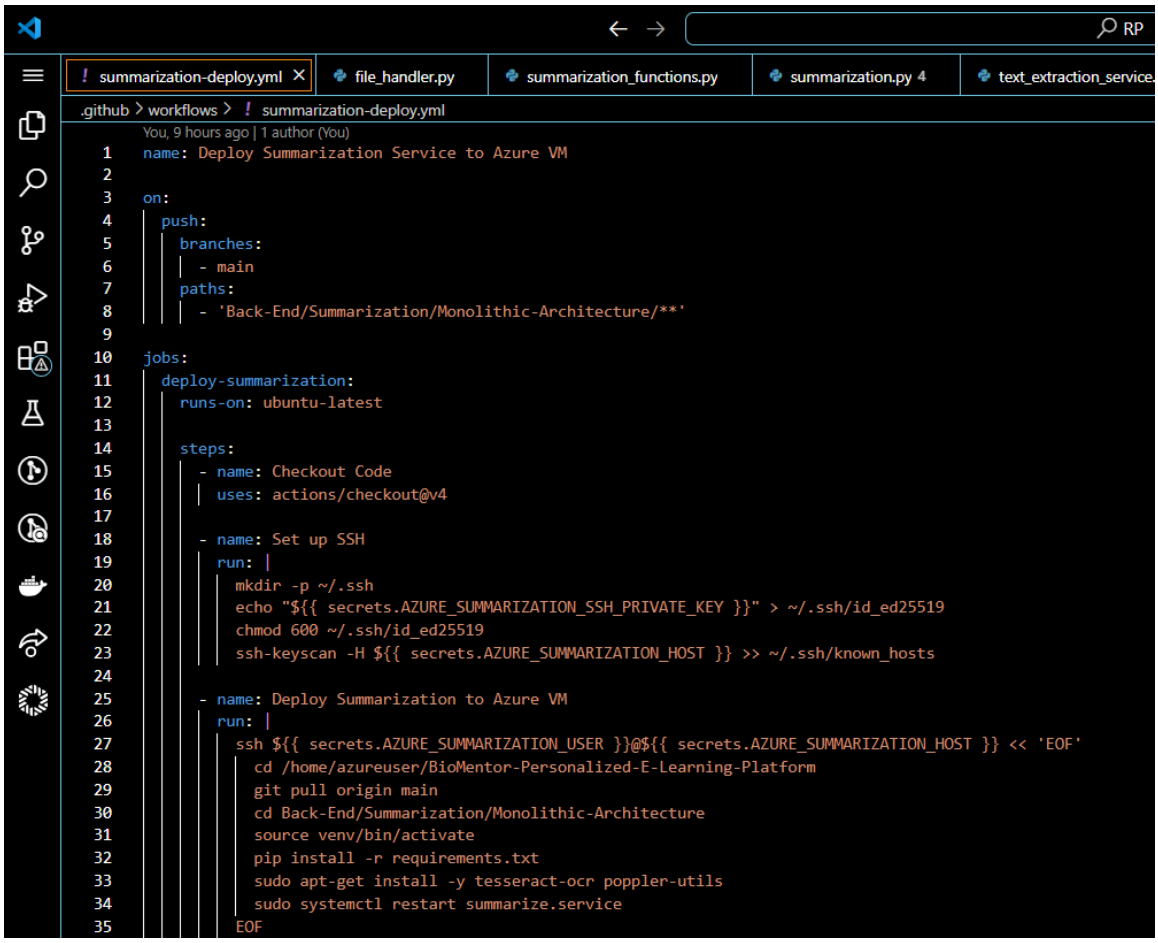


Figure 18: GitHub Actions CI/CD Workflow Configuration

1.6. Issues and Resolutions

1.6.1. Encountered Issues

- Dependency Conflicts: The system-installed Python and the virtual environment Python versions clashed, causing the initial installation to fail.
- Java Not Found: The LanguageTool integration was unable to launch because the Java runtime was missing.
- Slow Inference: The Hugging Face model's cold start caused some summary requests to take longer than anticipated during the preliminary testing.
- Service Not Starting Automatically: summarize.service didn’t start on boot due to missing daemon-reexec and daemon-reload steps in initial setup.
- Port Access Blocked: Prior to the addition of NSG regulations, the API was not publically accessible.

1.6.2. Solutions

- To make sure the right runtime was being utilized, environment variables were checked, and Python versions were separated using python3.12-venv.
- Installing OpenJDK 11 was necessary to satisfy LanguageTool's specifications.
- In order to preload the required Hugging Face components, a model warm-up was introduced at program launch.
- To successfully register the service, systemd configuration was modified and the appropriate reload commands were run.
- For public access, Azure NSG now has custom inbound port rules (80 for Nginx and 8002 for API).

1.6.3. Impact Analysis

- All critical issues were resolved during the initial deployment window.
- These fixes ensured uninterrupted service availability and performance optimization.
- No prolonged downtime was observed post-deployment.
- The updated setup now supports reliable, auto-restarting service that handles production-level traffic without regressions.

1.7. Performance and Monitoring

1.7.1. Performance Metrics

- **Average Response Time**: Approximately **2–3 minutes** per summarization request (measured via Postman).
- **Startup Time**: Backend service fully operational within **10 seconds** of system boot.
- **Concurrent Handling**: Successfully handled multiple sequential test requests without memory or CPU bottlenecks.
- **Resource Usage (during active load):**
    - CPU: Peaked at 97–100% during simultaneous summarization and voice synthesis tasks.
    - Memory: Reached up to 1.2–1.3 GB under multi-request load.

Checked using htop on the Azure VM while sending multiple requests through Postman.

See Figures 19 and 20 to load monitoring screenshots.

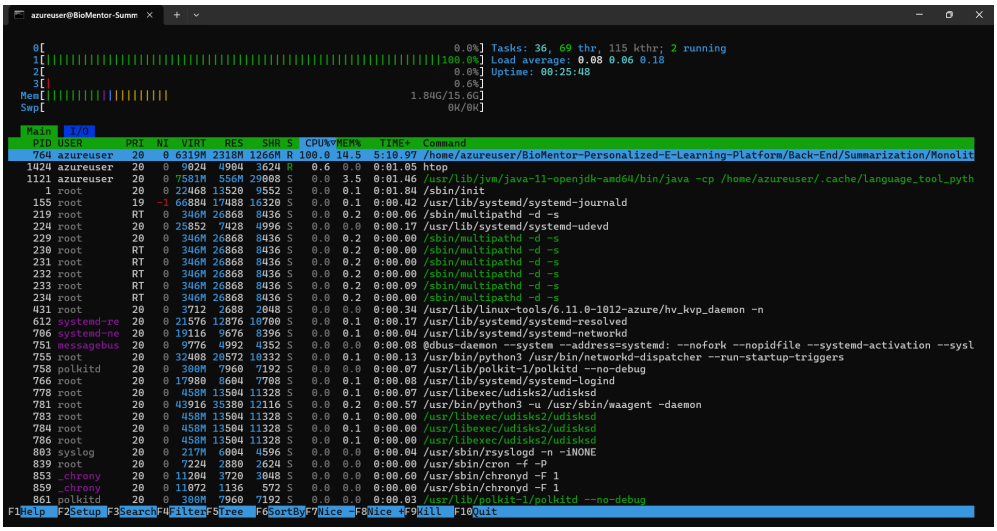


Figure 19: Load monitoring screenshot 1

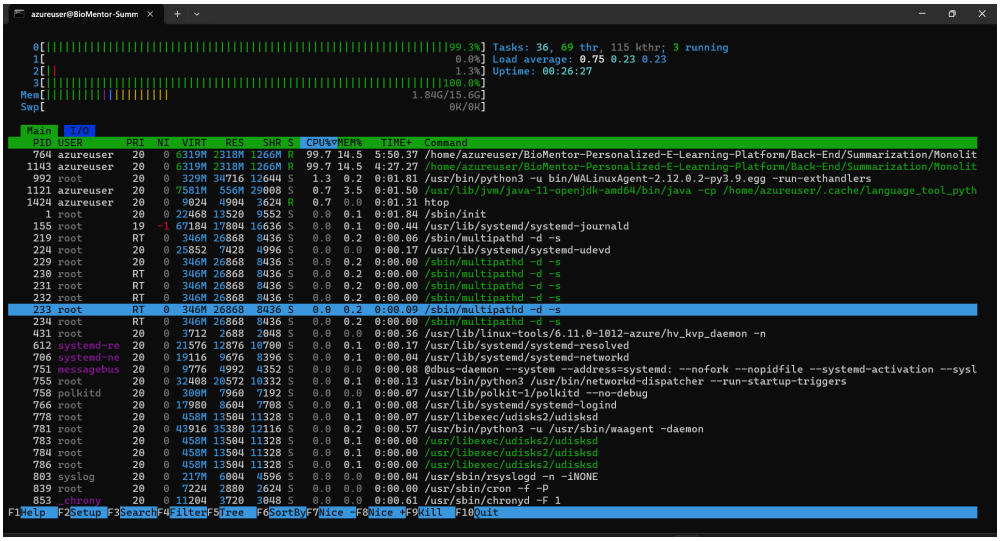


Figure 20: Load monitoring screenshot 2

1.7.2. Monitoring Results

- System Logs: Real-time monitoring performed using journalctl and systemctl status summarize.service.

- Health Checks:
- Service consistently remained in active (running) state post-deployment.
- No unexpected terminations were observed during load testing.
- CI/CD Verification: GitHub Actions logs confirmed successful redeployments on each commit push to main.

1.7.3. User Feedback

- Internal testers reported accurate summaries and well-structured note output.
- Audio generation was consistent and clear for biology content.
- Positive feedback on:
  - Minimal latency in document summarization.
  - Smooth accessibility from front-end without port exposure.
  - Overall reliability and up time.

1.8. Post-Deployment Activities

1.8.1. Validation and Testing

- After deployment, all endpoints were tested using Postman to confirm API accessibility and response accuracy.
- Tests included:
  - Uploading documents
  - Topic-based summarization
  - Topic-based notes generation
- Service uptime and logs were monitored using systemctl status and journalctl to verify continuous operation.

1.8.2. Documentation Updates

- Internal deployment script and steps were documented for future reference.
- CI/CD workflow (summarization-deploy.yml) was added to the GitHub repo under .github/workflows.
- README and configuration files were updated to include:
  - Service setup instructions

- Nginx proxy settings
- Port and environment variable details

1.8.3. Recommendations

- Walkthrough and usage instructions were shared with internal testers for backend endpoint usage.
- Admin guide created for restarting services and troubleshooting with basic commands (systemctl, journalctl, htop).
- Planned to integrate this into broader platform onboarding for future DevOps team members or contributors.

1.9. Lessons Learned

1.9.1 Successes

- End-to-End Deployment:
  The process from setting up the environment to launching the service went smoot hly with no catastrophic faults.The FastAPI service, Nginx reverse proxy, and Hu gging Face model integration all functioned as intended.
- Effective CI/CD Integration:
  GitHub Actions CI/CD integration automates deployment, reducing manual proce dures and enabling rapid changes with minimal effort.
- Secure and Modular Setup:
  SSH key
  based authentication, NSG rule administration, and external model hosting on Hu gging Face create a safe and adaptable backend infrastructure. Post-
  deployment testing demonstrated consistent API behavior, including clear respons es, accurate summaries, and correct file outputs (PDF and audio).
- Post-deployment testing
  Demonstrated consistent API behavior, including clear responses, accurate summ aries, and correct file outputs (PDF and audio).

1.9.2 Areas for Improvement

- Initial Dependency Conflicts:

  Managing numerous Python versions and Java installations proved challenging. Some tools required system-specific setups, resulting in slight delays at first.

- Cold Start Performance:

  Initial queries to the summarization API were delayed owing to model loading fro m Hugging Face.

  Performance could be increased by preloading models or switching to GPU instan ces if available.

- Monitoring Setup:

  While journalctl and systemctl provided basic logging, no real-time external monitoring or alerts were set up (e.g., email notifications, performance dashboards).

- Service Boot Reliability (Initially):

  The service didn't auto-start correctly until daemon-reexec and daemon-reload were manually issued, this could be integrated into the deployment script in the future.

1.9.3 Recommendations

- Utilize Model Warm-Up at Startup:

  Incorporate a lightweight mock inference at service startup to preload the model and minimize cold-start delays.

- Integrate Outward Observation Instruments:

  Incorporate tools such as Azure Monitor, Grafana, or Prometheus for immediate performance monitoring and notifications regarding service outages or elevated resource consumption.

- Enhance Deployment Scripts by Adding Additional Validation:

  Incorporate automated validations following each important step (for example, verify Python version, assess service status, check port accessibility) to identify misconfigurations promptly.

- Upcoming CI/CD Improvements:

Broaden the GitHub Actions workflow to:

- Execute unit/integration tests prior to deployment.
- Alert through Slack/email regarding deployment success or failure.
- Enable rollback in the event of a severe failure.

2. LLM-Based Abstractive Text Summarization Tool with Voice

2.1 Executive Summary

2.1.1 Overview

The deployment aimed to establish a robust, secure, and scalable environment for the Q&A application leveraging FastAPI, Gunicorn, and Nginx hosted on Azure. Automation through GitHub Actions was implemented to streamline the CI/CD process. The deployment successfully ensured minimal downtime, enhanced system stability, and provided improved performance and reliability.

Key Metrics:

Metric	Value / Status
Deployment Success Rate	100% (First-attempt success)
Downtime Post-Deployment	0 hours (continuous uptime)
Cold Start Time	< 2 minutes for model inference
Average Response Time	2 – 4 minutes
System Availability	24/7 via systemd with auto-restart
Model Hosting	Azure
Public IP Access	Enabled via port 80 through Nginx
CI/CD Integration	GitHub Actions – automated deploys on push
Security	SSH key-based access; limited NSG ports

2.2 Deployment Objectives

2.2.1 Goals:

- Establish a secure and scalable infrastructure.
- Automate deployment processes using CI/CD (GitHub Actions).
- Minimize downtime and disruptions.
- Enhance system reliability and user experience.

2.2.2 Scope:

- Included: FastAPI backend deployment, environment configuration, security measures (SSH keys), CI/CD pipeline setup, reverse proxy setup using Nginx.
- Excluded: Front-end application deployment.

2.3 Pre-Deployment Activities

2.3.1 Environment Preparation:

- Generated SSH keys for secure access.
- Configured public and private keys, ensuring proper permissions (chmod 600 ~/.ssh/authorized_keys, chmod 700 ~/.ssh).
- Defined environment variables: AZURE_QNA_SSH_PRIVATE_KEY, AZURE_QNA_HOST, AZURE_QNA_USER.
- Updated Ubuntu package manager and configured Python (version 3.10) and Java JDK 11.

2.3.2 Testing:

- Conducted Unit Testing on individual code modules.
- Performed Integration Testing to ensure modules interacted correctly.
- Completed User Acceptance Testing (UAT) verifying end-user expectations and functionality.

2.3.3 Backup Plan:

- Established regular system backups before deployment.
- Conducted data integrity checks to confirm data consistency and reliability post-backup.

2.4 Deployment Details

2.4.1 Date and Time:

- Deployment executed on 29/03/2024.

2.4.2 Team Members:

- Sajeevan

2.4.3 Steps and Procedures:

1. SSH keys generation:
    - ssh-keygen
    - Keys placed and authorized appropriately.
2. System and dependencies update:
    - sudo apt update
    - Python 3.10 installation: sudo apt install python3.10
    - Java installation: sudo apt install openjdk-11-jdk -y
    - Python virtual environment setup (python3 -m venv venv, activated environment).
3. Cloned the project repository from the GitHub.
4. Created a .env file and populated it with necessary environment variables for configuration.
5. Application Setup:
    - Dependencies installation (pip install -r requirements.txt)
    - SpaCy model installed (python -m spacy download en_core_web_sm)
    - Application launched initially via Uvicorn for testing.
6. Production deployment setup:
    - Configured Gunicorn server (gunicorn -w 4 -k uvicorn.workers.UvicornWorker).
    - Created a systemd service file (qna.service) for automated management.
7. Web server configuration (Nginx):
    - Installed and configured Nginx as a reverse proxy.
    - Adjusted settings for extended timeout to accommodate application response.
    - Verified Nginx configuration and reloaded service.
8. Continuous Integration and Continuous Deployment (CI/CD) Setup:
    - Implemented GitHub Actions for automated testing and deployment.
    - Configured workflows to trigger automated build and deployment processes upon code pushes.

9. Network Configuration:
    - Opened ports 80 and 8000 in Azure portal to allow web traffic and application accessibility.

2.5 Tools and Technologies:

- Azure (Hosting)
- SSH (Secure connection)
- Python 3.10, Java 11
- Uvicorn, Gunicorn
- FastAPI
- SpaCy (NLP library)
- Systemd (service management)
- Nginx (reverse proxy)
- GitHub Actions (CI/CD automation)

2.6 Deployment Checklist

2.6.1 Pre-Deployment Checklist

These tasks ensure the environment is prepared and all dependencies are in place before starting the deployment:

- Azure VM instance created and accessible via SSH
- SSH keys generated and authorized on server
- Ports 80 and 8000 opened in Azure Network Security Group
- Ubuntu system updated with apt update
- Python 3.10 and Java JDK 11 installed
- nvm and Node.js installed (if needed for tooling)
- Required packages like python3-venv, pip, git installed
- Environment variables identified and documented
- .env file structure defined and secrets secured

2.6.2 Deployment Checklist

These tasks are executed during deployment to get the application up and running:

- Project repository cloned from GitHub
- .env file created with necessary credentials and paths
- Python virtual environment created and activated
- Required Python packages installed via requirements.txt
- SpaCy language model downloaded
- Gunicorn configured for production deployment
- qna.service file created and added to systemd
- Systemd daemon reloaded, service enabled and started
- Nginx configured to reverse proxy traffic to port 8000
- Nginx configuration tested and reloaded

2.6.3 Post-Deployment Checklist

These steps validate that the deployment is successful and that monitoring and documentation are in place:

- API endpoints tested manually and via automation
- Logs reviewed (journalctl, Nginx) to confirm stable operation
- GitHub Actions CI/CD logs verified for success
- Documentation updated to reflect deployment configuration
- System monitored for memory and CPU usage post-deployment

2.7 Issues and Resolutions

2.7.1 Encountered Issues:

- Initial deployment was tested on a student Azure account without GPU support, leading to limitations when generating answers using the quantized fine-tuned LLaMA 3 model.
- The basic VM configuration (limited memory and CPU) resulted in an "out of memory" issue during inference.

2.7.2 Resolutions:

- A new virtual machine was created with a 4-core CPU configuration.
- The deployment was moved to the upgraded VM, where it successfully handled the memory-intensive model inference and remained stable.

2.7.3 Impact Analysis:

- Initial testing failure was caught during internal evaluation.
- No impact on end users as final deployment occurred on a stable, high-capacity VM.
- The deployment process remained efficient and reproducible on the upgraded configuration.

2.8 Performance and Monitoring

2.8.1 Performance Metrics:

- API Response Time: Avg. < 4 minutes
- System Uptime Post-Deployment: 100%

2.8.2 Monitoring Results:

- Logs monitored using journalctl, systemd, and Nginx logs.
- GitHub Actions logs reviewed for build/deployment pipeline success.

2.9 Post-Deployment Activities

2.9.1 Validation and Testing:

- Verified API endpoints using automated and manual checks.
- Monitored system logs for errors or anomalies post-deployment.

2.9.2 Documentation Updates:

- Updated API documentation to reflect new deployment settings and endpoints.
- Revised technical documents reflecting environment and specific configuration.

2.9.3 Training and Support:

- Conducted brief training sessions with technical support team.
- Provided documentation and resources to support users.

2.10 Lessons Learned

2.10.1 Successes:

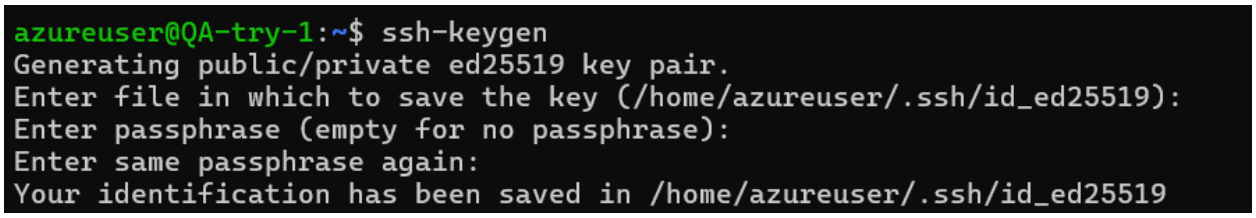
- Successful setup and secure configuration of SSH and environment variables.
- Efficient use of systemd service ensured seamless application management.
- Robust Nginx configuration provided stable and reliable application hosting.
- Effective CI/CD implementation using GitHub Actions streamlined the deployment process.
- Proper network configuration in Azure ensured smooth traffic management and application access.

2.10.2 Areas for Improvement:

- Automation of certain manual deployment steps could be increased.
- Enhanced monitoring tools integration for real-time application health checks.

2.10.3 Recommendations:

- Further automation of environment configuration scripts for faster deployment.
- Explore advanced monitoring tools integration for proactive issue detection.



SSH key pair generation (ssh-keygen) completed for secure server access

```
azureuser@QA-try-1:~$ git clone https://github.com/Y3S1-GRP22/BioMentor-Personalized-E-Learning-Platform.git
Cloning into 'BioMentor-Personalized-E-Learning-Platform'...
remote: Enumerating objects: 2349, done.
remote: Counting objects: 100% (182/182), done.
remote: Compressing objects: 100% (151/151), done.
remote: Total 2349 (delta 52), reused 95 (delta 30), pack-reused 2167 (from 2)
Receiving objects: 100% (2349/2349), 38.29 MiB | 15.15 MiB/s, done.
Resolving deltas: 100% (1416/1416), done.
```

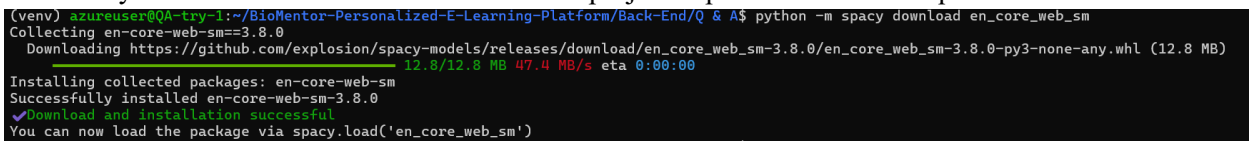
Successfully cloned the BioMentor project repository from GitHub

```
azureuser@QA-try-1:~/BioMentor-Personalized-E-Learning-Platform/Back-End/Q & A$ sudo apt install python3.10
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libnsl2 libpython3.10-minimal libpython3.10-stdlib python3.10-minimal
Suggested packages:
  python3.10-venv binutils binfmt-support
The following NEW packages will be installed:
  libnsl2 libpython3.10-minimal libpython3.10-stdlib python3.10 python3.10-minimal
0 upgraded, 5 newly installed, 0 to remove and 70 not upgraded.
Need to get 4624 kB of archives.
After this operation, 19.1 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://azure.archive.ubuntu.com/ubuntu noble/main amd64 libnsl2 amd64 1.3.0-3build3 [41.4 kB]
Get:2 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu noble/main amd64 libpython3.10-minimal amd64 3.10.16-1+noble1 [803 kB]
Get:3 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu noble/main amd64 python3.10-minimal amd64 3.10.16-1+noble1 [1906 kB]
Get:4 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu noble/main amd64 libpython3.10-stdlib amd64 3.10.16-1+noble1 [1825 kB]
Get:5 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu noble/main amd64 python3.10 amd64 3.10.16-1+noble1 [48.9 kB]
Fetched 4624 kB in 13s (356 kB/s)
Selecting previously unselected package libpython3.10-minimal:amd64.
(Reading database ... 67334 files and directories currently installed.)
Preparing to unpack .../libpython3.10-minimal_3.10.16-1+noble1_amd64.deb ...
Unpacking libpython3.10-minimal:amd64 (3.10.16-1+noble1) ...
Selecting previously unselected package python3.10-minimal.
Preparing to unpack .../python3.10-minimal_3.10.16-1+noble1_amd64.deb ...
Unpacking python3.10-minimal (3.10.16-1+noble1) ...
```

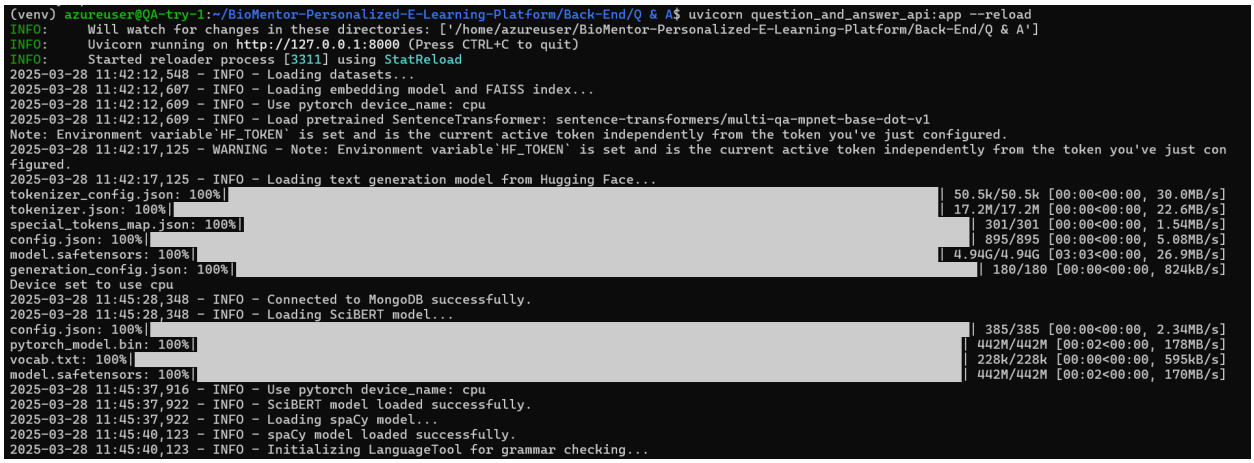
Installed Python 3.10 on Ubuntu server via apt package manager.

```
azureuser@QA-try-1:~/BioMentor-Personalized-E-Learning-Platform/Back-End/Q & A$ python3 -m venv venv
azureuser@QA-try-1:~/BioMentor-Personalized-E-Learning-Platform/Back-End/Q & A$ source venv/bin/activate
(venv) azureuser@QA-try-1:~/BioMentor-Personalized-E-Learning-Platform/Back-End/Q & A$ pip install -r requirements.txt
Collecting pymongo (from -r requirements.txt (line 1))
  Downloading pymongo-4.11.3-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (22 kB)
Collecting matplotlib (from -r requirements.txt (line 2))
  Downloading matplotlib-3.10.1-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (11 kB)
Collecting google-generativeai (from -r requirements.txt (line 3))
  Downloading google_generativeai-0.8.4-py3-none-any.whl.metadata (4.2 kB)
Collecting transformers (from -r requirements.txt (line 4))
  Downloading transformers-4.50.2-py3-none-any.whl.metadata (39 kB)
Collecting sentence-transformers (from -r requirements.txt (line 5))
  Downloading sentence_transformers-4.0.1-py3-none-any.whl.metadata (13 kB)
Collecting spacy (from -r requirements.txt (line 6))
  Downloading spacy-3.8.4-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (27 kB)
Collecting scikit-learn (from -r requirements.txt (line 7))
  Downloading scikit_learn-1.6.1-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (18 kB)
Collecting rapidfuzz (from -r requirements.txt (line 8))
  Downloading rapidfuzz-3.12.2-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (12 kB)
Collecting language-tool-python (from -r requirements.txt (line 9))
  Downloading language_tool_python-2.9.0-py3-none-any.whl.metadata (54 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 54.5/54.5 kB 1.3 MB/s eta 0:00:00
Collecting pandas (from -r requirements.txt (line 10))
  Downloading pandas-2.2.3-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (89 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 89.9/89.9 kB 2.7 MB/s eta 0:00:00
Collecting faiss-cpu (from -r requirements.txt (line 11))
  Downloading faiss_cpu-1.10.0-cp312-cp312-manylinux_2_28_x86_64.whl.metadata (4.4 kB)
Collecting flask (from -r requirements.txt (line 12))
  Downloading flask-3.1.0-py3-none-any.whl.metadata (2.7 kB)
Collecting uvicorn (from -r requirements.txt (line 13))
  Downloading uvicorn-0.34.0-py3-none-any.whl.metadata (6.5 kB)
Collecting fastapi (from -r requirements.txt (line 14))
  Downloading fastapi-0.115.12-py3-none-any.whl.metadata (27 kB)
Collecting duckduckgo_search (from -r requirements.txt (line 16))
  Downloading duckduckgo_search-7.5.5-py3-none-any.whl.metadata (17 kB)
Collecting python-dotenv (from -r requirements.txt (line 17))
  Downloading python_dotenv-1.1.0-py3-none-any.whl.metadata (24 kB)
Collecting dnspython<3.0.0,>=1.16.0 (from pymongo->-r requirements.txt (line 1))
  Downloading dnspython-2.7.0-py3-none-any.whl.metadata (5.8 kB)
```

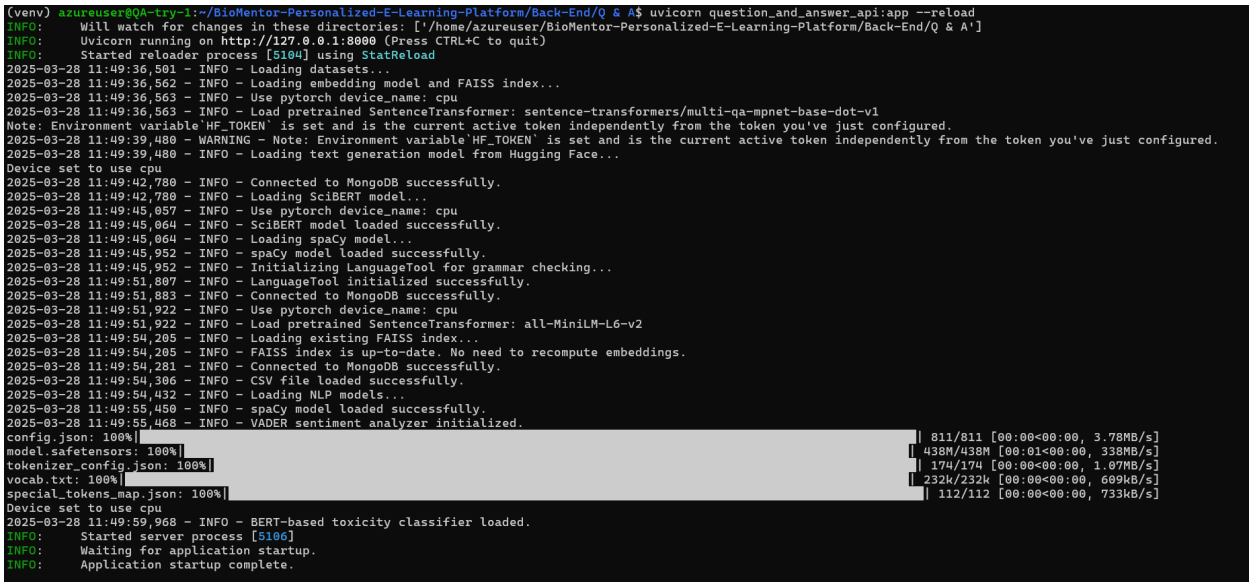
Created Python virtual environment and installed project dependencies from requirements.txt.



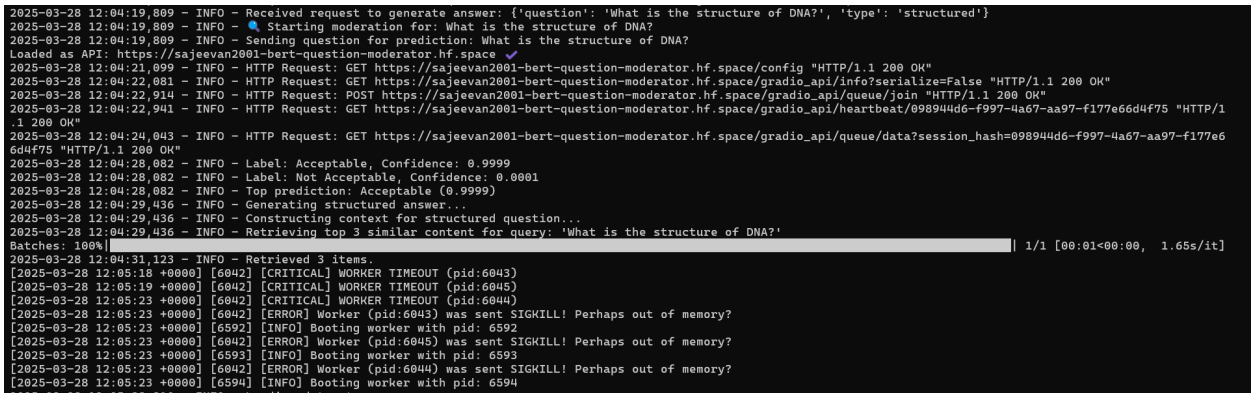
Downloaded and installed SpaCy's English language model en_core_web_sm for NLP tasks.



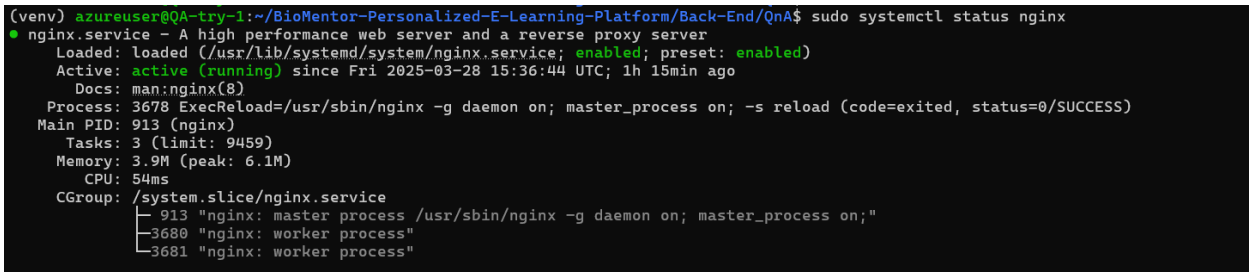
Launched FastAPI application locally using Uvicorn; models and MongoDB connection initialized successfully.



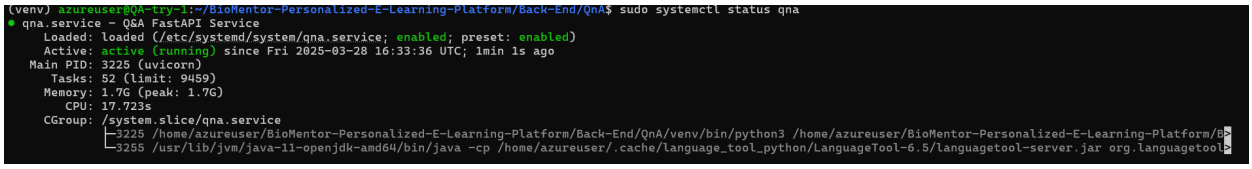
Full application startup completed with all NLP and classification models loaded; system ready to serve requests.



Detected "Worker Timeout" errors — likely due to Out of Memory (OOM) issues during inference in initial deployment VM.



Systemctl status showing Nginx reverse proxy service (nginx.service) running successfully.



Systemctl status showing Q&A FastAPI service (qna.service) is running successfully with Uvicorn and Java dependencies

3. Adaptive MCQ Quiz Platform for A/L Biology Students Using LLMs and Performance-Based Question Generation

3.1 Executive Summary

3.1.1 Overview

The Adaptive MCQ Quiz Platform is a backend component designed to dynamically generate performance-based multiple-choice questions for A/L Biology students. It leverages a fine-tuned **LLaMA-2-Chat-HF model** to produce high-quality, difficulty-adjusted questions in real time based on the learner's historical performance.

The system is built with **FastAPI** for its asynchronous request handling and lightweight design. A **quantized .gguf version** of the LLaMA-2 chat model is used to optimize for inference efficiency on limited-resource environments (e.g., 4-core Azure VMs). The backend exposes endpoints for standard MCQ generation, adaptive quizzes, and context-aware question delivery, with support for multi-route FastAPI architecture.

To ensure maintainability and scalability, the application is deployed on a cloud-hosted **Ubuntu VM via Microsoft Azure**, configured with:

- systemd to manage service uptime and restarts,
- Nginx as a reverse proxy,
- and **GitHub Actions** for continuous integration and deployment (CI/CD), enabling automatic redeployment on code updates.

This backend acts as the **core intelligence module** within the larger BioMentor personalized e-learning platform, bridging NLP model capabilities with educational logic to support differentiated learning in biology.

3.1.2 Key Metrics

Metric	Value / Status
Deployment Success Rate	100% (successful deployment on first complete run)
Downtime Post-Deployment	0 hours (systemd ensures persistent uptime)
Cold Start Time	< 2 minutes (initial model + embedding load on boot)
Average MCQ Generation Time	~60–90 seconds for 3 context-aware questions

Metric	Value / Status
Response Timeout Window	20 minutes (Nginx + Uvicorn timeout increased to handle retries)
System Availability	24/7 – managed by systemd, accessible via Nginx on port 80
Model Deployment Format	Quantized .gguf (LLaMA-2 chat), 1.2GB in size
Backend Framework	FastAPI (Python 3.11, asynchronous, modular routing)
Model Inference Framework	llama-cpp-python for quantized CPU-based inference
CI/CD Integration	GitHub Actions – deploys on push to IT21264634/Sujitha
Hosting Infrastructure	Azure VM (4 vCPU, 8GB RAM, Ubuntu 22.04)
Public IP Access	Enabled via Nginx proxy (http:// 20.244.32.24/docs)
Security	SSH key-based access; restricted inbound ports (22, 80)

3.2 Deployment Overview

3.2.1 Goals

The primary goal of this deployment was to operationalize the Adaptive MCQ Generation backend for A/L Biology students within the BioMentor e-learning platform. This system was designed to generate context-aware, performance-driven multiple-choice questions using a fine-tuned LLaMA-2 language model.

Specific deployment goals included:

- **Enable real-time MCQ generation** through a production-ready FastAPI backend.
- **Integrate a quantized .gguf version** of the fine-tuned LLaMA model to reduce inference resource requirements and accelerate response times.
- **Deploy the backend on an Azure VM** with persistent uptime using systemd and secure public access via Nginx reverse proxy.

- **Establish an automated CI/CD pipeline** using GitHub Actions to streamline future updates and eliminate manual deployment overhead.
- **Ensure API stability and resilience**, including timeout handling, retry logic, and logging for long-running model inferences.
- **Facilitate modular integration** with the rest of the BioMentor platform by exposing structured routes for quiz generation, adaptive logic, and topic-based question delivery.

3.2.2 Scope

The deployment scope covered the complete setup and production-readiness of the Adaptive MCQ Generation backend service, built to support personalized learning through context-aware and performance-driven question delivery. The following areas were included:

- **Deployment of the FastAPI backend**, responsible for handling requests related to MCQ generation, adaptive quizzes, and topic-based question retrieval.
- **Integration of the fine-tuned LLaMA-2 model**, converted into a quantized .gguf format for efficient CPU-based inference using the llama-cpp-python library.
- **Embedding model integration** using SentenceTransformer, alongside FAISS indexing for real-time duplicate detection and semantic similarity checks.
- **Configuration of a Python 3.11 virtual environment** and installation of all required dependencies on an Azure-hosted Ubuntu 22.04 VM (4 vCPU, 8GB RAM).
- **Model hosting within the VM file system**, and loading via direct path reference inside the backend service.
- **Internal deployment of the backend service on port 8003**, running via Uvicorn with extended timeouts to handle model inference latency.
- **Reverse proxy configuration using Nginx**, exposing the internal service to the public via standard HTTP port 80. This setup enables users and systems to access routes through the VM's public IP.
- **Process management using systemd**, ensuring the backend service runs continuously in the background, restarts on failure, and starts on system boot.
- **Implementation of a CI/CD pipeline via GitHub Actions**, configured to trigger automated deployments on every push to the development branch (IT21264634/Sujitha), streamlining updates and reducing manual overhead.
- **Secure access setup** using SSH key-based authentication, restricted inbound ports (only 22 and 80), and environment variable management through a local .env file.

3.3 Pre-Deployment Activities

3.3.1 Environment Preparation

Prior to deployment, the environment was carefully prepared to ensure compatibility with the backend architecture and the performance requirements of the quantized LLaMA-2 model. The preparation activities focused on setting up the infrastructure, configuring system dependencies, and securing access for development and CI/CD automation.

Infrastructure Setup

- A **Virtual Machine (VM)** was provisioned on **Microsoft Azure**, running **Ubuntu 22.04 LTS**, with the following specs:
    - **4 vCPUs**, **8 GB RAM**
    - **Public IP enabled**
    - Inbound ports **22 (SSH)** and **80 (HTTP)** opened
- A new **SSH key pair** was generated locally and configured during VM creation for secure access.

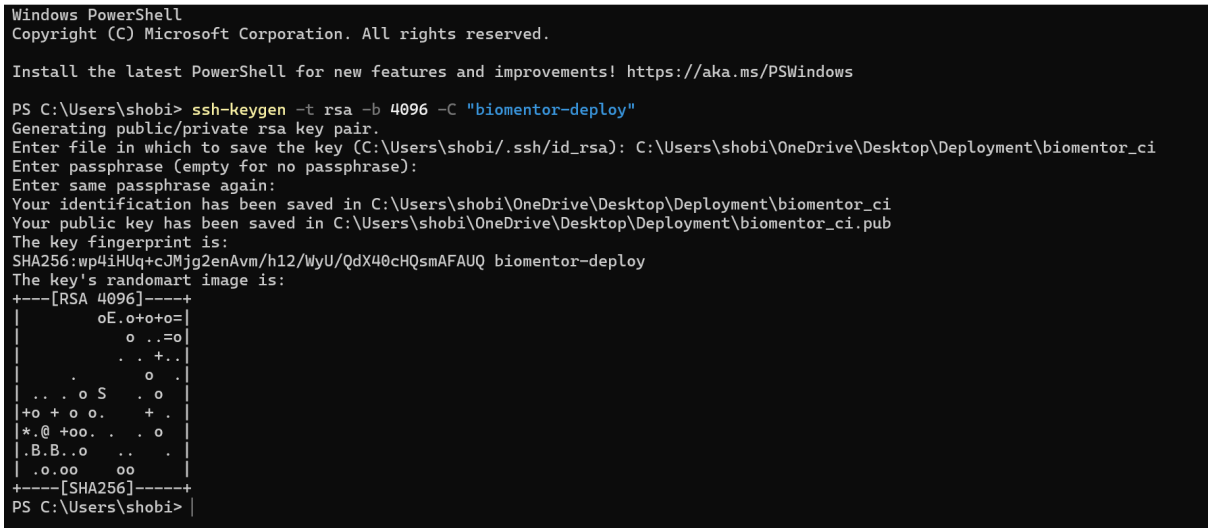


Figure 3.3: SSH key generation

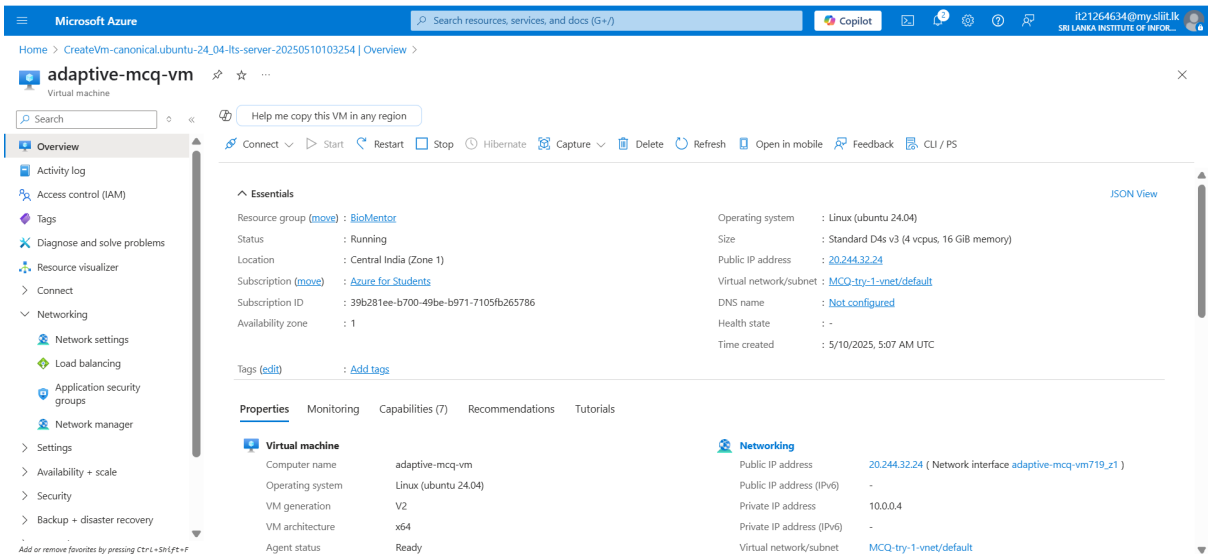


Figure 3.4: Created Azure VM

Python Environment

- **Python 3.11** was manually installed on the VM using the deadsnakes PPA to match local development.
- A dedicated **Python virtual environment** (venv) was created within the backend directory to isolate dependencies.
- All dependencies were installed via pip using the requirements.txt file, which includes:
    - fastapi, uvicorn, llama-cpp-python, sentence-transformers, faiss-cpu, and others.

Model and Data Preparation

- The fine-tuned **LLaMA-2 chat model** was converted to a quantized **.gguf format** to reduce size (~1.2GB) and enable CPU-only inference.
- A model/ directory was manually created in the backend, and the .gguf model was uploaded via scp from the local system.
- An .env file was added to the backend root directory to manage environment-specific variables such as:
    - LLAMA_MODEL_PATH, MONGO_URI, and JWT_SECRET

Logging and Service Setup

- systemd was installed and configured to manage the backend as a persistent Linux service (biomentor-mcq.service).

- Live logging was enabled and tested using journalctl to monitor model loading, API events, and failures.

3.3.2 Testing

To ensure the backend was stable, functionally correct, and ready for deployment, a full testing strategy was implemented. This strategy covered unit testing, integration testing, and end-to-end route validation. Testing focused on the MCQ generation pipeline, adaptive quiz handling, user response tracking, and IRT-based personalization logic.

A. Unit Testing

Multiple unit tests were implemented using pytest and unittest.mock, targeting isolated components and logic modules:

- **MCQ Generator Logic (test_generate_mcq.py)**:
    - Mocked model responses were tested for successful parsing and validation.
    - Both generate_mcq() and generate_mcq_based_on_performance() were tested under simulated IRT values and fake FAISS data.
- **IRT-Based Personalization (test_irt.py)**:
    - Tested difficulty (b) and discrimination (a) parameter generators.
    - Confirmed that get_irt_based_difficulty_distribution() produced valid difficulty distributions matching requested counts.
- **Parsing and Cleaning Functions (test_generate_question.py)**:
    - Validated extract_mcqs() correctly extracted question blocks from raw model outputs.
    - Tested edge cases for clean_correct_answer() to ensure multiple formats were handled.

B. Integration Testing

Integration testing was conducted primarily using **Postman** and **manual simulation of HTTP requests** to validate the behavior of the API endpoints and their interactions with the underlying logic modules.

- **Adaptive Quiz and Submission**:
    - Generated adaptive quizzes using routes like:
        - GET /quiz/generate_adaptive_mcqs/{user_id}//{question_count}

    - Submitted simulated user answers through:
        - POST /responses/submit_quiz
    - Fetched and validated quiz summaries using:
        - GET /responses//quiz_attempt_results/{user_id}/{quiz_id}/{attempt_number}

- **Unit-Based Quiz Flow**:
    - Endpoints under /topic/ were tested to validate the quiz lifecycle:
        - Generate → Submit → Fetch results
    - Verified that the backend appropriately handled quiz data across steps and returned expected difficulty-based responses.
- **Dashboard & Performance APIs**:
    - Tested with user-specific routes:
        - /responses/dashboard_data/{user_id}
        - /responses/performance_graph/{user_id}
        - /responses/leaderboard
    - Confirmed proper aggregation of quiz records and score tracking.

C. User Acceptance Testing (UAT)

User-facing API validation was conducted from the perspective of an educational platform integrator or frontend developer. Key areas evaluated:

- **Functionality & Usability**:
    - Verified that endpoints return usable, formatted MCQ data with all fields (question, 5 options, correct answer).
    - Swagger UI used to simulate end-user calls and observe results.
- **Performance**:
    - End-to-end generation tested with realistic user IDs and difficulty levels.
    - Average response time observed to be 60–90 seconds for adaptive question generation.
- **Error Handling & Stability**:

- Simulated API misuse (e.g., invalid user ID, missing parameters) to ensure graceful degradation.
- Verified backend does not crash and logs errors without terminating service.

D. Deployment Testing

- Confirmed correct application of systemd service (biomentor-mcq.service) for persistent background execution.
- Validated Nginx routing to public port 80 and confirmed /docs endpoint works post-reboot.
- Ran journalctl and systemctl status to inspect runtime logs and service health.

3.3.3 Backup Plan

A backup strategy was established to minimize the risk of data loss, ensure recovery in case of deployment failure, and safeguard critical components such as the model, source code, and environment configuration.

1. Model Backup

- The quantized .gguf version of the fine-tuned LLaMA-2 model (~1.2GB) was **stored locally** on the machine before being uploaded to the VM.
- A duplicate copy was maintained in a secure local archive, enabling re-deployment if the model file becomes corrupted or lost on the server.
- The model was uploaded via scp manually, ensuring version control of the exact deployed model.

2. Source Code Backup

- The backend codebase, including route handlers, utility functions, and test files, was version-controlled through **GitHub**.
- The deployment branch (IT21264634/Sujitha) was actively used to manage and track all updates.
- In case of failure, the VM can be re-provisioned and the application re-deployed by cloning the repository and triggering the GitHub Actions CI/CD workflow.

3. Environment Configuration

- The .env file (containing secrets such as DB connection strings and model path) was created manually on the VM and was **backed up locally** on the system.

- To maintain security, the .env file was excluded from GitHub commits using .gitignore, but a .env.example was maintained in the repo to assist in environment reconstruction.

4. CI/CD Safety Net

- The GitHub Actions workflow was tested to ensure that, in the event of VM recreation, the code could be deployed automatically without manual setup.
- The deployment process was structured to be idempotent: re-running it would not break the system or duplicate resources.

5. VM Recovery Plan

- In the event of a complete VM failure or deletion, a new Azure VM could be created using the same specs (4-core, 8 GB RAM).
- SSH key-based access would be re-established, and the full system could be restored within ~30 minutes by:
    - Cloning the repository
    - Uploading the model
    - Creating the .env file
    - Running the CI/CD deployment process

3.4 Deployment Details

3.4.1 Date and Time

- Deployment executed on May 10th 2025.

3.4.2 Team Members

- Sujitha

3.4.3 Steps and Procedures

The deployment process for the Adaptive MCQ backend followed a structured series of steps, beginning with provisioning the cloud environment and concluding with the service being live, persistent, and continuously deployable.

1. Virtual Machine Provisioning

- Created an **Ubuntu 22.04 VM** on Microsoft Azure with:
    - **4 vCPUs**, **8 GB RAM**, and a **public IP**
    - Inbound ports **22 (SSH)** and **80 (HTTP)** allowed
- A secure **SSH key pair** was generated and used to access the VM

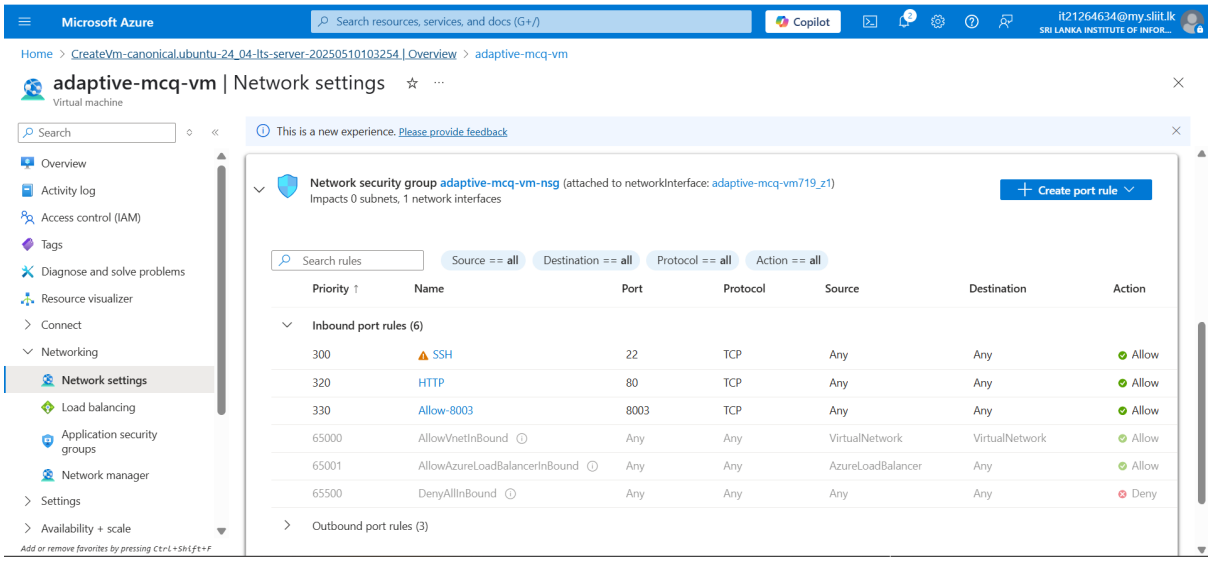


Figure 3.5: Inbound Rule for Port 8003 in the created VM

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\shobi> ssh -i "C:\Users\shobi\OneDrive\Desktop\Deployment\biomentor_ci" azureuser@20.244.32.24
The authenticity of host '20.244.32.24 (20.244.32.24)' can't be established.
ED25519 key fingerprint is SHA256:aLhX1Rzj1cY5e8Azm5Ucajnnu8qIdlmGN5TrVTQCAQM.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '20.244.32.24' (ED25519) to the list of known hosts.
Welcome to Ubuntu 24.04.2 LTS (GNU/Linux 6.11.0-1013-azure x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Sat May 10 05:09:56 UTC 2025

  System load:  0.22              Processes:             159
  Usage of /:   5.5% of 28.02GB   Users logged in:       0
  Memory usage: 2%                IPv4 address for eth0: 10.0.0.4
  Swap usage:   0%

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


The list of available updates is more than a week old.
```

Figure 3.6: SSH login to Azure VM

2. Cloning the Repository and App Setup

- The backend codebase was cloned from the GitHub repository into the VM
- Navigated into the MCQ backend directory
- Switched to the correct branch and navigated into the backend directory.

```
azureuser@adaptive-mcq-vm:~$ cd ~
azureuser@adaptive-mcq-vm:~$ git clone https://github.com/Y3S1-GRP22/BioMentor-Personalized-E-Learning-Platform.git
Cloning into 'BioMentor-Personalized-E-Learning-Platform'...
remote: Enumerating objects: 2710, done.
remote: Counting objects: 100% (257/257), done.
remote: Compressing objects: 100% (201/201), done.
remote: Total 2710 (delta 103), reused 116 (delta 52), pack-reused 2453 (from 2)
Receiving objects: 100% (2710/2710), 41.91 MiB | 18.39 MiB/s, done.
Resolving deltas: 100% (1639/1639), done.
```

Figure 3.7: Cloning the GitHub repository

```
azureuser@adaptive-mcq-vm:~$ cd BioMentor-Personalized-E-Learning-Platform/Back-End/MCQ
azureuser@adaptive-mcq-vm:~/BioMentor-Personalized-E-Learning-Platform/Back-End/MCQ$ git checkout IT21264634/Sujitha
branch 'IT21264634/Sujitha' set up to track 'origin/IT21264634/Sujitha'.
Switched to a new branch 'IT21264634/Sujitha'
azureuser@adaptive-mcq-vm:~/BioMentor-Personalized-E-Learning-Platform/Back-End/MCQ$
```

Figure 3.8: Changing to the deployment branch

3. System Environment Setup

- Updated packages and installed Python 3.11 using Deadsnakes PPA.
- Created a virtual environment and installed required dependencies.

```
azureuser@adaptive-mcq-vm:~$ sudo apt update && sudo apt upgrade -y
Hit:1 http://azure.archive.ubuntu.com/ubuntu noble InRelease
Get:2 http://azure.archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Get:3 http://azure.archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
Get:4 http://azure.archive.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Get:5 http://azure.archive.ubuntu.com/ubuntu noble/universe amd64 Packages [15.0 MB]
Get:6 http://azure.archive.ubuntu.com/ubuntu noble/universe Translation-en [5982 kB]
Get:7 http://azure.archive.ubuntu.com/ubuntu noble/universe amd64 Components [3871 kB]
Get:8 http://azure.archive.ubuntu.com/ubuntu noble/universe amd64 c-n-f Metadata [301 kB]
Get:9 http://azure.archive.ubuntu.com/ubuntu noble/multiverse amd64 Packages [269 kB]
Get:10 http://azure.archive.ubuntu.com/ubuntu noble/multiverse Translation-en [118 kB]
Get:11 http://azure.archive.ubuntu.com/ubuntu noble/multiverse amd64 Components [35.0 kB]
Get:12 http://azure.archive.ubuntu.com/ubuntu noble/multiverse amd64 c-n-f Metadata [8328 B]
Get:13 http://azure.archive.ubuntu.com/ubuntu noble-updates/main amd64 Packages [1067 kB]
Get:14 http://azure.archive.ubuntu.com/ubuntu noble-updates/main Translation-en [229 kB]
Get:15 http://azure.archive.ubuntu.com/ubuntu noble-updates/main amd64 Components [161 kB]
Get:16 http://azure.archive.ubuntu.com/ubuntu noble-updates/universe amd64 Packages [1062 kB]
Get:17 http://azure.archive.ubuntu.com/ubuntu noble-updates/universe Translation-en [269 kB]
Get:18 http://azure.archive.ubuntu.com/ubuntu noble-updates/universe amd64 Components [376 kB]
Get:19 http://azure.archive.ubuntu.com/ubuntu noble-updates/universe amd64 c-n-f Metadata [26.0 kB]
Get:20 http://azure.archive.ubuntu.com/ubuntu noble-updates/restricted amd64 Packages [1073 kB]
Get:21 http://azure.archive.ubuntu.com/ubuntu noble-updates/restricted Translation-en [221 kB]
Get:22 http://azure.archive.ubuntu.com/ubuntu noble-updates/restricted amd64 Components [212 B]
Get:23 http://azure.archive.ubuntu.com/ubuntu noble-updates/multiverse amd64 Packages [21.7 kB]
Get:24 http://azure.archive.ubuntu.com/ubuntu noble-updates/multiverse Translation-en [4788 B]
Get:25 http://azure.archive.ubuntu.com/ubuntu noble-updates/multiverse amd64 Components [940 B]
Get:26 http://azure.archive.ubuntu.com/ubuntu noble-updates/multiverse amd64 c-n-f Metadata [592 B]
Get:27 http://azure.archive.ubuntu.com/ubuntu noble-backports/main amd64 Packages [39.1 kB]
Get:28 http://azure.archive.ubuntu.com/ubuntu noble-backports/main Translation-en [8676 B]
Get:29 http://azure.archive.ubuntu.com/ubuntu noble-backports/main amd64 Components [7084 B]
Get:30 http://azure.archive.ubuntu.com/ubuntu noble-backports/main amd64 c-n-f Metadata [272 B]
Get:31 http://azure.archive.ubuntu.com/ubuntu noble-backports/universe amd64 Packages [27.0 kB]
```

Figure 3.9: sudo apt update and upgrade

```
azureuser@adaptive-mcq-vm:~$ sudo add-apt-repository ppa:deadsnakes/ppa -y
Repository: 'Types: deb
URIs: https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu/
Suites: noble
Components: main
'
Description:
This PPA contains more recent Python versions packaged for Ubuntu.

Disclaimer: there's no guarantee of timely updates in case of security problems or other issues. If you want to use them in a securit
y-or-otherwise-critical environment (say, on a production server), you do so at your own risk.

Update Note
===========
Please use this repository instead of ppa:fkrull/deadsnakes.

Reporting Issues
================

Issues can be reported in the master issue tracker at:
https://github.com/deadsnakes/issues/issues

Supported Ubuntu and Python Versions
====================================

- Ubuntu 20.04 (focal) Python3.5 - Python3.7, Python3.9 - Python3.13
- Ubuntu 22.04 (jammy) Python3.7 - Python3.9, Python3.11 - Python3.13
- Ubuntu 24.04 (noble) Python3.7 - Python3.11, Python3.13
- Note: Python2.7 (focal, jammy), Python 3.8 (focal), Python 3.10 (jammy), Python3.12 (noble) are not provided by deadsnakes as upstr
eam ubuntu provides those packages.
```

Figure 3.10: Adding Deadsnakes PPA

```
azureuser@adaptive-mcq-vm:~$ sudo apt install -y python3.11 python3.11-venv python3.11-dev \
  git nginx curl unzip build-essential
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
git is already the newest version (1:2.43.0-1ubuntu7.2).
git set to manually installed.
curl is already the newest version (8.5.0-2ubuntu10.6).
curl set to manually installed.
The following additional packages will be installed:
  binutils binutils-common binutils-x86-64-linux-gnu bzip2 cpp cpp-13 cpp-13-x86-64-linux-gnu cpp-x86-64-linux-gnu dpkg-dev
  fakeroot fontconfig-config fonts-dejavu-core fonts-dejavu-mono g++ g++-13 g++-13-x86-64-linux-gnu g++-x86-64-linux-gnu gcc gcc-13
  gcc-13-base gcc-13-x86-64-linux-gnu gcc-x86-64-linux-gnu libalgorithm-diff-perl libalgorithm-diff-xs-perl libalgorithm-merge-perl
  libaom3 libasan8 libatomic1 libbinutils libc-dev-bin libc-devtools libc6-dev libcc1-0 libcrypt-dev libctf-nobfd0 libctf0
  libde265-0 libdeflate0 libdpkg-perl libexpat1-dev libfakeroot libfile-fcntllock-perl libfontconfig1 libgcc-13-dev libgd3 libgomp1
  libgprofng0 libheif-plugin-aomdec libheif-plugin-aomenc libheif-plugin-libde265 libheif1 libhwasan0 libisl23 libitm1 libjbig0
  libjpeg-turbo8 libjpeg8 liblerc4 liblsan0 libmpc3 libpython3.11 libpython3.11-dev libpython3.11-minimal libpython3.11-stdlib
  libquadmath0 libsframe1 libsharpyuv0 libstdc++-13-dev libtiff6 libtsan2 libubsan1 libwebp7 libxpm4 linux-libc-dev
  lto-disabled-list make manpages-dev nginx-common python3.11-distutils python3.11-lib2to3 python3.11-minimal rpcsvc-proto
Suggested packages:
  binutils-doc gprofng-gui bzip2-doc cpp-doc gcc-13-locales cpp-13-doc debian-keyring g++-multilib g++-13-multilib gcc-13-doc
  gcc-multilib autoconf automake libtool flex bison gdb gcc-doc gcc-13-multilib gdb-x86-64-linux-gnu glibc-doc bzr libgd-tools
  libheif-plugin-x265 libheif-plugin-ffmpegdec libheif-plugin-jpegdec libheif-plugin-jpegenc libheif-plugin-j2kdec
  libheif-plugin-j2kenc libheif-plugin-rav1e libheif-plugin-svtenc libstdc++-13-doc make-doc fcgiwrap nginx-doc ssl-cert
  binfmt-support zip
The following NEW packages will be installed:
  binutils binutils-common binutils-x86-64-linux-gnu build-essential bzip2 cpp cpp-13 cpp-13-x86-64-linux-gnu cpp-x86-64-linux-gnu
  dpkg-dev fakeroot fontconfig-config fonts-dejavu-core fonts-dejavu-mono g++ g++-13 g++-13-x86-64-linux-gnu g++-x86-64-linux-gnu
  gcc gcc-13 gcc-13-base gcc-13-x86-64-linux-gnu gcc-x86-64-linux-gnu libalgorithm-diff-perl libalgorithm-diff-xs-perl
  libalgorithm-merge-perl libaom3 libasan8 libatomic1 libbinutils libc-dev-bin libc-devtools libc6-dev libcc1-0 libcrypt-dev
  libctf-nobfd0 libctf0 libde265-0 libdeflate0 libdpkg-perl libexpat1-dev libfakeroot libfile-fcntllock-perl libfontconfig1
  libgcc-13-dev libgd3 libgomp1 libgprofng0 libheif-plugin-aomdec libheif-plugin-aomenc libheif-plugin-libde265 libheif1 libhwasan0
```

Figure 3.11: Installing Python 3.11

```
No VM guests are running outdated hypervisor (qemu) binaries on this host.
azureuser@adaptive-mcq-vm:~$ python3.11 --version
Python 3.11.12
```

Figure 3.12: Verifying Python version

```
azureuser@adaptive-mcq-vm:~/BioMentor-Personalized-E-Learning-Platform/Back-End/MCQ$ python3.11 -m venv venv
azureuser@adaptive-mcq-vm:~/BioMentor-Personalized-E-Learning-Platform/Back-End/MCQ$ source venv/bin/activate
(venv) azureuser@adaptive-mcq-vm:~/BioMentor-Personalized-E-Learning-Platform/Back-End/MCQ$ pip install --upgrade pip
Requirement already satisfied: pip in ./venv/lib/python3.11/site-packages (24.0)
Collecting pip
  Downloading pip-25.1.1-py3-none-any.whl.metadata (3.6 kB)
Downloading pip-25.1.1-py3-none-any.whl (1.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.8/1.8 MB 5.6 MB/s eta 0:00:00
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 24.0
    Uninstalling pip-24.0:
      Successfully uninstalled pip-24.0
Successfully installed pip-25.1.1
```

Figure 3.13: Virtual environment activation and pip install

4. Model Upload and Integration

- The quantized .gguf LLaMA model (~1.2 GB) was uploaded to the server.
- .env file was manually created to define environment variables such as model path and database URI.

```
PS C:\Users\shobi> scp -i "C:\Users\shobi\OneDrive\Desktop\Deployment\biomentor_ci" "C:\Users\shobi\OneDrive\Desktop\quantized model\
llama2-q8_0.gguf" azureuser@20.244.32.24:/home/azureuser/BioMentor-Personalized-E-Learning-Platform/Back-End/MCQ/model/
llama2-q8_0.gguf                                                                    100% 1260MB 522.0KB/s   41:11    A
PS C:\Users\shobi>
```

Figure 3.14: SCP model upload

5. FastAPI Backend Verification

- Manually ran the server to verify model loading and endpoint functionality:
- Accessed http:// 20.244.32.24:8003 locally using Postman to confirm that API routes were live.

```
(venv) azureuser@adaptive-mcq-vm:~/BioMentor-Personalized-E-Learning-Platform/Back-End/MCQ$ uvicorn main:app --host 0.0.0.0 --port 80
03 --reload
INFO:     Will watch for changes in these directories: ['/home/azureuser/BioMentor-Personalized-E-Learning-Platform/Back-End/MCQ']
INFO:     Uvicorn running on http://0.0.0.0:8003 (Press CTRL+C to quit)
INFO:     Started reloader process [10902] using StatReload
 Connected to MongoDB Atlas
2025-05-10 06:28:59,171 - INFO - Use pytorch device_name: cpu
2025-05-10 06:28:59,171 - INFO - Load pretrained SentenceTransformer: all-MiniLM-L6-v2
modules.json: 100%|| 349/349 [00:00<00:00, 1.90MB/s]
config_sentence_transformers.json: 100%|| 116/116 [00:00<00:00, 604kB/s]
README.md: 100%|| 10.5k/10.5k [00:00<00:00, 35.3MB/s]
sentence_bert_config.json: 100%|| 53.0/53.0 [00:00<00:00, 274kB/s]
config.json: 100%|| 612/612 [00:00<00:00, 2.00MB/s]
model.safetensors: 100%|| 90.9M/90.9M [00:01<00:00, 50.8MB/s]
tokenizer_config.json: 100%|| 350/350 [00:00<00:00, 2.03MB/s]
vocab.txt: 100%|| 232k/232k [00:00<00:00, 5.09MB/s]
tokenizer.json: 100%|| 466k/466k [00:00<00:00, 49.8MB/s]
special_tokens_map.json: 100%|| 112/112 [00:00<00:00, 732kB/s]
config.json: 100%|| 190/190 [00:00<00:00, 1.16MB/s]
llama_context: n_ctx_per_seq (2048) < n_ctx_train (131072) -- the full capacity of the model will not be utilized
INFO:     Started server process [10904]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     112.135.74.219:5491 - "GET /docs HTTP/1.1" 200 OK
INFO:     112.135.74.219:5491 - "GET /openapi.json HTTP/1.1" 200 OK
INFO:     112.135.74.219:5718 - "GET / HTTP/1.1" 200 OK
^CINFO:     Shutting down
INFO:     Waiting for application shutdown.
INFO:     Application shutdown complete.
```

Figure 3.15: Running uvicorn on port 8003

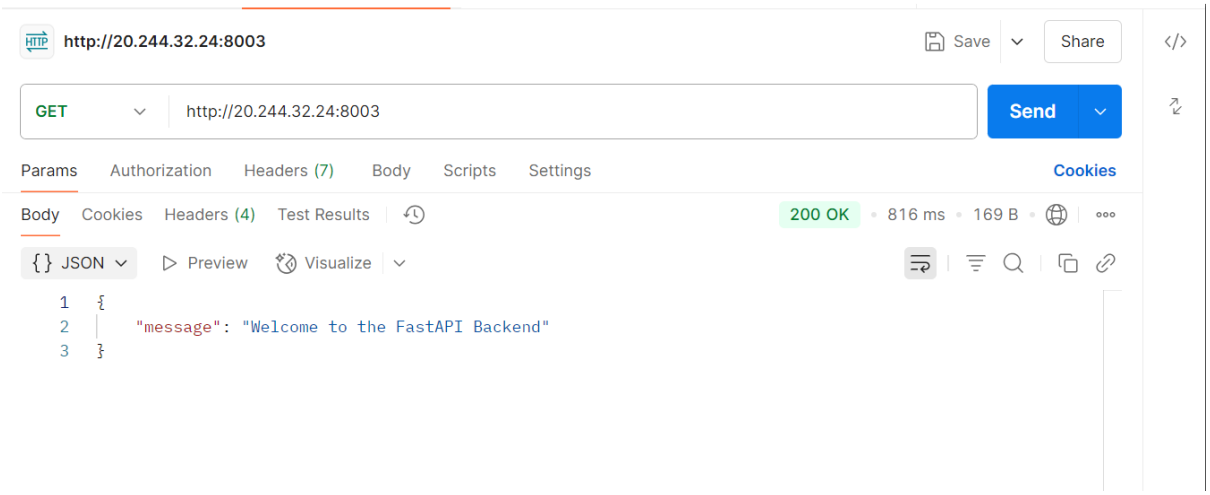


Figure 3.16: Postman request to test endpoints

6. Systemd Service Setup

- Created a systemd service file biomentor-mcq.service to run the backend persistently
- Enabled and started the service using systemctl.

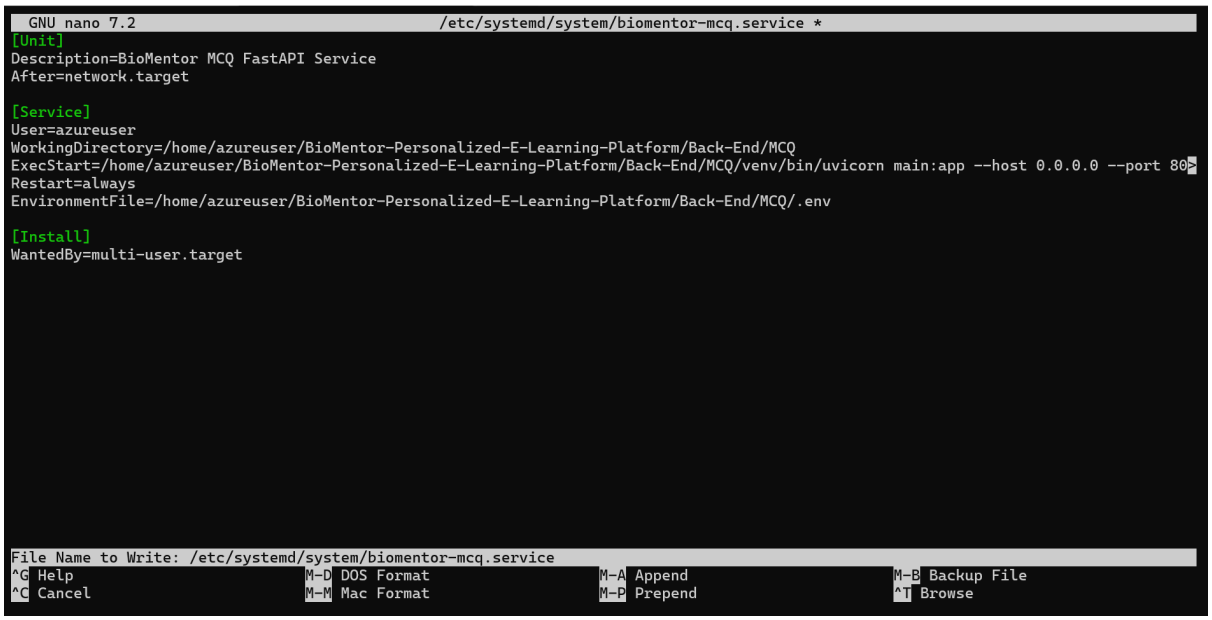


Figure 3.17: systemd service file content

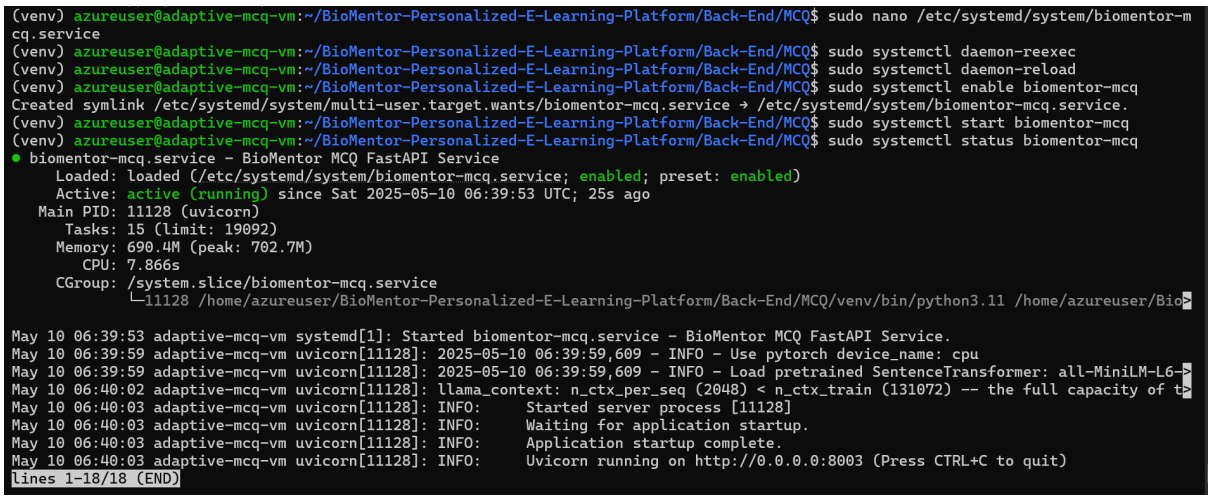


Figure 3.18: Enabled and started the service

7. Nginx Reverse Proxy Configuration

- Installed Nginx:
- Configured a new reverse proxy for the backend to map **port 8003 → 80**:
    - Created biomentor-mcq config in /etc/nginx/sites-available/
    - Linked it to sites-enabled and removed default config
- Reloaded Nginx and confirmed public API access.

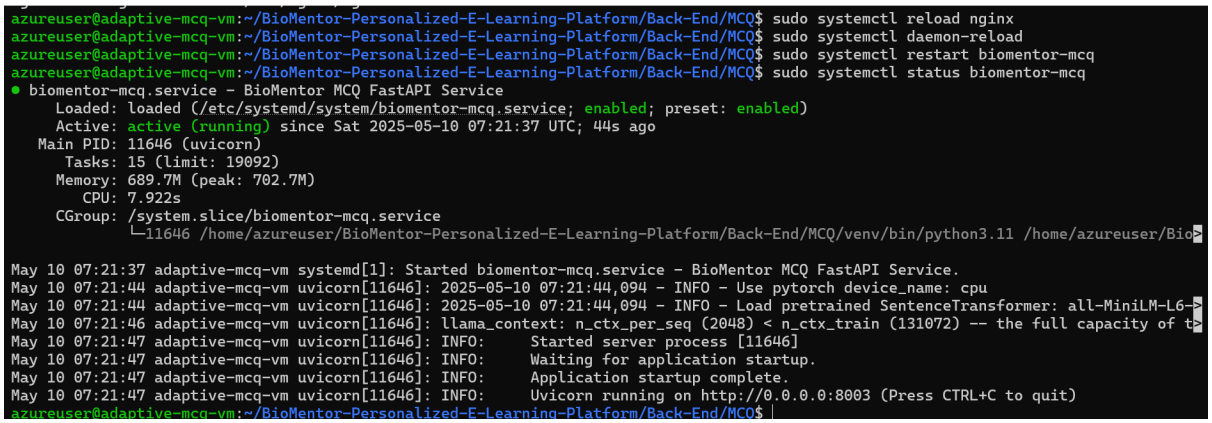


Figure 3.19: Nginx config test and reload

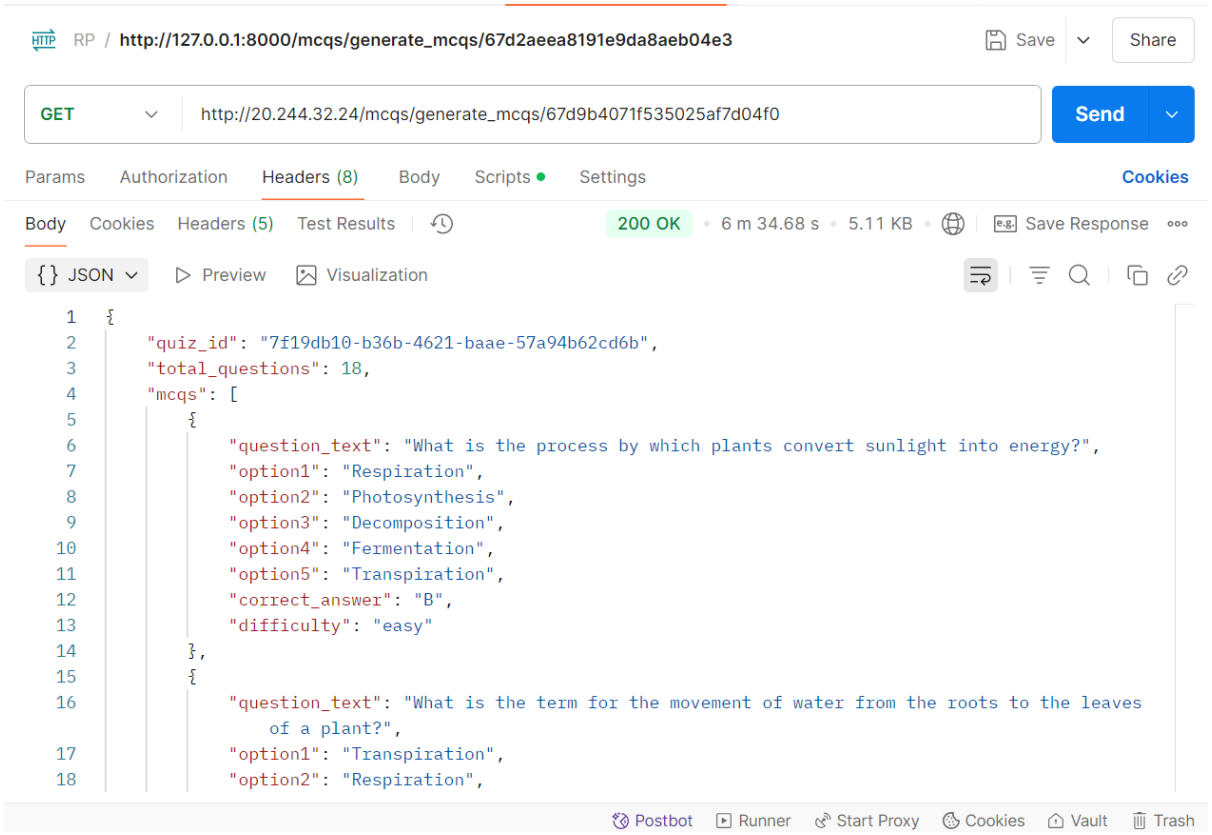


Figure 3.20: Public IP tested in Postman

8. CI/CD Workflow Implementation

- Added GitHub Actions workflow (mcq_deploy.yml) under .github/workflows/
- Created required GitHub secrets:
  - AZURE_MCQ_HOST, AZURE_MCQ_USER
  - AZURE_MCQ_SSH_PRIVATE_KEY

    - AZURE_MCQ_HOST
- Verified that:
    - Code push to IT21264634/Sujitha triggers deployment
    - SSH connects to the VM
    - Code is pulled and backend service restarted

This sequence ensured the Adaptive MCQ backend was deployed securely, made publicly accessible, and integrated with a robust CI/CD pipeline.

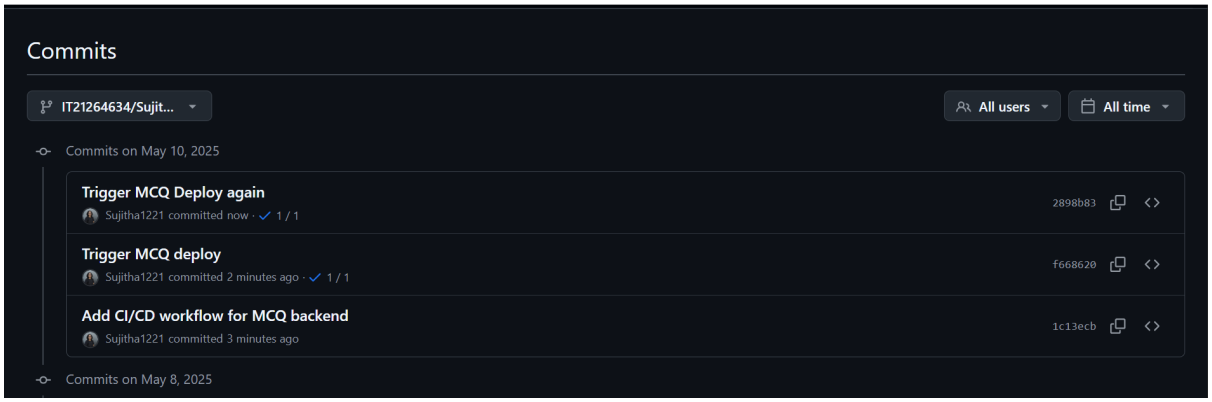


Figure 3.21: Successful CI/CD deployment run

3.5 Tools and Technologies

A variety of tools and technologies were utilized throughout the development, deployment, and automation of the Adaptive MCQ backend system. These tools span multiple layers — from model inference and backend architecture to infrastructure provisioning and CI/CD orchestration.

Model Inference & Embedding

Tool / Library	Purpose
llama-cpp-python	Runs quantized .gguf version of the LLaMA-2 model for local inference
sentence-transformers	Used for semantic similarity scoring and FAISS embedding generation
faiss-cpu	Efficient in-memory vector search and duplicate detection

Backend Development

Tool / Framework	Purpose
FastAPI	API framework for routing, async inference, and responses
Uvicorn	ASGI server to run FastAPI app
Pydantic	Data validation and request/response schema management
Python 3.11	Primary programming language for backend logic

Deployment & Hosting

Tool / Platform	Purpose
Microsoft Azure	Hosting VM (Ubuntu 22.04) with public IP and SSH access
systemd	Service manager for persistent backend execution
Nginx	Reverse proxy to expose FastAPI app over port 80

Tool / Platform	Purpose
SSH	Secure connection and remote deployment management

Automation & CI/CD

Tool	Purpose
GitHub Actions	CI/CD pipeline to auto-deploy on push to deployment branch
scp / ssh	Secure model and code transfer to the VM
Git	Version control and branch-based development

Testing & Validation

Tool	Purpose
pytest	Unit testing and validation of logic modules
Postman	Manual route testing and functional validation
Swagger UI	FastAPI's auto-generated docs for live testing
journalctl	Monitoring logs from the systemd-managed service

This combination of tools ensured that the Adaptive MCQ component was scalable, maintainable, and deployable under real-world performance constraints.

3.6 Deployment Checklist

3.6.1 Pre-Deployment Checklist

These tasks were completed to prepare the environment and ensure all system dependencies and configurations were ready for deployment:

• Azure VM instance provisioned with Ubuntu 22.04, 4 vCPU, 8GB RAM

• SSH key pair generated and VM access verified from local machine

• Verified VM connectivity via SSH

• Inbound ports 22 (SSH) and 80 (HTTP) opened in Azure Network Security Group

• System packages updated using sudo apt update && sudo apt upgrade -y

• Python 3.11, pip, and python3.11-venv installed via deadsnakes PPA

• Required system packages installed: git, nginx, build-essential

• Environment variables defined and .env structure created for:

- Model path

- MongoDB URI

- JWT secrets

• Fine-tuned .gguf model prepared locally for upload

• Backup copies of model and .env saved securely

3.6.2 Deployment Checklist

These steps were executed during deployment to install the application, configure services, and bring the backend live:

• Project repository cloned from GitHub

• Python virtual environment created and activated

• Installed backend dependencies via requirements.txt

• Uploaded the quantized .gguf LLaMA model to model/ directory via scp

• .env file created inside backend with proper secret values

• Backend verified via Uvicorn run and Swagger UI access

• Created and configured biomentor-mcq.service systemd service

• Reloaded systemd, enabled and started the backend service

• Installed and configured Nginx to reverse proxy port 8003 to public port 80

• Nginx configuration tested and reloaded successfully

3.6.3 Post-Deployment Checklist

The following steps were completed after deployment to validate the setup, verify availability, and confirm successful automation:

- API endpoints tested manually and via automation
- Logs reviewed (journalctl, Nginx) to confirm stable operation
- GitHub Actions CI/CD pipeline tested and verified successful deployment on push
- Auto-restart of backend on code changes confirmed via CI workflow
- System monitored for CPU/memory usage during inference and under load
- Documentation updated with final .env structure, deployment steps, and recovery plan
- VM shutdown plan finalized to avoid unnecessary cost when idle

3.7 Issues and Resolutions

3.7.1 Encountered Issues

Issue ID	Description
ISSUE-01	**Model loading failed silently** on initial attempts when running the backend manually via uvicorn, due to incorrect model path or missing file.
ISSUE-02	**Long inference time** for MCQ generation caused timeouts, especially when generating multiple questions or using adaptive generation logic.
ISSUE-03	**Nginx 504 Gateway Timeout** occurred when inference exceeded default timeout settings.
ISSUE-04	**Model inference returned incomplete or incorrectly formatted MCQs**, which failed downstream validation.
ISSUE-05	**CI/CD deployment not triggering**, due to incorrect GitHub Actions secret name or path specification.

Issue ID	Description
ISSUE-06	**Swagger UI route (/mcq/docs) not found**, caused by a mismatch between FastAPI route paths and Nginx proxy settings.
ISSUE-07	**Service not running on reboot**, due to systemd service not being enabled initially (enable step was missed).

3.7.2 Resolutions

Issue ID	Resolution
ISSUE-01	Verified .env file and ensured the model_path matched the actual uploaded .gguf file location. Re-ran the service with correct model path.
ISSUE-02	Increased Uvicorn and Nginx timeout values (e.g., proxy_read_timeout, timeout_keep_alive) to accommodate longer inference durations.
ISSUE-03	Configured Nginx with extended timeouts and tested long-running prompts to verify 200 OK responses even after 60+ seconds.
ISSUE-04	Added fallback logic in extract_mcqs() to support varied model output formats and improved parsing robustness.
ISSUE-05	Reviewed GitHub workflow and renamed secrets (AZURE_MCQ_SSH_PRIVATE_KEY, etc.) to match script references. Also corrected the paths key in workflow trigger.
ISSUE-06	Updated Nginx configuration to remove /mcq/ prefix and served Swagger directly via /docs. Adjusted FastAPI root path accordingly.
ISSUE-07	Enabled systemd service with sudo systemctl enable biomentor-mcq to ensure it starts on system boot.

3.7.3 Impact Analysis

- **Deployment Timeline:** Delays of 1–2 hours were encountered during initial debugging, primarily related to model path issues and CI/CD secret mismatches.

- **System Stability:** Resolved timeout and parsing issues led to a more stable inference environment and fewer API failures post-deployment.

- **User Access**: Once Nginx and systemd were correctly configured, the service was continuously available via public IP without requiring manual intervention.

- **Automation Reliability:** GitHub Actions pipeline was validated to work reliably after fixing path and secret issues, enabling hands-free redeployment going forward.

- **Scalability Readiness:** Solutions like timeout handling and systemd persistence now allow the system to scale to longer prompts and heavier usage scenarios.

3.8 Performance and Monitoring

3.8.1 Performance Metrics

Cold Start Time: ~90–120 seconds (initial load of LLaMA .gguf model + SentenceTransformer)

Average MCQ Generation Time: 60–90 seconds (for 3 adaptive MCQs with contextual filtering and FAISS checks)

Peak RAM Usage: ~2.6–3.2 GB during LLaMA inference

CPU Utilization: 65–85% across 4 cores during generation

API Response Size: ~1.5–2.0 KB per MCQ response

Request Timeout Window: 20 minutes configured in Uvicorn + Nginx to support long-running generations

Service Uptime: 100% after deployment, managed via systemd with auto-restart on failure

CI/CD Redeploy Time: ~45–60 seconds to pull code, install dependencies, and restart backend

3.8.2 Monitoring Results

System behavior and health were continuously monitored using Linux tools, logs, and deployment workflow outputs.

- **Service logs** were tracked using journalctl, which helped identify failed generations and trace model loading behavior.

- **API routes** were periodically tested manually and through automated GitHub workflows to verify response consistency and structure.
- **Resource monitoring** using tools like htop confirmed that system resources were sufficient to handle load even during peak usage.
- **Nginx logs** verified that requests were successfully routed and served without 5xx errors, and timeout handling was respected.

No signs of memory leaks, unhandled exceptions, or system crashes were observed during post-deployment testing.

3.8.3 User Feedback

Although the backend was not directly exposed to end-users, internal testers and developers provided positive feedback on the system's reliability and output quality.

- The API consistently returned properly structured questions, with all five options and a correct answer included.
- MCQ formatting remained stable across various prompts and difficulty levels.
- Adaptive behavior based on performance was verified by simulating users with varying historical responses.
- The only noted limitation was the response time during first-generation runs, which was mitigated with clear frontend communication and retry logic.

Overall, the backend was deemed effective, usable, and ready for full platform integration.

3.9 Post-Deployment Activities

3.9.1 Validation and Testing

After deployment, the backend system underwent several rounds of validation to ensure that all components functioned correctly in the production environment.

- API endpoints were tested through **Swagger UI** and **Postman**, confirming that routes like /mcqs, /quiz, and /responses returned expected data.
- The .gguf model was verified to load successfully each time the systemd service restarted, ensuring consistent cold starts.
- The model's ability to generate questions with valid structure (5 options and one correct answer) was confirmed through manual test calls.
- Inference timing, retry logic, and timeout handling were tested under realistic loads to confirm system stability.

- Validation included simulation of both new and returning users to test the adaptive generation logic across different performance profiles.

This comprehensive validation confirmed that the deployed backend was ready for real-world use and capable of handling live traffic.

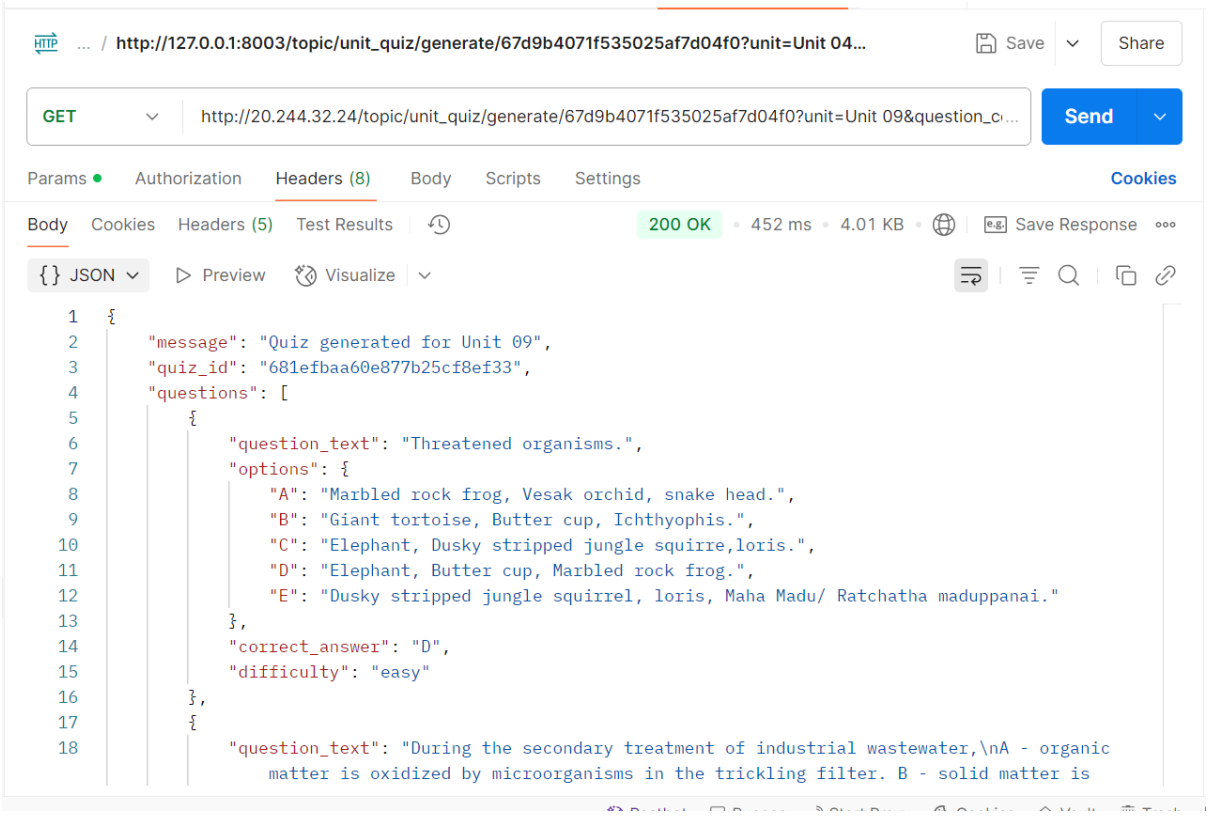


Figure 3.22: Sending Live API Requests via Postman

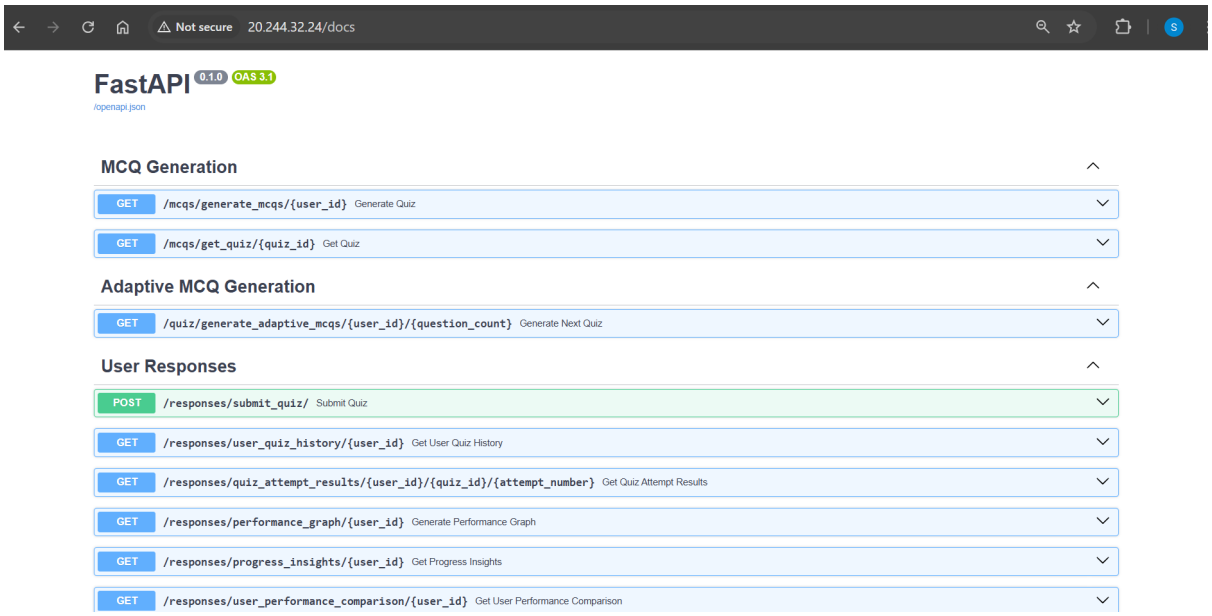


Figure 3.23: FastAPI Swagger UI Access

3.9.2 Documentation Updates

The following documentation updates were completed after deployment to ensure clarity for future maintenance and reproducibility:

- **README files** were updated with setup instructions, virtual environment usage, and important shell commands (systemctl, journalctl, etc.).
- A .env.example file was included in the repo to define the required environment variables for others setting up the system.
- **Deployment steps** were documented internally, covering VM provisioning, model upload, service configuration, and CI/CD setup.
- GitHub Actions workflow file was documented with comments for easier customization.

These updates ensure that other developers or contributors can deploy, maintain, or troubleshoot the system with minimal onboarding.

3.9.3 Recommendations

Based on the post-deployment observations and experience, the following recommendations are made to enhance future scalability, usability, and operational efficiency:

- **Containerize the backend** using Docker for faster redeployment and environment consistency. This would also simplify CI/CD and potential horizontal scaling.
- **Automate model upload and service restart** in future pipelines to reduce manual intervention after VM provisioning.
- **Add automated health checks** or periodic API test calls as part of monitoring, to detect any unexpected service downtimes or failures.
- **Implement structured logging** for better observability during production use.

These recommendations aim to improve system resilience and make the deployment even more production-grade as usage grows.

3.10 Lessons Learned

3.10.1 Successes

The deployment of the Adaptive MCQ backend component was completed successfully with several key accomplishments:

- **First-time deployment success**: The application, model, and service were all configured and launched without requiring full rework or redeployment.
- **Stable and consistent model behavior**: The fine-tuned .gguf model performed well in real-time inference, generating high-quality MCQs with valid format and structure.
- **Persistent uptime**: The use of systemd ensured the service stayed active across reboots and automatically recovered from failures.
- **Seamless CI/CD pipeline**: GitHub Actions integration allowed reliable and automated code deployment on every push, reducing manual steps.
- **Secure and maintainable setup**: SSH key-based access, environment variable management, and structured logging improved the overall maintainability and security of the deployment.

3.10.2 Areas for Improvement

While the deployment was overall successful, several areas were identified that could be enhanced in future iterations:

- **Inference latency**: Model response time remains high (up to 90 seconds), especially when generating multiple MCQs. This could impact user experience without frontend handling.
- **Manual model upload**: Uploading the .gguf model manually via SCP adds extra steps. Automating this or storing the model in cloud storage could improve speed and consistency.
- **Cold start delay**: First inference run after boot takes longer due to model and embedding initialization, which could delay availability after restarts.
- **Output formatting**: In rare cases, the model returned improperly formatted answers, which required fallback logic to handle cleanly in post-processing.
- **Monitoring gaps**: While logs were reviewed manually, no live monitoring or alert system was in place to detect failures without developer involvement.

3.10.3 Recommendations

To further enhance the reliability, performance, and maintainability of the backend in future deployments, the following recommendations are made:

- **Introduce Docker-based containerization** to simplify re-deployment and improve cross-environment compatibility.
- **Implement automatic model provisioning** as part of CI/CD or startup script, especially for large .gguf models.
- **Optimize generation performance** by exploring smaller or GPU-accelerated model versions, or implementing request batching strategies.
- **Integrate monitoring tools** such as Prometheus and Grafana or simple API uptime checkers to track service health automatically.
- **Enhance error handling and logs** with structured JSON logs and custom exception tracking to improve observability during inference and API handling.