NLP Applications - Assignment 2 – PS-7 (Part A – TASK B)

Group 39

Group Member Names:

- Akilan K. S. L., 2024AB05003
- Devender Kumar, 2024aa05065
- Nagendra Prasad Reddy K. V. S., 2024aa05960
- Sai Venkata Naga Sesh Kumar Ghanta, 2024aa05989
- Piramanayagam P., 2024AB05015

Table of Contents

Problem Definition: PART A - Task B : Enhancement Plan (2 Marks)

Provide a detailed documentation that would explain the step – by- step process to enhance your Sentiment Analysis Application to develop and expose RESTful APIs that allow external applications to access the sentiment analysis engine.

Enhancement Plan

Summary:

REST APIs for Sentiment Analysis Application (preprocessing + model inference) through REST endpoints such as:

- POST /api/v1/sentiment → analyze a text string
- POST /api/v1/sentiment/batch → analyze multiple texts
- POST /api/v1/sentiment/file → analyze an uploaded .txt file
- GET /api/v1/health → health check for monitoring

Details of all enhancements in steps: (Refactor code into a clean service layer)

1. Restructure code
2. API contracts
3. REST implementation
4. Validation & Security
5. Versioning & Documentation
6. Performance Improvements
7. Logging, Metrics and Error Handling
8. Testing
9. Deployment Plan

Step 1 : Restructure - UI (HTML) and APIs call the same core inference function.

```
project/app.py              # Flask entry
sentiment/preprocess.py       # cleaning/tokenization/lemmatization
model.py          # load model + predict
service.py         # sentiment_pipeline(text) orchestration
api/routes.py          # API endpoints (Blueprint)
schemas.py          # request/response validation helpers
templates/          # existing UI
static/            # existing UI assets
```

In sentiment/service.py, create a single callable:

- analyze_text(text: str) -> dict
  - returns: label, confidence, optional scores, cleaned text, etc.

Step 2 — Define API contract (inputs/outputs)

2.1 Request format (JSON)

For single text:

```
{ "text": "I love this product!" }
```

Optional fields you can support:
return_scores: true/false
language: "en" (if you later add multilingual support)

2.2 Response format (JSON)

```
{
        "request_id": "uuid",
        "label": "positive",
        "confidence": 0.91,
        "scores": { "positive": 0.91, "neutral": 0.07, "negative": 0.02 },
        "processing_ms": 12
}
```

2.3 Standard error format

```
{
        "request_id": "uuid",
        "error": {
        "code": "INVALID_INPUT",
        "message": "Field 'text' must be a non-empty string."
  }
}
```

Step 3 — Implement REST endpoints in Flask using Blueprints

Create a blueprint api/routes.py and register it in app.py under /api/v1.

Health

GET /api/v1/health
Returns: status OK + model loaded + version

```
{ "status": "ok", "model_loaded": true, "version": "1.0.0" }
```

Single text inference

POST /api/v1/sentiment
Body: { "text": "..." }
Returns: label + confidence (+ scores optional)

Batch inference

POST /api/v1/sentiment/batch
Body:

```
{ "texts": ["Good", "Bad", "Okay"] }
```

Returns list of results with indices.

File upload

POST /api/v1/sentiment/file
multipart/form-data with file=@input.txt
Server reads file content → analyze.

Step 4 — Add input validation + security defaults

Validation rules

- text must be string, trimmed, length > 0
- Set max length (example: 5,000 chars) to prevent abuse
- For batch: max items (example: 100)

Security & hardening

- Disable debug in production
- Add CORS if external frontend/mobile apps will call it
- Add basic API key auth (simple and assignment-friendly)

API key approach

- Client sends: Authorization: Bearer <API_KEY>
- Server checks it against env var API_KEY

Step 5 — Add versioning and documentation (OpenAPI)

Versioning

Use /api/v1/... so you can add /api/v2/... later without breaking clients.

Documentation

a simple README.md section with curl examples

Step 6 — Performance improvements for API mode

Few points:

- Load model once at startup, not per request
- Keep preprocessing objects (tokenizer/vectorizer) in memory

Step 7 — Logging, monitoring, and error handling

Logging

- Log per request:
- request_id
- endpoint
- processing time
- errors (stack trace only in server logs)

Monitoring essentials

- /health endpoint for uptime checks
- response time tracking (processing_ms)

Consistent error handling

Use Flask error handlers for:

400 (bad request)
401 (unauthorized)
500 (server error)

Step 8 — Testing (very important for “detailed documentation”)

- Unit tests
- preprocessing functions
- model prediction function
- API tests
- Use Flask test client or pytest:
- valid request returns 200 + expected schema
- empty text returns 400 with INVALID_INPUT
- missing/invalid API key returns 401

Manual tests (curl examples)

```
curl -X POST http://localhost:5000/api/v1/sentiment \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer YOUR_KEY" \
  -d '{"text":"This is amazing"}'
```

Step 9 — Deployment plan

Local run

- python app.py (or flask run)

Production run

- Use a WSGI server:
- gunicorn app:app

Add config via environment variables:

- API_KEY
- MODEL_PATH
- MAX_TEXT_LENGTH
- LOG_LEVEL

Deploying on Docker:

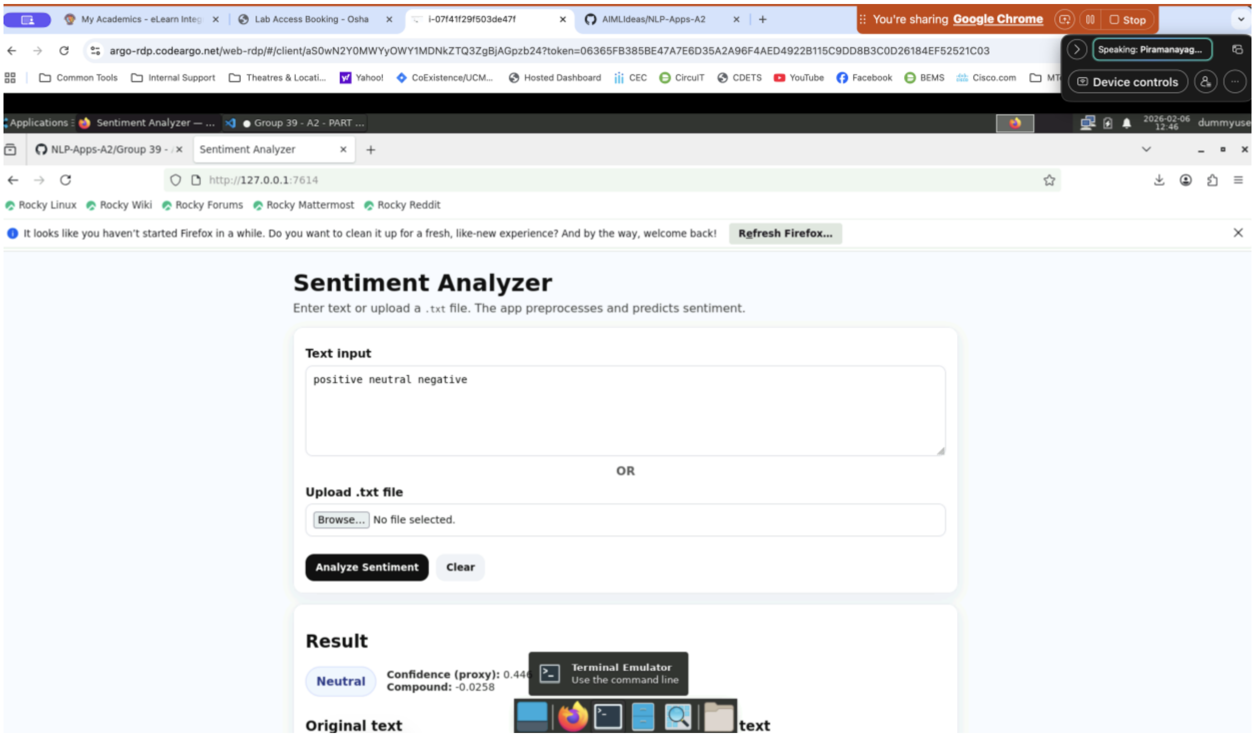
- Dockerfile builds image
- container exposes port 5000
- env vars injected at runtime

Appendix:

1. API Endpoint Table

Endpoint	Method	Purpose
/api/v1/health	GET	Health + readiness
/api/v1/sentiment	POST	Single text inference
/api/v1/sentiment/batch	POST	Batch inference
/api/v1/sentiment/file	POST	File upload inference

2. Evidence of executing in BITS OSHA Cloud lab



argo-rdp.codeargo.net/web-rdp/#/client/aS0wN2Y0MWYyOWY1MDNkZTQ3ZgBjAGpzb24?token=06365FB385BE47A7E6D35A2A96F4AED4922B115C9DD8B3C0D26184EF52521C03

http://127.0.0.1:7614

It looks like you haven't started Firefox in a while. Do you want to clean it up for a fresh, like-new experience? And by the way, welcome back!

Sentiment Analyzer

Enter text or upload a .txt file. The app preprocesses and predicts sentiment.

Text input

```
positive neutral negative
```

OR

Upload .txt file

Browse... No file selected.

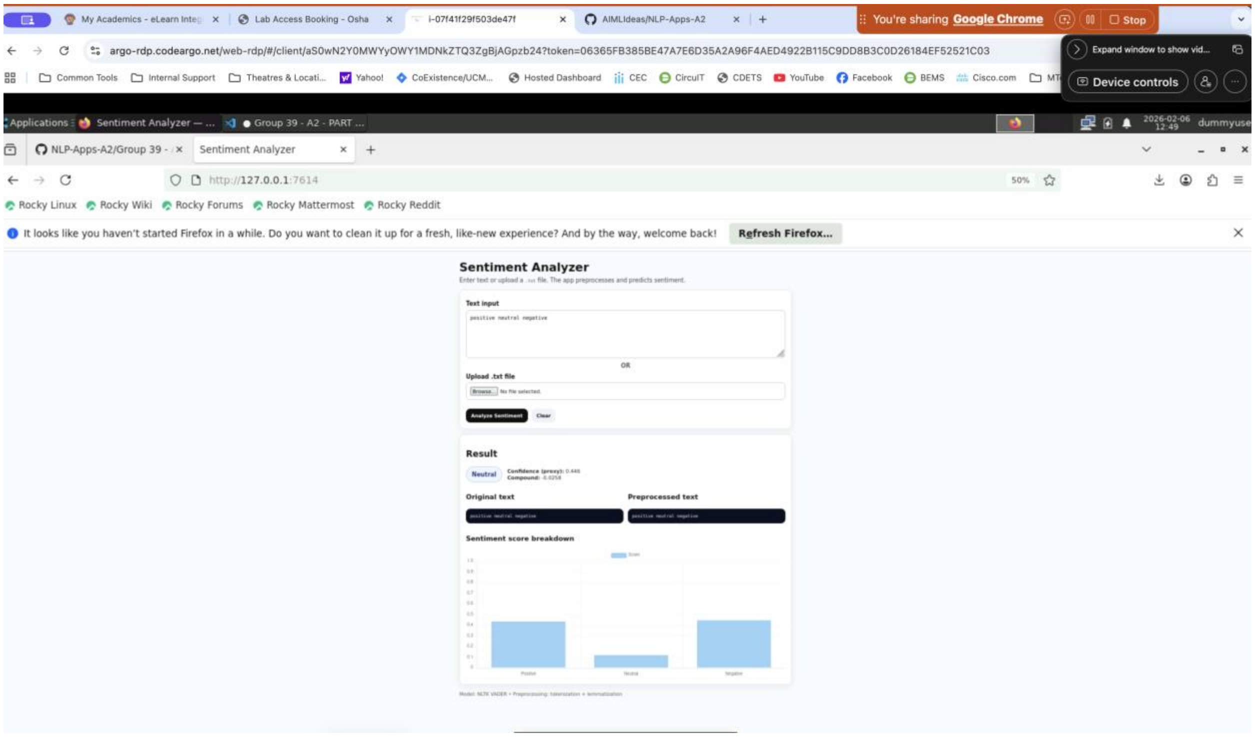
Analyze Sentiment   Clear

Result

Neutral   **Confidence (proxy):** 0.44
Compound: -0.0258

Original text

text



argo-rdp.codeargo.net/web-rdp/#/client/aS0wN2Y0MWYyOWY1MDNkZTQ3ZgBjAGpzb24?token=06365FB385BE47A7E6D35A2A96F4AED4922B115C9DD8B3C0D26184EF52521C03

http://127.0.0.1:7614

It looks like you haven't started Firefox in a while. Do you want to clean it up for a fresh, like-new experience? And by the way, welcome back!

Sentiment Analyzer

Enter text or upload a .txt file. The app preprocesses and predicts sentiment.

Text input

```
positive neutral negative
```

OR

Upload .txt file

Browse... No file selected.

Analyze Sentiment   Clear

Result

Neutral   **Confidence (proxy):** 0.446
Compound: -0.0258

Original text

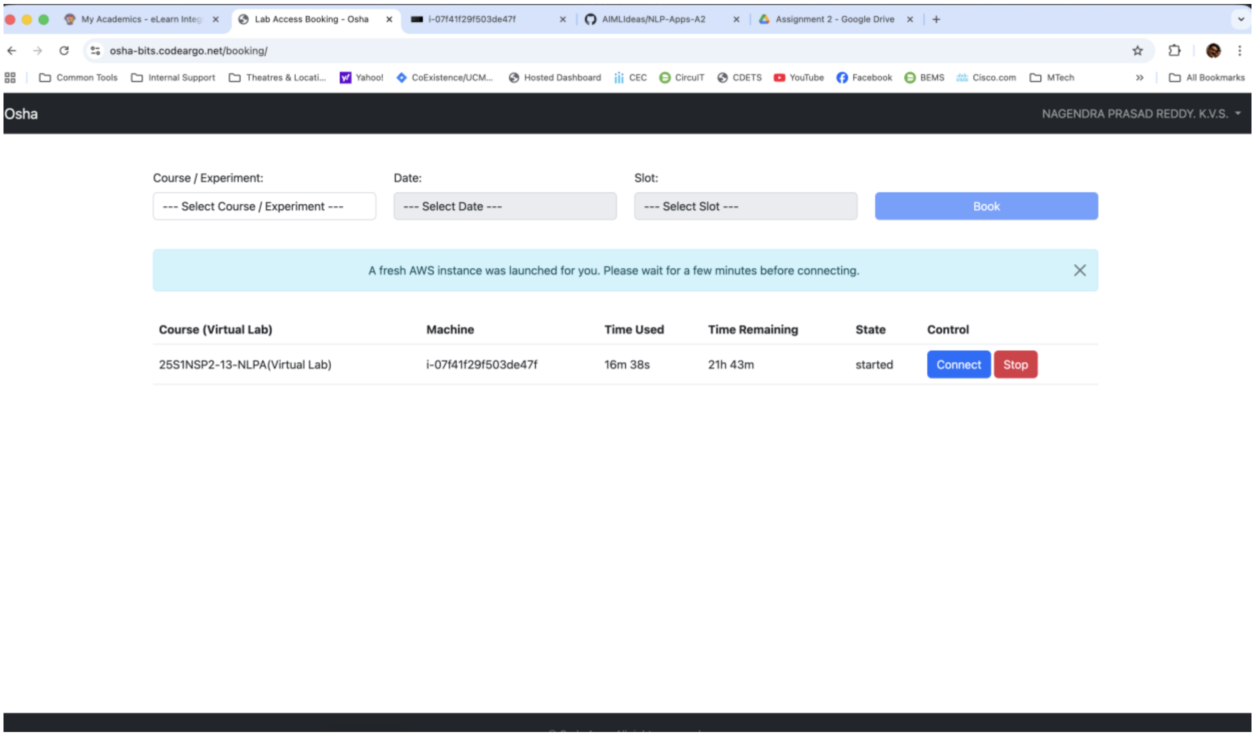
```
positive neutral negative
```

Preprocessed text

```
positive neutral negative
```

Sentiment score breakdown

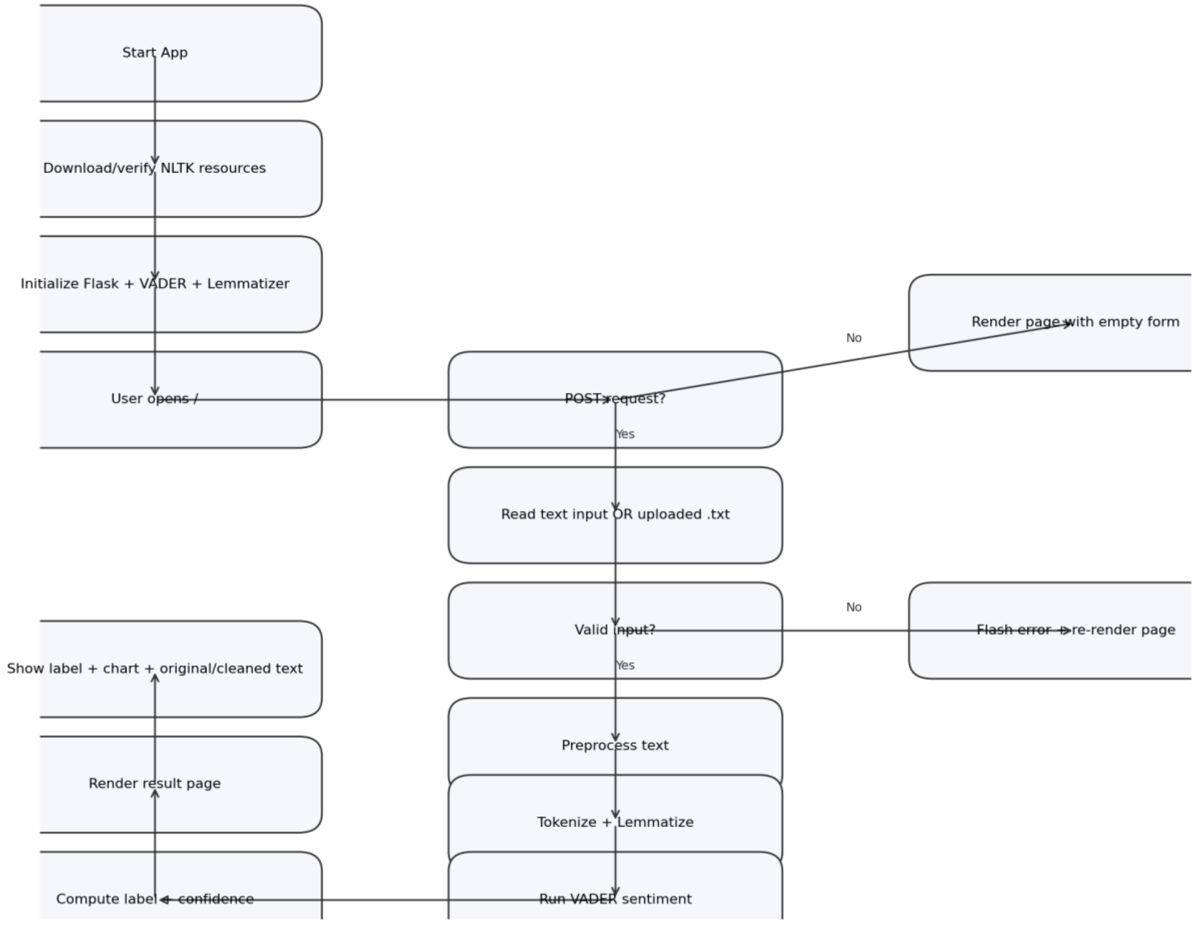
Model: NLTK VADER • Preprocessing: tokenization + lemmatization



A fresh AWS instance was launched for you. Please wait for a few minutes before connecting.

Course (Virtual Lab)	Machine	Time Used	Time Remaining	State	Control
25S1NSP2-13-NLPA(Virtual Lab)	i-07f41f29f503de47f	16m 38s	21h 43m	started	Connect Stop

3. Design & Flows


Start App
Download/verify NLTK resources
Initialize Flask + VADER + Lemmatizer
User opens /
POST request?
No
Render page with empty form
Yes
Read text input OR uploaded .txt
Valid input?
No
Flash error → re-render page
Yes
Preprocess text
Tokenize + Lemmatize
Run VADER sentiment
Compute label + confidence
Render result page
Show label + chart + original/cleaned text