AI POEM GENERATOR (NEBIUS + RAG) – FULL OFFLINE STUDY GUIDE

1. IMPORTS

```
import os
import gradio as gr
from openai import OpenAI
```

These three imports prepare the environment: - os → handles system-level operations (mainly used for environment variables like API keys). - gradio → library for building web interfaces for AI models. - openai → Python client compatible with Nebius's OpenAI-style API endpoint.

2. SETTING THE API KEY

```
os.environ["NEBIUS_API_KEY"] = ""
```

This stores your API key in an environment variable. The key is required to authenticate when making requests to Nebius AI.

3. CREATING THE NEBIUS CLIENT

```
client = OpenAI(
base_url="https://api.studio.nebius.ai/v1",
api_key=os.getenv("NEBIUS_API_KEY"),
)
```

This initializes the Nebius client: - base_url specifies where API requests are sent. - api_key fetches your stored key. Because Nebius follows the OpenAI standard, the OpenAI client works seamlessly.

4. RAG-LIKE POET REFERENCE DATA

```
poet_references = {
"shakespeare": "Shall I compare thee to a summer's day? ...",
"poe": "Once upon a midnight dreary, ...",
"tagore": "Where the mind is without fear ...",
}
```

Here you store small text chunks representing each poet's style. This is a **mock RAG (Retrieval-Augmented Generation)** component. The model receives these lines as stylistic inspiration.

5. CONTEXT RETRIEVAL FUNCTION

```
def get_poetic_context(poet_name):
return poet_references.get(poet_name.lower(), "")
```

This function retrieves the poet's text snippet from the dictionary. If the poet doesn't exist, returns an empty string.

6. THE MAIN EMOTION-BASED POEM GENERATOR

```
def generate_poem(emotion, rhyme_scheme, mood, style, theme, language, poet_style):
# 1. Rhyme instruction
rhyme_instruction = {...}.get(rhyme_scheme.lower(), "Use free verse")

# 2. Base prompt construction
base_prompt = f"Write a 2-stanza poem ... expressing '{emotion}' ..."

# 3. Optional poet style RAG context
if poet_style in poet_references:
context = get_poetic_context(poet_style)
base_prompt = f"Using this style:\n\n{context}\n\nNow, {base_prompt}"

# 4. Call Nebius model
response = client.chat.completions.create(
model="meta-llama/Meta-Llama-3.1-8B-Instruct",
...
)
return response.choices[0].message.content.strip()
```

This function does all the heavy lifting: 1. Creates rhyme scheme instructions (ABAB, AABB, etc.). 2. Builds a structured prompt describing the poem. 3. Adds RAG-style context if the user selects a poet. 4. Calls the Llama 3.1 model on Nebius. 5. Returns the generated poem text.

7. PHRASE-BASED POEM GENERATOR

```
def generate_phrase_poem(phrase, language, style, mood, poet_style):
base_prompt = f"Write a poem inspired by '{phrase}'..."

if poet_style in poet_references:
context = get_poetic_context(poet_style)
base_prompt = f"Using this style:\n\n{context}\n\nNow, {base_prompt}"

response = client.chat.completions.create(...)
return response.choices[0].message.content.strip()
```

This version is similar to the emotion-based generator, but takes a **phrase** as the core inspiration instead of emotion/theme.

8. INTERFACE LOGIC FUNCTION

```
def generate_interface(choice, ...):
if choice == "Emotion-based":
return generate_poem(...)
else:
return generate_phrase_poem(...)
```

This controller selects the correct generator depending on the user's choice. It makes the entire system flexible and clean.

9. BUILDING THE GRADIO USER INTERFACE

```
with gr.Blocks() as demo:
gr.Markdown("# AI Poem Generator")
choice = gr.Radio([...])

with gr.Group(visible=True):
...

with gr.Group(visible=False):
...

output = gr.Textbox()

choice.change(toggle, ...)

generate_btn.click(fn=generate_interface, outputs=output)

demo.launch(share=True)
```

This section builds the full web interface using Gradio Blocks: - Markdown → titles/descriptions - Radio → lets user choose poem type - Textboxes & Dropdowns → capture poem parameters - Group → allows showing/hiding input sections - Button → triggers poem creation - launch() → starts the app and generates a public link for sharing

10. OVERALL SYSTEM WORKFLOW

```
1. User selects poem type
2. Enters required inputs
3. Gradio reveals relevant fields
4. User clicks Generate
5. Prompt is built with optional RAG context
6. Nebius Llama model creates poem
7. Poem appears in output field
```

This summarizes how the entire application behaves end to end.