Overview: I first tried to do it by myself, but it seems that the slide doesn't provide me enough information. So I have to use ChatGPT to help me do it. But I made sure to ask ChatGPT to ask what's going on in the code. The description and analysis are my own ideas(but I use ChatGPT to fix my English).

https://chatgpt.com/share/696c54ff-cf40-8010-91c3-9da92f85b561

any recommendation for lightweight llm for testing?
Demonstrates at least 2 different models via OpenRouter as a dropdown in Gradio. (Not necessarily at the same time.)
Try to pick models of different sizes to compare the differences. If you search for free models (https://openrouter.ai/models?q=free) you can also avoid incurring any costs.

how do I do this?
what API do I use?

```
import gradio as gr
from pydantic import BaseModel

MODELS = {
    "gpt-oss-20b (free, smaller)": "openai/gpt-oss-20b:free",
    "MiMo-V2-Flash (free, huge MoE)": "xiaomi/mimo-v2-flash:free",
}

class GameInformation(BaseModel):
  name: str
  genre: str
  console: str
  price: float
  year: int

def chat(model_label, user_msg, history):
    model_id = MODELS[model_label]

    messages = history + [
        {"role": "system", "content": "You help users find the game they are looking for."},
        {"role": "user", "content": user_msg}
        ]

    response = client.chat.completions.create(
        model=model_id,
        messages=messages,
        response_format=GameInformation,
        stream=True, # Enable streaming
    )
```

```
    assistant_msg = response.choices[0].message.content

    history.append([user_msg, assistant_msg])
    return history, ""  # update chatbot, clear textbox

with gr.Blocks() as demo:
    gr.Markdown("## OpenRouter model switcher (Gradio)")

    model_dd = gr.Dropdown(
        choices=list(MODELS.keys()),
        value=list(MODELS.keys())[0],
        label="Model",
    )

    chatbot = gr.Chatbot(label="Chat", height=360)
    txt = gr.Textbox(label="Your message")
    send = gr.Button("Send")

    send.click(chat, inputs=[model_dd, txt, chatbot], outputs=[chatbot, txt])
    txt.submit(chat, inputs=[model_dd, txt, chatbot], outputs=[chatbot, txt])

demo.launch()
```

is there something wrong with this?
yes, I want it to be streaming

```
import gradio as gr
from pydantic import BaseModel

MODELS = {
    "gpt-oss-20b (free, smaller)": "openai/gpt-oss-20b:free",
    "MiMo-V2-Flash (free, huge MoE)": "xiaomi/mimo-v2-flash:free",
}

class GameInformation(BaseModel):
  name: str
  genre: str
  console: str
  price: float
  year: int

SYSTEM = (
    "You help users find the game they are looking for.\n"
    "Return ONLY valid JSON with keys: name, genre, console, price, year."
```

```
)

def history_to_messages(history):
    messages = [{"role": "system", "content": SYSTEM}]
    for u, a in history:
        messages.append({"role": "user", "content": u})
        messages.append({"role": "assistant", "content": a})
    return messages

def chat(model_label, user_msg, history):
    model_id = MODELS[model_label]

    messages = history + [{"role": "user", "content": user_msg}]

    stream = client.chat.completions.create(
        model=model_id,
        messages=messages,
        stream=True,
    )

    response_text = ""
    for chunk in stream:
        if chunk.choices[0].delta.content is not None:
            token = chunk.choices[0].delta.content
            response_text += token
            yield response_text



with gr.Blocks() as demo:
    gr.Markdown("## OpenRouter model switcher (Gradio)")

    model_dd = gr.Dropdown(
        choices=list(MODELS.keys()),
        value=list(MODELS.keys())[0],
        label="Model",
   )

    chatbot = gr.Chatbot(label="Chat", height=360)
    txt = gr.Textbox(label="Your message")
    send = gr.Button("Send")

    send.click(chat, inputs=[model_dd, txt, chatbot], outputs=[chatbot, txt])
    txt.submit(chat, inputs=[model_dd, txt, chatbot], outputs=[chatbot, txt])
```

```
demo.launch()

what causes error here?
import os
import gradio as gr
from pydantic import BaseModel
from openai import OpenAI

MODELS = {
    "openai/gpt-5.2-chat": "openai/gpt-5.2-chat",
    "MiMo-V2-Flash (free, huge MoE)": "xiaomi/mimo-v2-flash:free",
}

client = OpenAI(
    base_url="https://openrouter.ai/api/v1",
    api_key=OPENROUTER_API_KEY,
)

class GameInformation(BaseModel):
    name: str
    genre: str
    console: str
    price: float
    year: int

SYSTEM = (
    "You help users find the game they are looking for.\n"
    "Return ONLY valid JSON with keys: name, genre, console, price, year."
)

def history_to_messages(history):
    messages = [{"role": "system", "content": SYSTEM}]
    for u, a in history:
        messages.append({"role": "user", "content": u})
        messages.append({"role": "assistant", "content": a})
    return messages

def chat(model_label, user_msg, history):
    model_id = MODELS[model_label]

    messages = history_to_messages(history)
    messages.append({"role": "user", "content": user_msg})
```

```
    # Add a new chat row immediately
    history = history + [[user_msg, ""]]
    yield history, ""   # update chatbot + clear textbox

    stream = client.chat.completions.create(
        model=model_id,
        messages=messages,
        stream=True,
    )

    response_text = ""
    for chunk in stream:
        delta = chunk.choices[0].delta
        if delta and getattr(delta, "content", None):
            response_text += delta.content
            history[-1][1] = response_text
            yield history, ""  # keep streaming updates

with gr.Blocks() as demo:
    gr.Markdown("## OpenRouter model switcher (Gradio)")

    model_dd = gr.Dropdown(list(MODELS.keys()), value=list(MODELS.keys())[0], label="Model")
    chatbot = gr.Chatbot(label="Chat", height=360)
    txt = gr.Textbox(label="Your message")
    send = gr.Button("Send")

    send.click(chat, inputs=[model_dd, txt, chatbot], outputs=[chatbot, txt])
    txt.submit(chat, inputs=[model_dd, txt, chatbot], outputs=[chatbot, txt])

demo.launch()
"
```

I want it to output a chat and json output later.

also make it begin with a message like what kind of game are you looking for
It produces this, this is not what I want

```
import os
import json
import gradio as gr
from pydantic import BaseModel, ValidationError
from openai import OpenAI

MODELS = {
    "openai/gpt-5.2-chat": "openai/gpt-5.2-chat",
```

```
    "MiMo-V2-Flash (free)": "xiaomi/mimo-v2-flash:free",
}

client = OpenAI(
    base_url="https://openrouter.ai/api/v1",
    api_key=os.environ["OPENROUTER_API_KEY"],
    default_headers={
        "HTTP-Referer": "http://localhost:7860",
        "X-Title": "Gradio OpenRouter Streaming + JSON",
    },
)

class GameInformation(BaseModel):
    name: str
    genre: str
    console: str
    price: float
    year: int

DELIM = "---JSON---"

SYSTEM = f"""
You help users find the game they are looking for.

Output format rules:
1) First: reply normally in plain English (recommendation + brief reasoning).
2) Then on a new line output exactly: {DELIM}
3) Then output ONLY one valid JSON object with keys:
   name, genre, console, price, year

No markdown code fences. No extra text after the JSON.
"""

STARTER = "What kind of game are you looking for?"

def history_to_messages(history):
    messages = [{"role": "system", "content": SYSTEM}]
    for u, a in history:
        if u:
            messages.append({"role": "user", "content": u})
        if a:
            messages.append({"role": "assistant", "content": a})
    return messages
```

```
def chat(model_label, user_msg, history):
    model_id = MODELS[model_label]

    messages = history_to_messages(history)
    messages.append({"role": "user", "content": user_msg})

    # add new row and clear textbox right away
    history = history + [[user_msg, ""]]
    yield history, "", None

    stream = client.chat.completions.create(
        model=model_id,
        messages=messages,
        temperature=0.7,
        stream=True,
    )

    visible_text = ""
    json_text = ""
    saw_delim = False

    for chunk in stream:
        delta = chunk.choices[0].delta
        token = getattr(delta, "content", None)
        if not token:
            continue

        if not saw_delim:
            visible_text += token

            # If delimiter appears in the visible stream, split it
            if DELIM in visible_text:
                before, after = visible_text.split(DELIM, 1)
                visible_text = before.rstrip()
                json_text += after
                saw_delim = True

            history[-1][1] = visible_text
            yield history, "", None
        else:
            # After delimiter: do NOT show in chat, just collect JSON tokens
            json_text += token

    # Parse JSON after streaming completes
```

```
    json_blob = json_text.strip()
    # Some models may accidentally prepend newlines; strip again
    json_blob = json_blob.strip()

    try:
        data = json.loads(json_blob)
        game = GameInformation(**data)
        json_out = game.model_dump()
    except (json.JSONDecodeError, ValidationError) as e:
        json_out = {
            "error": "Could not parse valid GameInformation JSON after delimiter.",
            "details": str(e),
            "raw_json_section": json_blob,
            "raw_visible_section": visible_text.strip(),
        }

    yield history, "", json_out

with gr.Blocks() as demo:
    gr.Markdown("## OpenRouter model switcher (Streaming chat + JSON output)")

    model_dd = gr.Dropdown(list(MODELS.keys()), value=list(MODELS.keys())[0], label="Model")
    chatbot = gr.Chatbot(label="Chat", height=360, value=[[None, STARTER]])
    txt = gr.Textbox(label="Your message")
    send = gr.Button("Send")
    json_panel = gr.JSON(label="GameInformation (parsed after completion)")

    send.click(chat, inputs=[model_dd, txt, chatbot], outputs=[chatbot, txt, json_panel])
    txt.submit(chat, inputs=[model_dd, txt, chatbot], outputs=[chatbot, txt, json_panel])

demo.launch()
```

it's good but please add GameInformation when normally reply in the plain English as well. Also, please allow multiple games answer
"X-Title": "Video Game Finder Expert",

do you think this is a good name?

```
import os
import json
import gradio as gr
from pydantic import BaseModel, ValidationError
from openai import OpenAI
from typing import List
```

```
MODELS = {
    "openai/gpt-5.2-chat": "openai/gpt-5.2-chat",
    "MiMo-V2-Flash (free)": "xiaomi/mimo-v2-flash:free",
}

client = OpenAI(
    base_url="https://openrouter.ai/api/v1",
    api_key=OPENROUTER_API_KEY,
    default_headers={
        "HTTP-Referer": "http://localhost:7860",
        "X-Title": "Game Recommendation Assistant",
    },
)

class GameInformation(BaseModel):
    name: str
    genre: str
    console: str
    price: float
    year: int

DELIM = "---JSON---"

SYSTEM = f"""
You help users find the game they are looking for.

Output format rules:
1) First: reply normally in plain English (recommendations + brief reasoning).
   - Include a short section titled "GameInformation (summary)".
   - In that section, list each recommended game in 1 line using this format:
     Name | Genre | Console | Price | Year
   - Do NOT use JSON in this first section.

2) Then on a new line output exactly: {DELIM}

3) Then output ONLY valid JSON as an ARRAY of objects (a list), each object with keys:
   name, genre, console, price, year

No markdown code fences. No extra text after the JSON.
"""

STARTER = "What kind of game are you looking for?"

def history_to_messages(history):
```

```
    messages = [{"role": "system", "content": SYSTEM}]
    for u, a in history:
        if u:
            messages.append({"role": "user", "content": u})
        if a:
            messages.append({"role": "assistant", "content": a})
    return messages

def chat(model_label, user_msg, history):
    model_id = MODELS[model_label]

    messages = history_to_messages(history)
    messages.append({"role": "user", "content": user_msg})

    # add new row and clear textbox right away
    history = history + [[user_msg, ""]]
    yield history, "", None

    stream = client.chat.completions.create(
        model=model_id,
        messages=messages,
        temperature=0.7,
        stream=True,
    )

    visible_text = ""
    json_text = ""
    saw_delim = False

    for chunk in stream:
        delta = chunk.choices[0].delta
        token = getattr(delta, "content", None)
        if not token:
            continue

        if not saw_delim:
            visible_text += token

            if DELIM in visible_text:
                before, after = visible_text.split(DELIM, 1)
                visible_text = before.rstrip()
                json_text += after
                saw_delim = True
```

```
            history[-1][1] = visible_text
            yield history, "", None
        else:
            json_text += token

    # Parse JSON after streaming completes
    json_blob = json_text.strip()

    try:
        data = json.loads(json_blob)
        if not isinstance(data, list):
            raise ValidationError.from_exception_data(
                "GameInformationList",
                [{"loc": ("root",), "msg": "Expected a JSON array (list) of games.", "type":
"value_error"}],
            )

        games: List[GameInformation] = [GameInformation(**item) for item in data]
        json_out = [g.model_dump() for g in games]

    except Exception as e:
        json_out = {
            "error": "Could not parse valid JSON array of GameInformation after delimiter.",
            "details": str(e),
            "raw_json_section": json_blob,
            "raw_visible_section": visible_text.strip(),
        }

    yield history, "", json_out

with gr.Blocks() as demo:
    gr.Markdown("## OpenRouter model switcher (Streaming chat + JSON output)")

    model_dd = gr.Dropdown(list(MODELS.keys()), value=list(MODELS.keys())[0], label="Model")
    chatbot = gr.Chatbot(label="Chat", height=360, value=[[None, STARTER]])
    txt = gr.Textbox(label="Your message")
    send = gr.Button("Send")
    json_panel = gr.JSON(label="GameInformation list (parsed after completion)")

    send.click(chat, inputs=[model_dd, txt, chatbot], outputs=[chatbot, txt, json_panel])
    txt.submit(chat, inputs=[model_dd, txt, chatbot], outputs=[chatbot, txt, json_panel])

demo.launch()
```

it all works but please explain each line of code. what do they do?
{
"error"
:
"Could not parse valid JSON array of GameInformation after delimiter."
,
"details"
:
"1 validation error for GameInformation price Input should be a valid number, unable to parse string as a number [type=float_parsing, input_value='$59.99', input_type=str] For further information visit https://errors.pydantic.dev/2.12/v/float_parsing"
,
"raw_json_section"
:
"[{ "name": "Donkey Kong Country: Tropical Freeze", "genre": "Platformer", "console": "Nintendo Switch", "price": "$59.99", "year": 2018 }, { "name": "Donkey Kong 64", "genre": "3D Platformer", "console": "Nintendo 64", "price": "$49.99", "year": 1999 }, { "name": "Gorilla Tag", "genre": "VR Multiplayer", "console": "PC (VR), Meta Quest", "price": "Free", "year": 2021 }, { "name": "Rampage World Tour", "genre": "Arcade Action", "console": "PlayStation, N64", "price": "$19.99", "year": 1997 }, { "name": "Super Smash Bros. Ultimate", "genre": "Fighting", "console": "Nintendo Switch", "price": "$59.99", "year": 2018 }]"
,
"raw_visible_section"
:
"If you’re specifically looking for **games that feature gorillas as main characters or major elements**, here are some strong and well-known options across different styles and platforms. I’ve focused on games where gorillas are central, not just background enemies. ### Recommendations - **Donkey Kong Country series** – The most iconic gorilla-led platformers, known for tight controls, memorable music, and creative levels. - **Donkey Kong 64** – A 3D adventure where Donkey Kong and other Kongs explore a large world, collect items, and battle enemies. - **Gorilla Tag** – A unique VR multiplayer game where players move like gorillas using their arms; extremely popular for its physical gameplay. - **Rampage series** – Lets you control giant monsters including George the gorilla, smashing cities in classic arcade-style chaos. - **Super Smash Bros. Ultimate** – While not gorilla-only, Donkey Kong is a major playable character with a gorilla-focused moveset. ### GameInformation (summary) Donkey Kong Country: Tropical Freeze | Platformer | Nintendo Switch | $59.99 | 2018 Donkey Kong 64 | 3D Platformer | Nintendo 64 | $49.99 | 1999 Gorilla Tag | VR Multiplayer | PC (VR), Meta Quest | Free | 2021 Rampage World Tour | Arcade Action | PlayStation, N64 | $19.99 | 1997 Super Smash Bros. Ultimate | Fighting | Nintendo Switch | $59.99 | 2018"

So there is always a json output, I don't get why it is always error(in a good way)
OpenRouter model switcher (Streaming chat + JSON output)

why the title doesn't change?

```
client = OpenAI(
    base_url="https://openrouter.ai/api/v1",
    api_key=OPENROUTER_API_KEY,
    default_headers={
        "HTTP-Referer": "http://localhost:7860",
        "X-Title": "Game Recommendation Assistant",
    },
)
```

What Does this Notebook Do?

Overview

This program, "Game Recommendation Assistant," is created to output the game that users are looking for with the based JSON format "Game Information" that includes "name, genre, console, price, and year".

To make it more like an assistant, not a database created just to give information, I design it in a way that it outputs a message first, then output the game recommendation, then details based on the JSON so that the user can use them to make the decision.

Technical Overview

I make sure that the program has the visible text part and the json part. What is the point of parsing the JSON. It might not have a purpose now, but I just want to know how to do this for the future purpose. I might need to use JSON from generated from chatGPT to use it within the game for example. ANd user can get the generated output for their data as well.

Fix my English

From these examples. In my opinion, with the temperature 0.7, the chat outputs between these 2 paid and free models are about the same respectively. However, the one thing that the paid model can do and the free model can't do is generating JSON. This is very important. Let's say I want to update the game state within the Unity where the JSON is parsed to update, I need to use the paid model.

Fix my English