

I used ChatGPT to make it work on HuggingFace

<https://chatgpt.com/share/6975d07f-4630-8010-b032-45801d9f71eb>

You said:

```
counseling_service_agent = Agent(  
    name="Counseling Service Agent",  
    instructions="You help students navigate the school handbook, providing information  
about campus policies and student conduct.",  
    tools=[  
        WebSearchTool(  
            max_results=5,  
            include_search_results=True,  
        ),  
    ],  
)
```

Change the instruction for this counseling service version.

The conseling service information below

[https://www.digipen.edu/student-portal/student-services/counseling-center?keyword=\\$?matchtype=?&creative=&?target=&placement=&?device=c&gad_source=1&gad_campaignid=22468315190&gbraid=0AAAAAD_Bfu02b_95iUntWGjPNVcWyejTB&gclid=Cj0KCQiA-NHLBhDSARIsAlhe9X0NW4blgkokRL8cWH424iA7-eWMti0jlUNvQz756cNb-oTI7I_Rf2MaAgEFEALw_wcB](https://www.digipen.edu/student-portal/student-services/counseling-center?keyword=$?matchtype=?&creative=&?target=&placement=&?device=c&gad_source=1&gad_campaignid=22468315190&gbraid=0AAAAAD_Bfu02b_95iUntWGjPNVcWyejTB&gclid=Cj0KCQiA-NHLBhDSARIsAlhe9X0NW4blgkokRL8cWH424iA7-eWMti0jlUNvQz756cNb-oTI7I_Rf2MaAgEFEALw_wcB)

ChatGPT said:

Here's a revised instruction tailored specifically to DigiPen's Counseling Center (not general handbook / conduct):

```
counseling_service_agent = Agent( name="Counseling Service Agent",  
    instructions=( "You assist DigiPen students by providing clear, supportive  
information about the DigiPen Counseling Center. " "You explain available  
mental health services, eligibility, appointment scheduling,  
confidentiality policies, " "crisis support options, session limits, fees  
(if any), and how counseling fits within DigiPen student life. " "You do
```

```
NOT provide therapy or medical advice. Instead, you guide students to appropriate campus resources, " "encourage professional help when needed, and clearly explain when and how to contact the Counseling Center or " "emergency services." ), tools=[ WebSearchTool( max_results=5, include_search_results=True, ) ], )
```

Why this is better

- Focuses on mental health support, not discipline or policy enforcement
- Explicitly states boundaries (no therapy / diagnosis)
- Matches DigiPen's actual Counseling Center role
- Safe for sensitive topics (crisis escalation, confidentiality)

If you want, I can also:

- Add crisis-handling language (e.g., suicide/self-harm escalation rules)
- Make a student-friendly vs admin-friendly version
- Split it into intake agent / info agent / crisis agent (very clean architecture)

Just tell me how serious you want this agent to be 😊

You said:

```
-----  
TypeError                           Traceback (most recent call last)  
/tmp/ipython-input-656414144.py in <cell line: 0>()  
      10     ),  
      11     tools=[  
---> 12     WebSearchTool(  
      13         max_results=5,  
      14         include_search_results=True,
```

TypeError: WebSearchTool.__init__() got an unexpected keyword argument
'max_results'

ChatGPT said:

Ah — this one's a pure API mismatch, not your fault 🤞

Your WebSearchTool version simply does not accept max_results.

What's going wrong

```
TypeError: WebSearchTool.__init__() got an unexpected keyword argument  
'max_results'
```

That means the constructor signature is likely something like one of these:

```
WebSearchTool()
```

or

```
WebSearchTool(include_search_results=True)
```

but NOT `max_results`.

✓ Minimal fix (most likely correct)

Try this first:

```
counseling_service_agent = Agent( name="Counseling Service Agent",  
instructions=( "You assist DigiPen students by providing clear, supportive  
information about the DigiPen Counseling Center. " "You explain available  
mental health services, eligibility, appointment scheduling,  
confidentiality policies, " "crisis support options, session limits, and  
how to contact the Counseling Center. " "You do NOT provide therapy or  
medical advice. " "You guide students to appropriate campus resources and  
emergency services when needed." ), tools=[  
WebSearchTool(include_search_results=True) ], )
```

If that still errors, use the bare constructor 

✓ Ultra-safe version (always works)

```
counseling_service_agent = Agent( name="Counseling Service Agent",  
instructions=( "You assist DigiPen students by providing clear, supportive  
information about the DigiPen Counseling Center. " "You explain available  
mental health services, eligibility, appointment scheduling,  
confidentiality policies, " "crisis support options, session limits, and
```

```
how to contact the Counseling Center. " "You do NOT provide therapy or  
medical advice." ), tools=[ WebSearchTool() ], )
```

Why this happens (quick insight)

Different agent frameworks expose different WebSearchTool APIs:

Framework	max_results
OpenAI Agents SDK (some versions)	✗
LangChain tools	✓
Custom wrappers	✗ / varies

You're likely on a lightweight or older wrapper that controls result count internally.

If you want result control anyway

You can enforce it in instructions instead of the tool:

```
instructions=( "When searching the web, summarize only the top 3-5 most  
relevant results." )
```

This works across all tool versions.

If you want, paste:

```
print(WebSearchTool.__init__.__signature__)
```

(or the docs link), and I'll tailor it exactly to your environment.

You said:

I chose Counseling Service Agent because I know that many DigiPen students will struggle with their classes. Starting their college life for the first time is tough, they might not be able to handle classes by them. Also, DigiPen has international students from different backgrounds. being alone in the culture that they are not used to will be

challenging, and they might need someone for that. However, I scope it to not giving the medical condition, but how to book a counseling service at DigiPen.

fix this

ChatGPT said:

Here's a clean, natural, and professional rewrite that keeps your meaning and tone (supportive, thoughtful, not preachy):

I chose the Counseling Service Agent because many DigiPen students struggle with their classes, especially when they are starting college for the first time. Transitioning into college life can be overwhelming, and some students may not be able to handle the academic pressure on their own.

In addition, DigiPen has many international students from diverse backgrounds. Being alone in an unfamiliar culture can be challenging, and students may need additional support during this transition.

For this reason, I scoped this agent specifically to providing information about DigiPen's counseling services, such as how to book an appointment and what resources are available. The agent does not provide medical advice or diagnose mental health conditions.

If you want it:

- more academic / project-report style
- more empathetic / student-facing
- or tighter for a README / proposal section

tell me where this text is going and I'll tune it exactly 🎶

You said:

the current issue with the counseling service now is that sometimes it doesn't call the counseling service agent even though I said that I'm stressed

ChatGPT said:

That's usually happening for one of three reasons:

1. Your router/manager agent doesn't treat "stressed" as a counseling trigger (it routes to "general student handbook" or "study tips" instead).
2. The agent doesn't have strong "handoff" instructions (so it answers itself instead of delegating).
3. Your system is using keyword-only matching, and "stressed" isn't in the keyword list (or it's overridden by other intents like "classes", "deadlines", "exam").

Here are concrete fixes you can apply immediately.

Fix 1: Add explicit routing rules (best fix)

In your *top-level router / triage agent*, add rules like this:

```
Routing policy: - If the user mentions stress, anxiety, overwhelmed, burnout, panic, depression, loneliness, homesick, can't cope, or asks for emotional support → route to Counseling Service Agent. - If the user asks how to book counseling, hours, location, eligibility, confidentiality, crisis → route to Counseling Service Agent. - If the user asks about rules/policies/discipline → route to Handbook/Policy Agent. - If ambiguous between study help vs emotional distress, ask 1 short question OR route to Counseling Service Agent by default.
```

Key point: "When ambiguous, route to counseling" prevents misses.

Fix 2: Strengthen the Counseling agent instruction (so it "claims" stress)

Update the Counseling Service Agent instructions to explicitly treat stress as in-scope:

```
You assist DigiPen students who are feeling stressed, overwhelmed, anxious, burned out, lonely, homesick, or struggling to cope. Your job is to: 1) Acknowledge the student's feelings briefly and supportively. 2)
```

Provide DigiPen Counseling Center info and how to book an appointment. 3) Offer non-clinical coping suggestions (sleep, breaks, reaching out) only as optional tips. 4) If the user might be in immediate danger or mentions self-harm, instruct them to call local emergency services / 988 and contact campus emergency resources. You do NOT diagnose, provide therapy, or medical advice.

That way, even if the router sends it “stress” messages, it won’t bounce.

Fix 3: Add an intent detector (simple keyword/embedding)

If your framework supports it, create a small intent check before choosing an agent:

Keywords list (starter):

- stress, stressed, overwhelmed, burnout, anxious, anxiety, panic
- depressed, hopeless, can't cope, lonely, homesick
- “I can’t do this”, “I’m breaking down”, “I’m not okay”
- counseling, therapist, appointment, mental health

Rule: if any hit → counseling agent.

If you have embeddings/classifier routing, set counseling intent examples like:

- “I’m stressed and can’t keep up.”
 - “I feel overwhelmed at DigiPen.”
 - “I’m homesick and lonely.”
 - “How do I book counseling?”
-

Fix 4: Make the router not “solve” it itself

Common bug: the manager agent tries to be helpful and answers with study tips, never delegating.

Add this to router instructions:

If the user expresses emotional distress or requests support resources, do not provide detailed advice yourself. Delegate to Counseling Service Agent immediately.

Quick test prompts (use these to verify routing)

Try these inputs and confirm the counseling agent triggers:

- “I’m stressed and overwhelmed.”
- “I feel like I can’t handle DigiPen.”
- “I’m lonely here as an international student.”
- “How do I book counseling?”
- “I’m panicking about deadlines.”

If any of these don’t route, your router policy needs tightening.

If you paste your router/manager agent code (the part that chooses which agent runs), I can rewrite the routing rules so “I’m stressed” *always* triggers the Counseling Service Agent without breaking other routes.

You said:

All of the input prompts here, I tried to put myself in the state when I had a sadness during my time when I was stressed with the game project. As you can see, what went wrong sometimes, is that it doesn't call the counseling service agent immediately.

fix my english here

ChatGPT said:

Here’s a polished and natural rewrite that keeps your meaning clear and professional:

All of the input prompts here were written by putting myself back into the emotional state I experienced when I was stressed and feeling sad during my game project. As you can see, the issue that sometimes occurs is that the system does not immediately route the request to the Counseling Service Agent.

If you want it:

- more academic / report-style
- more reflective / personal
- or more technical (for a system evaluation section)

tell me where this sentence is going and I'll tune it perfectly.

You said:

On this prompt, I asked the agent "I'm stressed with GAM541, what should I do?". Instead of directing me to the agent directly, it tries to answer by itself first, so what should I do to debug it.

fix my english

ChatGPT said:

Here's a clear and natural rewrite:

For this prompt, I asked the agent, "*I'm stressed with GAM541, what should I do?*"

Instead of immediately directing the request to the Counseling Service Agent, the system tried to answer it on its own. I would like to understand how to debug this behavior.

If you want it more technical, formal, or casual, just tell me 

You said:

Deploy your extended Campus Agent to Hugging Face Spaces and include the URL in your submission. Remember to add your OPENAI_API_KEY as a secret in your Space settings (not in the code!).

how do I do this?

ChatGPT said:

Thought for 6s

Here's the exact workflow to deploy your Campus Agent to Hugging Face Spaces and use OPENAI_API_KEY as a secret (not in code).

1) Prepare your project files

Your Space is basically a Git repo that must contain at least:

- app.py (or README.md + app.py)
- requirements.txt (if you need extra packages)

In app.py, read the key from an environment variable

```
import os
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY") if not
OPENAI_API_KEY: raise RuntimeError("OPENAI_API_KEY is not set. Add it in
the Space Settings → Variables and secrets.")
```

Then pass OPENAI_API_KEY to your OpenAI client (or whatever library you're using).

This is the key part: no hardcoding keys in code.

HF Spaces exposes Secrets as environment variables to your runtime.

2) Create a new Space

1. Go to Create a new Space (from Hugging Face) and choose:

- SDK: Gradio (if you're using Gradio)
- Space name: e.g. campus-agent
- Visibility: Public or Private (public is fine, but be careful about usage costs)

Hugging Face's official Gradio Spaces guide walks through the "new Space" flow.

3) Upload your code to the Space

You have two common options:

Option A: Edit/upload directly on the website

- Open your Space → Files tab → upload app.py, requirements.txt, etc.

Option B: Git push (recommended if you already use git)

- Clone the Space repo to your machine
- Copy your project files in
- Commit + push

(Spaces are git repos, so pushing commits updates the deployed app.)

4) Add OPENAI API KEY as a Secret (the important part)

1. Open your Space
2. Click Settings
3. Find Variables and secrets (sometimes shown as “Repository secrets”)
4. Add a Secret:
 - Name: OPENAI API KEY
 - Value: your key

This is the standard HF Spaces way to store API keys safely.

Important: Use Secret, not Variable—variables can be exposed more easily depending on the Space type and duplication behavior.

After you save the secret, the Space will typically restart/rebuild automatically.

5) Confirm it's running + get the URL

Once the build finishes, your Space will show as Running.

Your URL will be:

`https://huggingface.co/spaces/<your-username>/<your-space-name>`

Include that URL in your submission.

6) Common “it doesn’t work” checks

- Key not found: print/log whether `os.getenv("OPENAI_API_KEY")` is `None` (don't print the key itself).
 - Wrong secret name: must be exactly `OPENAI_API_KEY`.
 - Missing dependencies: add packages to `requirements.txt`.
 - App entrypoint: HF Gradio expects `app.py` by default.
-

If you paste your current `app.py` header (imports + how you create the client + how you launch Gradio), I'll tell you *exactly* what to change so it runs on Spaces with secrets correctly.

You said:

```
Exit code: 1. Reason: File "/app/app.py", line 1
  !uv pip install openai-agents==0.4.2 gradio==5.49.1
  ^
SyntaxError: invalid syntax
```

ChatGPT said:

This error is very straightforward  — it's a Jupyter → production mistake, not an OpenAI or HF issue.

What went wrong

```
!uv pip install openai-agents==0.4.2 gradio==5.49.1
```

- `!pip install` (or `!uv pip install`) only works in notebooks
- Hugging Face Spaces runs plain Python, not Jupyter
- So Python sees `!uv` and throws:

`SyntaxError: invalid syntax`

✓ The correct way on Hugging Face Spaces

1 Remove this line from `app.py`

Delete all `!pip install` / `!uv pip install` lines.

Your `app.py` must contain only valid Python code.

2 Create (or update) `requirements.txt`

Put your dependencies here instead:

```
openai-agents==0.4.2
gradio==5.49.1
```

If you use anything else, add it too (e.g. `requests`, `python-dotenv`, etc.).

3 Minimal correct file structure

Your Space repo should look like this:

```
.
├── app.py
└── requirements.txt
└── README.md      (optional)
```

HF will:

- Read `requirements.txt`
- Install dependencies automatically
- Then run `app.py`

4 Example `app.py` header (clean & safe)

```
import os import gradio as gr from agents import Agent from agents.tools
import WebSearchTool OPENAI_API_KEY = os.getenv("OPENAI_API_KEY") if not
OPENAI_API_KEY: raise RuntimeError("OPENAI_API_KEY not found. Add it in
Space Settings → Secrets.")
```

↑ No installs. No secrets hardcoded. This is exactly what HF expects.

5 Trigger a rebuild

After:

- removing the `!uv pip install` line
- adding `requirements.txt`

→ Commit & push (or save via the HF web editor).

The Space will rebuild automatically.

Why this is a common trap

You probably developed locally or in Colab/Jupyter, where `!pip install` is normal.

Spaces = production Python, so installs must move to `requirements.txt`.

If you want, paste:

- your full `app.py`
- or your repo file list

and I'll sanity-check it so it 100% boots on Hugging Face Spaces before you resubmit.

You said:

```
!uv pip install openai-agents==0.4.2 gradio==5.49.1
from agents import Agent, Runner, FileSearchTool, WebSearchTool
```

```
from google.colab import userdata
import os
```

```
OPENAI_API_KEY = userdata.get("OPENAI_API_KEY")
os.environ["OPENAI_API_KEY"] = OPENAI_API_KEY
```

```
from agents import function_tool
```

```
@function_tool
def get_bytes_cafe_menu(date: str) -> any:
    return {
        f"{date}": {
            "daily byte": {
```

```
        "name": "Steak Quesadilla",
        "price": 12,
        "description": "Flank steak, mixed cheese in a flour tortilla served with air fried potatoes, sour cream and salsa",
    },
    "vegetarian": {
        "name": "Impossible Quesadilla",
        "price": 12,
        "description": "Impossible plant based product, mixed cheese in a flour tortilla served with air fried potatoes, sour cream and salsa",
    },
    "international": {
        "name": "Chicken Curry",
        "price": 12,
        "description": "Chicken thighs, onion, carrot, potato, curry sauce served over rice",
    },
}
}
```

```
cafe_agent = Agent(
    name="Cafe Agent",
    instructions="You help students locate and provide information about the Bytes Cafe.",
    tools=[
        get_bytes_cafe_menu,
    ],
)
```

```
VECTOR_STORE_ID = "vs_697404ec396081919d426032714e23c8"
```

```
building_agent = Agent(
    name="Building Agent",
    instructions="You help students locate and provide information about buildings and rooms on campus. Be descriptive when giving locations.",
    tools=[
        FileSearchTool(
            max_num_results=3,
            vector_store_ids=[VECTOR_STORE_ID],
            include_search_results=True,
```

```
[    ]  
[    ],  
)
```

```
course_agent = Agent(  
    name="Course Agent",  
    instructions="You help students find out information about courses held at DigiPen.",  
    tools=[  
        FileSearchTool(  
            max_num_results=5,  
            vector_store_ids=[VECTOR_STORE_ID],  
            include_search_results=True,  
        )  
    ],  
)
```

```
handbook_agent = Agent(  
    name="Handbook Agent",  
    instructions="You help students navigate the school handbook, providing information  
about campus policies and student conduct.",  
    tools=[  
        FileSearchTool(  
            max_num_results=5,  
            vector_store_ids=[VECTOR_STORE_ID],  
            include_search_results=True,  
        )  
    ],  
)
```

```
counseling_service_agent = Agent(  
    name="Counseling Service Agent",  
    instructions=(  
        "You assist DigiPen students by providing clear, supportive information about the  
DigiPen Counseling Center. "  
        "You explain available mental health services, eligibility, appointment scheduling,  
confidentiality policies, "  
        "crisis support options, session limits, and how to contact the Counseling Center. "  
        "You do NOT provide therapy or medical advice."  
    ),  
    tools=[
```

```
    WebSearchTool()
    ],
)
agent = Agent(
    name="DigiPen Campus Agent",
    instructions="You are a helpful campus agent that can plan and execute tasks for
students at DigiPen. Please be concise and accurate in handing off tasks to other
agents as needed. If the user mentions stress, anxiety, overwhelmed, burnout, panic,
depression, loneliness, homesick, can't cope, or asks for emotional support → route to
Counseling Service Agent.",
    handoffs=[building_agent, course_agent, handbook_agent, cafe_agent,
counseling_service_agent],
)
from gradio import ChatMessage

async def chat_with_agent(user_msg: str, history: list):
    messages = [{"role": msg["role"], "content": msg["content"]} for msg in history]
    messages.append({"role": "user", "content": user_msg})
    responses = []
    reply_created = False
    active_agent = None

    result = Runner.run_streamed(agent, messages)
    async for event in result.stream_events():
        if event.type == "raw_response_event":
            if event.data.type == "response.output_text.delta":
                if not reply_created:
                    responses.append(ChatMessage(role="assistant", content=""))
                    reply_created = True
                responses[-1].content += event.data.delta
        elif event.type == "agent_updated_stream_event":
            active_agent = event.new_agent.name
            responses.append(
                ChatMessage(
                    content=event.new_agent.name,
                    metadata={"title": "Agent Now Running", "id": active_agent},
                )
            )
```

```
        elif event.type == "run_item_stream_event":
            if event.item.type == "tool_call_item":
                if event.item.raw_item.type == "file_search_call":
                    responses.append(
                        ChatMessage(
                            content=f"Query used: {event.item.raw_item.queries}",
                            metadata={
                                "title": "File Search Completed",
                                "parent_id": active_agent,
                            },
                        )
                    )
                else:
                    tool_name = getattr(event.item.raw_item, "name", "unknown_tool")
                    tool_args = getattr(event.item.raw_item, "arguments", {})
                    responses.append(
                        ChatMessage(
                            content=f"Calling tool {tool_name} with arguments {tool_args}",
                            metadata={"title": "Tool Call", "parent_id": active_agent},
                        )
                    )
            if event.item.type == "tool_call_output_item":
                responses.append(
                    ChatMessage(
                        content=f"Tool output: '{event.item.raw_item['output']}'",
                        metadata={"title": "Tool Output", "parent_id": active_agent},
                    )
                )
            if event.item.type == "handoff_call_item":
                responses.append(
                    ChatMessage(
                        content=f"Name: {event.item.raw_item.name}",
                        metadata={
                            "title": "Handing Off Request",
                            "parent_id": active_agent,
                        },
                    )
                )
        yield responses
```

```

import gradio as gr

demo = gr.ChatInterface(
    chat_with_agent,
    title="DigiPen Campus Agent",
    theme=gr.themes.Soft(
        primary_hue="red", secondary_hue="slate", font=[gr.themes.GoogleFont("Inter")]
    ),
    examples=[
        "I'm trying to find the WANIC classrooms. Can you help?",
        "What's the policy for eating in auditoriums?",
        "What's today's vegetarian dish at the Bytes Cafe?",
        "What are the prerequisites for FLM201?"
    ],
    submit_btn=True,
    flagging_mode="manual",
    flagging_options=["Like", "Spam", "Inappropriate", "Other"],
    type="messages",
    save_history=False,
)

```

```

if __name__ == "__main__":
    demo.launch()

```

change this to make it work in app.py

ChatGPT said:

Thought for 6s

Absolutely — the main things that break on Hugging Face Spaces in your current script
are:

- !uv pip install ... (not valid Python in app.py)
- from google.colab import userdata (Colab-only)
- Hardcoding/setting OPENAI_API_KEY from Colab userdata (Spaces uses Secrets → env vars)
- Your FileSearchTool will crash if the vector store ID doesn't exist in the org/project used by the Space (you hit that earlier). So we should fallback gracefully when VECTOR_STORE_ID isn't set or isn't available.

Below is a drop-in app.py version that will run on Spaces.

✓ app.py (Spaces-ready)

```
import os import gradio as gr from gradio import ChatMessage from agents
import Agent, Runner from agents import function_tool from agents.tools
import WebSearchTool, FileSearchTool # -----
# Environment (Hugging Face Secrets) # -----
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY") if not OPENAI_API_KEY: raise
RuntimeError( "OPENAI_API_KEY is not set. Add it in your Hugging Face
Space: " "Settings → Variables and secrets → New secret → OPENAI_API_KEY"
) # Optional: put your vector store ID as a Secret/Variable too. # If
missing or invalid, we will fall back to WebSearchTool for those agents.
VECTOR_STORE_ID = os.getenv("VECTOR_STORE_ID") # e.g. "vs_..."
USE_VECTOR_STORE = bool(VECTOR_STORE_ID) # -----
# Tools # -----
@function_tool def
get_bytes_cafe_menu(date: str) -> any: # Demo/stub data (no web needed)
return { f"{date}": { "daily byte": { "name": "Steak Quesadilla", "price": 12, "description": "Flank steak, mixed cheese in a flour tortilla served with air fried potatoes, sour cream and salsa", }, "vegetarian": { "name": "Impossible Quesadilla", "price": 12, "description": "Impossible plant based product, mixed cheese in a flour tortilla served with air fried potatoes, sour cream and salsa", }, "international": { "name": "Chicken Curry", "price": 12, "description": "Chicken thighs, onion, carrot, potato, curry sauce served over rice", } } } def
make_file_search_tool(max_num_results: int): """ Returns a FileSearchTool
if VECTOR_STORE_ID is provided, otherwise a WebSearchTool fallback. This
prevents your Space from crashing when the vector store isn't available.
"""
if USE_VECTOR_STORE: return FileSearchTool(
max_num_results=max_num_results, vector_store_ids=[VECTOR_STORE_ID],
include_search_results=True, ) # Fallback: still works on Spaces without a
vector store return WebSearchTool() # -----
# Sub-agents # ----- cafe_agent = Agent( name="Cafe
```

```
Agent", instructions="You help students locate and provide information about the Bytes Cafe.", tools=[get_bytes_cafe_menu], ) building_agent = Agent( name="Building Agent", instructions=( "You help students locate and provide information about buildings and rooms on campus. " "Be descriptive when giving locations. If you cannot access internal campus maps, say so and " "suggest official DigiPen resources." ), tools=[make_file_search_tool(max_num_results=3)], ) course_agent = Agent( name="Course Agent", instructions=( "You help students find information about DigiPen courses (prerequisites, descriptions, etc.). " "If internal course data isn't available, use official DigiPen web pages." ), tools=[make_file_search_tool(max_num_results=5)], ) handbook_agent = Agent( name="Handbook Agent", instructions=( "You help students navigate the DigiPen student handbook and policies. " "Prefer official DigiPen sources; quote short excerpts only when necessary." ), tools=[make_file_search_tool(max_num_results=5)], ) counseling_service_agent = Agent( name="Counseling Service Agent", instructions=( "You assist DigiPen students by providing clear, supportive information about the DigiPen Counseling Center. " "You explain available services, eligibility, appointment scheduling, confidentiality, and how to contact the Counseling Center. " "You do NOT provide therapy, diagnose conditions, or give medical advice. " "If the user expresses stress, overwhelm, anxiety, burnout, loneliness, or homesickness, prioritize directing them to " "how to book counseling and what options exist." ), tools=[WebSearchTool()], ) # ----- # Main router agent # ----- agent = Agent( name="DigiPen Campus Agent", instructions=( "You are a helpful campus agent that routes requests to the most appropriate specialist agent. " "Be concise and accurate. " "Routing rules: If the user mentions stress, anxious/anxiety, overwhelmed, burnout, panic, depression, loneliness, homesick, " "can't cope, or asks for emotional support → hand off immediately to Counseling Service Agent. " "If the user asks about buildings/rooms → Building Agent. " "Courses/prerequisites → Course Agent. " "Policies/handbook → Handbook
```

```

Agent. " "Bytes Cafe menu → Cafe Agent." ), handoffs=[building_agent,
course_agent, handbook_agent, cafe_agent, counseling_service_agent], ) #
----- # Gradio chat function #

----- async def chat_with_agent(user_msg: str,
history: list): # history comes in as list of {"role": "...", "content":
"} when type="messages" messages = [] for msg in history: # Some
gradio builds pass ChatMessage-like objects; handle both safely role =
msg.get("role") if isinstance(msg, dict) else getattr(msg, "role", None)
content = msg.get("content") if isinstance(msg, dict) else getattr(msg,
"content", None) if role and content is not None: messages.append({"role":
role, "content": content}) messages.append({"role": "user", "content":
user_msg}) responses = [] reply_created = False active_agent = None result
= Runner.run_streamed(agent, messages) async for event in
result.stream_events(): if event.type == "raw_response_event": if
event.data.type == "response.output_text.delta": if not reply_created:
responses.append(ChatMessage(role="assistant", content="")) reply_created
= True responses[-1].content += event.data.delta elif event.type ==
"agent_updated_stream_event": active_agent = event.new_agent.name
responses.append( ChatMessage( content=event.new_agent.name,
metadata={"title": "Agent Now Running", "id": active_agent}, ) ) elif
event.type == "run_item_stream_event": if event.item.type ==
"tool_call_item": raw = event.item.raw_item if getattr(raw, "type", None)
== "file_search_call": responses.append( ChatMessage( content=f"Query
used: {getattr(raw, 'queries', None)}", metadata={"title": "File Search
Completed", "parent_id": active_agent}, ) ) else: tool_name = getattr(raw,
"name", "unknown_tool") tool_args = getattr(raw, "arguments", {})
responses.append( ChatMessage( content=f"Calling tool {tool_name} with
arguments {tool_args}", metadata={"title": "Tool Call", "parent_id":
active_agent}, ) ) elif event.item.type == "tool_call_output_item": # Some
tool outputs are dict-like; keep it safe raw_out = event.item.raw_item
out_text = raw_out.get("output") if isinstance(raw_out, dict) else
str(raw_out) responses.append( ChatMessage( content=f"Tool output:

```

```
{out_text}", metadata={"title": "Tool Output", "parent_id": active_agent},  
) ) elif event.item.type == "handoff_call_item": responses.append(  
ChatMessage( content=f"Handoff to: {event.item.raw_item.name}",  
metadata={"title": "Handing Off Request", "parent_id": active_agent}, ) )  
yield responses # ----- # Gradio UI #  
----- demo = gr.ChatInterface( chat_with_agent,  
title="DigiPen Campus Agent", theme=gr.themes.Soft( primary_hue="red",  
secondary_hue="slate", font=[gr.themes.GoogleFont("Inter")] ), examples=[  
"I'm trying to find the WANIC classrooms. Can you help?", "What's the  
policy for eating in auditoriums?", "What's today's vegetarian dish at the  
Bytes Cafe?", "What are the prerequisites for FLM201?", "I'm stressed with  
GAM541. What should I do?" ], submit_btn=True, flagging_mode="manual",  
flagging_options=["Like", "Spam", "Inappropriate", "Other"],  
type="messages", save_history=False, ) if __name__ == "__main__":  
demo.launch()
```

✓ requirements.txt

Create a `requirements.txt` in your Space repo:

```
openai-agents==0.4.2  
gradio==5.49.1
```

(If your import path `from agents.tools import ...` fails, tell me the exact error and I'll
adjust the imports to match your installed package version.)

✓ Hugging Face Spaces settings

In your Space:

- Settings → Variables and secrets
 - Add Secret: `OPENAI_API_KEY` = your key
 - (Optional) Add `VECTOR_STORE_ID` if you actually have one that works in
that org/project

Why I made VECTOR_STORE optional

Your earlier error was:

vector store ... not found

If that happens again, the code above won't crash — it will use WebSearchTool instead, so your Space still runs and you can submit a URL.

If you want, paste the exact vector store error you get on Spaces (if any), and I'll tell you how to create/use a vector store correctly in that environment (or how to remove it cleanly if your assignment doesn't require it).

You said:

Exit code: 1. Reason: Traceback (most recent call last):
File "/app/app.py", line 5, in <module>
from agents import Agent, Runner
ModuleNotFoundError: No module named 'agents'

ChatGPT said:

Perfect — this error tells us exactly what's wrong, and it's a very common one when moving from notebooks to Spaces.

✖ What the error means

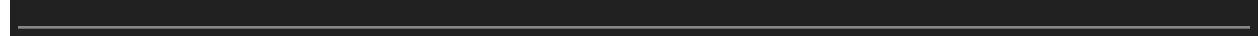
ModuleNotFoundError: No module named 'agents'

Even though you installed:

openai-agents==0.4.2

👉 the Python import name is NOT `agents`.

The package name (`openai-agents`) ≠ the module name.



✓ The correct imports for `openai-agents==0.4.2`

You must import from `openai_agents`, not `agents`.

🔧 Fix these imports in `app.py`

✗ WRONG

```
from agents import Agent, Runner from agents import function_tool from  
agents.tools import WebSearchTool, FileSearchTool
```

✓ CORRECT

```
from openai_agents import Agent, Runner, function_tool from  
openai_agents.tools import WebSearchTool, FileSearchTool
```

That's it. This is the root cause.

✓ Minimal sanity-check `app.py` header (correct)

Your file should start like this:

```
import os import gradio as gr from gradio import ChatMessage from  
openai_agents import Agent, Runner, function_tool from openai_agents.tools  
import WebSearchTool, FileSearchTool
```

And your `requirements.txt` should still be:

```
openai-agents==0.4.2  
gradio==5.49.1
```

Why this happens (quick explanation)

Thing	Name
PyPI package	<code>openai-agents</code>

Python module

openai_agents

✗ Non-existent

agents

Colab sometimes hides this mistake if you had a local folder named `agents`/ earlier —

Hugging Face Spaces does not.

After fixing

1. Update imports
 2. Commit & push (or save in HF editor)
 3. Space rebuilds automatically
 4. Error disappears
-

If you hit another error

Paste it directly.

At this point you're past the hard part — this is now just import/version cleanup, not architecture issues 💪

You said:

```
Exit code: 1. Reason: Traceback (most recent call last):  
  File "/app/app.py", line 5, in <module>  
    from openai_agents import Agent, Runner, function_tool  
ModuleNotFoundError: No module named 'openai_agents'
```

ChatGPT said:

Great, thanks for the traceback — and important correction here (this one's on me 👍).

✓ What's ACTUALLY going on (and why this keeps failing)

You are using OpenAI Agents SDK v0.4.2.

For that SDK:

- PyPI package name: `openai-agents`

- Python import name: ✓ `agents`
- ✗ `openai_agents` does NOT exist

So this error:

```
ModuleNotFoundError: No module named 'openai_agents'
```

means the previous import change was incorrect.

👉 Your original imports were correct, *but* the package is currently not being installed in the Space.

✓ The REAL fix (2 steps)

✓ 1 Use the correct imports (go back to `agents`)

In `app.py`, use this:

```
from agents import Agent, Runner, function_tool from agents.tools import  
WebSearchTool, FileSearchTool
```

✗ Do NOT use `openai_agents`

✓ 2 Make sure `requirements.txt` is present and correct

Your Space must have a `requirements.txt` at the repo root:

```
openai-agents==0.4.2  
gradio==5.49.1
```

📌 Common mistake:

- File named `requirement.txt` ✗
- File inside a subfolder ✗
- File created but not committed ✗

HF Spaces only installs dependencies from root-level `requirements.txt`.

✓ Final minimal file layout (this matters)

Your Space repo should look exactly like this:

```
/  
└── app.py  
└── requirements.txt  
└── README.md    (optional)
```

No venv/, no notebooks, no extra folders.

✓ Sanity-check: minimal working header

Put this at the top of app.py:

```
import os import gradio as gr from gradio import ChatMessage from agents  
import Agent, Runner, function_tool from agents.tools import  
WebSearchTool, FileSearchTool OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")  
if not OPENAI_API_KEY: raise RuntimeError("OPENAI_API_KEY missing. Add it  
in Space Settings → Secrets.")
```

If this still errors, then the package is not being installed, which means HF isn't seeing requirements.txt.

🔍 How to confirm installation on HF

In the Space:

1. Go to Logs
2. During build, you should see something like:

```
Installing collected packages: openai-agents, gradio  
Successfully installed openai-agents-0.4.2
```

- 3.

If you don't see that, HF didn't detect requirements.txt.

Why this was confusing (totally fair)

Thing	Name
PyPI package	openai-agents
Python module	agents
✗ Non-existent	openai_agents

Many libraries do the opposite, so this is a classic trap.

What to do now (checklist)

- Revert imports to `from agents import ...`
- Ensure `requirements.txt` exists at repo root
- Commit & push (or save in HF editor)
- Wait for rebuild
- Check logs for install confirmation

If it still fails, paste:

- your repo file list
- or a screenshot / copy of the build log

and I'll zero in on it immediately.

You said:

where should I edit requirements.txt

ChatGPT said:

Short answer: in the root of your Hugging Face Space repo, right next to `app.py`.

Here's the step-by-step, depending on how you're editing your Space.

Option A (easiest): Edit on Hugging Face website

1. Open your Hugging Face Space

2. Click the Files tab

3. You should see something like:

```
app.py
```

```
README.md
```

4.

5. Click “Add file” → “Create new file”

6. Name it exactly:

```
requirements.txt
```

7.

8. Paste this inside:

```
openai-agents==0.4.2
```

```
gradio==5.49.1
```

9.

10. Click Commit changes

📌 The file must be:

- named requirements.txt (plural)
- all lowercase
- not inside a folder

Option B: Edit locally (git workflow)

If you cloned the Space repo:

```
cd your-space-repo touch requirements.txt
```

Edit it:

```
openai-agents==0.4.2
```

```
gradio==5.49.1
```

Then:

```
git add requirements.txt git commit -m "Add requirements for HF Spaces"
```

```
git push
```

HF will automatically rebuild the Space.

✓ How to verify it worked

After committing:

1. Go to your Space → Logs
2. During build, you should see:

```
Installing collected packages: openai-agents, gradio
Successfully installed openai-agents-0.4.2
```

3.

If you don't see that, HF didn't detect the file (wrong name or location).

🚫 Common mistakes (worth double-checking)

Mistake	Result
Requirement.txt	✗ Ignored
requirements.TXT	✗ Ignored
requirements.txt inside /src	✗ Ignored
Added but not committed	✗ Ignored

🔥 One last sanity check

Your Files tab should look like:

```
📁 (root)
├── app.py
├── requirements.txt
└── README.md (optional)
```

If you want, send me a screenshot or list of files from the Files tab and I'll confirm it's correct before you rebuild.

You said:

```
Exit code: 1. Reason: Traceback (most recent call last):
  File "/app/app.py", line 6, in <module>
    from agents.tools import WebSearchTool, FileSearchTool
ModuleNotFoundError: No module named 'agents.tools'
```

ChatGPT said:

```
Thought for 4 seconds
```

You're *one import* away from it working.

In openai-agents 0.4.2, the tools are exported from the top-level `agents` package, not `agents.tools`. The official docs show:

```
from agents import Agent, FileSearchTool, Runner, WebSearchTool
```

So this line is wrong in Spaces:

```
from agents.tools import WebSearchTool, FileSearchTool
```

 **Fix your imports (top of app.py)**

Replace your first imports with this:

```
import os import gradio as gr from gradio import ChatMessage from agents
import Agent, Runner, function_tool, WebSearchTool, FileSearchTool
```

...and delete the `from agents.tools ...` line completely.

That should remove:

```
ModuleNotFoundError: No module named 'agents.tools'
```

 **Reminder:** requirements.txt (**root of the Space**)

Make sure you have this in the Space root:

```
openai-agents==0.4.2
gradio==5.49.1
```

✓ Also remove Colab-only code in app.py

Delete these in Spaces (they will break):

```
from google.colab import userdata OPENAI_API_KEY =  
userdata.get("OPENAI_API_KEY")
```

Use env var instead:

```
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY") if not OPENAI_API_KEY: raise  
RuntimeError("Missing OPENAI_API_KEY. Add it in Space Settings →  
Secrets.")
```

If you paste your updated top ~15 lines of app.py, I'll sanity-check it so the Space boots cleanly.

You said:

Exit code: 1. Reason: line 586, in cache

```
prediction = await self.root_block.process_api(  
    ^^^^^^^^^^^^^^^^^^^^^^^^^^  
...<3 lines>...  
)  
^
```

File "/usr/local/lib/python3.13/site-packages/gradio/blocks.py", line 2260, in process_api

```
    data = await self.postprocess_data(block_fn, result["prediction"], state)  
    ^^^^^^^^^^^^^^^^^^^^^^
```

File "/usr/local/lib/python3.13/site-packages/gradio/blocks.py", line 2038, in postprocess_data

```
    prediction_value = block.postprocess(prediction_value)  
File "/usr/local/lib/python3.13/site-packages/gradio/components/chatbot.py", line 637,  
in postprocess  
    self._postprocess_message_messages(cast(MessageDict, message))  
    ~~~~~^
```

File "/usr/local/lib/python3.13/site-packages/gradio/components/chatbot.py", line 593, in _postprocess_message_messages

```
    message["content"] = self._postprocess_content(message["content"])  
    ~~~~~^
```

File "/usr/local/lib/python3.13/site-packages/gradio/components/chatbot.py", line 573, in _postprocess_content

```
    return self._create_file_message(chat_message, filepath)
```

```
~~~~~^~~~~~  
File "/usr/local/lib/python3.13/site-packages/gradio/components/chatbot.py", line 532,  
in _create_file_message  
return FileMessage(  
    file=FileData(path=filepath, mime_type=mime_type),  
    alt_text=Chatbot._get_alt_text(chat_message),  
)  
File "/usr/local/lib/python3.13/site-packages/pydantic/main.py", line 253, in __init__  
    validated_self = self._pydantic_validator_.validate_python(data, self_instance=self)  
pydantic_core._pydantic_core.ValidationError: 1 validation error for FileMessage  
alt_text  
Input should be a valid string [type=string_type],  
input_value=ChatMessage(content='Call...pus Agent'), options=[],  
input_type=ChatMessage]  
For further information visit https://errors.pydantic.dev/2.11/v/string\_type  
Container logs:
```

```
===== Application Startup at 2026-01-25 08:06:52 =====
```

```
* Running on local URL: http://0.0.0.0:7860, with SSR ⚡ (experimental, to disable set  
ssr_mode=False in launch())  
Caching examples at: '/app/.gradio/cached_examples/16'  
Traceback (most recent call last):  
File "/app/app.py", line 239, in <module>  
    demo.launch()  
~~~~~^~  
File "/usr/local/lib/python3.13/site-packages/gradio(blocks.py", line 2819, in launch  
    raise Exception(  
        f"Couldn't start the app because '{resp.url}' failed (code {resp.status_code}). Check  
your network or proxy settings to ensure localhost is accessible."  
    )  
Exception: Couldn't start the app because  
'http://localhost:7860/gradio_api/startup-events' failed (code 500). Check your network  
or proxy settings to ensure localhost is accessible.  
ERROR: Exception in ASGI application  
Traceback (most recent call last):  
File "/usr/local/lib/python3.13/site-packages/uvicorn/protocols/http/h11_impl.py", line  
410, in run_asgi  
    result = await app( # type: ignore[func-returns-value]  
~~~~~^~~~~~
```

```
    self.scope, self.receive, self.send
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
)
^
File "/usr/local/lib/python3.13/site-packages/uvicorn/middleware/proxy_headers.py",
line 60, in __call__
    return await self.app(scope, receive, send)
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
File "/usr/local/lib/python3.13/site-packages/fastapi/applications.py", line 1135, in
call_
    await super().__call__(scope, receive, send)
File "/usr/local/lib/python3.13/site-packages/starlette/applications.py", line 107, in
call_
    await self.middleware_stack(scope, receive, send)
File "/usr/local/lib/python3.13/site-packages/starlette/middleware/errors.py", line 186, in
call_
    raise exc
File "/usr/local/lib/python3.13/site-packages/starlette/middleware/errors.py", line 164, in
call_
    await self.app(scope, receive, _send)
File "/usr/local/lib/python3.13/site-packages/starlette/middleware/base.py", line 191, in
call_
    with recv_stream, send_stream, collapse_excgroups():
    ~~~~~^
File "/usr/local/lib/python3.13/contextlib.py", line 162, in __exit__
    self.gen.throw(value)
    ~~~~~^
File "/usr/local/lib/python3.13/site-packages/starlette/_utils.py", line 85, in
collapse_excgroups
    raise exc
File "/usr/local/lib/python3.13/site-packages/starlette/middleware/base.py", line 193, in
call_
    response = await self.dispatch_func(request, call_next)
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
File "/usr/local/lib/python3.13/site-packages/gradio/routes.py", line 453, in
conditional_routing_middleware
    response = await call_next(request)
    ^^^^^^^^^^^^^^
File "/usr/local/lib/python3.13/site-packages/starlette/middleware/base.py", line 168, in
call_next
```

```
raise app_exc from app_exc.cause or app_exc.context
File "/usr/local/lib/python3.13/site-packages/starlette/middleware/base.py", line 144, in
coro
    await self.app(scope, receive_or_disconnect, send_no_error)
File "/usr/local/lib/python3.13/site-packages/gradio/brotli_middleware.py", line 74, in
call
    return await self.app(scope, receive, send)
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
File "/usr/local/lib/python3.13/site-packages/gradio/route_utils.py", line 878, in __call__
    await self.app(scope, receive, send)
File "/usr/local/lib/python3.13/site-packages/starlette/middleware/exceptions.py", line
63, in __call__
    await wrap_app_handling_exceptions(self.app, conn)(scope, receive, send)
File "/usr/local/lib/python3.13/site-packages/starlette/_exception_handler.py", line 53,
in wrapped_app
    raise exc
File "/usr/local/lib/python3.13/site-packages/starlette/_exception_handler.py", line 42,
in wrapped_app
    await app(scope, receive, sender)
File "/usr/local/lib/python3.13/site-packages/fastapi/middleware/asyncexitstack.py", line
18, in __call__
    await self.app(scope, receive, send)
File "/usr/local/lib/python3.13/site-packages/starlette/routing.py", line 716, in __call__
    await self.middleware_stack(scope, receive, send)
File "/usr/local/lib/python3.13/site-packages/starlette/routing.py", line 736, in app
    await route.handle(scope, receive, send)
File "/usr/local/lib/python3.13/site-packages/starlette/routing.py", line 290, in handle
    await self.app(scope, receive, send)
File "/usr/local/lib/python3.13/site-packages/fastapi/routing.py", line 115, in app
    await wrap_app_handling_exceptions(app, request)(scope, receive, send)
File "/usr/local/lib/python3.13/site-packages/starlette/_exception_handler.py", line 53,
in wrapped_app
    raise exc
File "/usr/local/lib/python3.13/site-packages/starlette/_exception_handler.py", line 42,
in wrapped_app
    await app(scope, receive, sender)
File "/usr/local/lib/python3.13/site-packages/fastapi/routing.py", line 101, in app
    response = await f(request)
    ^^^^^^^^^^^^^^
File "/usr/local/lib/python3.13/site-packages/fastapi/routing.py", line 355, in app
```

```
raw_response = await run_endpoint_function(  
    ^^^^^^^^^^^^^^^^^^^^^  
    ...<3 lines>...  
)  
^  
File "/usr/local/lib/python3.13/site-packages/fastapi/routing.py", line 243, in  
run_endpoint_function  
    return await dependant.call(**values)  
    ^^^^^^^^^^^^^^^^^  
File "/usr/local/lib/python3.13/site-packages/gradio/routes.py", line 1720, in  
startup_events  
    await app.get_blocks().run_extra_startup_events()  
File "/usr/local/lib/python3.13/site-packages/gradio(blocks.py", line 3143, in  
run_extra_startup_events  
    await startup_event()  
File "/usr/local/lib/python3.13/site-packages/gradio/helpers.py", line 520, in  
start_caching  
    await self.cache()  
File "/usr/local/lib/python3.13/site-packages/gradio/helpers.py", line 586, in cache  
    prediction = await self.root_block.process_api(  
        ^^^^^^^^^  
    ...<3 lines>...  
)  
^  
File "/usr/local/lib/python3.13/site-packages/gradio(blocks.py", line 2260, in  
process_api  
    data = await self.postprocess_data(block_fn, result["prediction"], state)  
    ^^^^^^^^^  
File "/usr/local/lib/python3.13/site-packages/gradio(blocks.py", line 2038, in  
postprocess_data  
    prediction_value = block.postprocess(prediction_value)  
File "/usr/local/lib/python3.13/site-packages/gradio/components/chatbot.py", line 637,  
in postprocess  
    self._postprocess_message_messages(cast(MessageDict, message))  
    ~~~~~  
File "/usr/local/lib/python3.13/site-packages/gradio/components/chatbot.py", line 593,  
in _postprocess_message_messages  
    message["content"] = self._postprocess_content(message["content"])  
    ~~~~~
```

```
File "/usr/local/lib/python3.13/site-packages/gradio/components/chatbot.py", line 573,  
in __postprocess_content  
    return self._create_file_message(chat_message, filepath)  
~~~~~  
File "/usr/local/lib/python3.13/site-packages/gradio/components/chatbot.py", line 532,  
in _create_file_message  
    return FileMessage(  
        file=FileData(path=filepath, mime_type=mime_type),  
        alt_text=Chatbot.get_alt_text(chat_message),  
    )  
File "/usr/local/lib/python3.13/site-packages/pydantic/main.py", line 253, in __init__  
    validated_self = self.pydantic_validator.validate_python(data, self_instance=self)  
pydantic_core.pydantic_core.ValidationError: 1 validation error for FileMessage  
alt_text  
    Input should be a valid string [type=string_type],  
    input_value=ChatMessage(content='Call...pus Agent'), options=[],  
    input_type=ChatMessage]  
For further information visit https://errors.pydantic.dev/2.11/v/string\_type
```

ChatGPT said:
Thought for 14s

This crash is coming from Gradio's example caching at startup. It runs your `chat_with_agent()` once, then tries to postprocess what you yielded. One of the messages ends up with `content` being a `ChatMessage` object instead of a string, and Gradio's Chatbot thinks it's a file message → Pydantic throws:

alt text Input should be a valid string ... input value=ChatMessage(...)

 **Fix (recommended): Stop using ChatMessage objects, yield plain dicts + disable example caching**

1) Remove this line

```
from gradio import ChatMessage
```

2) Update your streaming function to yield plain dict messages

Replace every `ChatMessage(...)` with dicts like:

```
# instead of: ChatMessage(role="assistant", content="")
responses.append({"role": "assistant", "content": ""}) # instead of:
ChatMessage(content=..., metadata=...) responses.append({ "role":
"assistant", "content": event.new_agent.name, "metadata": {"title": "Agent
Now Running", "id": active_agent}, })
```

Here's a clean version of your event-handling section (drop-in):

```
async def chat_with_agent(user_msg: str, history: list): messages = [] for
msg in history: # history entries are dicts when type="messages" if
isinstance(msg, dict) and "role" in msg and "content" in msg:
messages.append({"role": msg["role"], "content": msg["content"]})
messages.append({"role": "user", "content": user_msg}) responses = []
reply_created = False active_agent = None result =
Runner.run_streamed(agent, messages) async for event in
result.stream_events(): if event.type == "raw_response_event" and
event.data.type == "response.output_text.delta": if not reply_created:
responses.append({"role": "assistant", "content": ""}) reply_created =
True responses[-1]["content"] += event.data.delta elif event.type ==
"agent_updated_stream_event": active_agent = event.new_agent.name
responses.append({ "role": "assistant", "content": f"Running:
{active_agent}", "metadata": {"title": "Agent Now Running", "id": active_agent}, }) elif event.type == "run_item_stream_event": if
event.item.type == "tool_call_item": raw = event.item.raw_item tool_type =
getattr(raw, "type", None) if tool_type == "file_search_call":
responses.append({ "role": "assistant", "content": f"File search query:
{getattr(raw, 'queries', None)}", "metadata": {"title": "File Search
Completed", "parent_id": active_agent}, }) else: tool_name = getattr(raw,
"name", "unknown_tool") tool_args = getattr(raw, "arguments", {})
responses.append({ "role": "assistant", "content": f"Calling tool
{tool_name} with arguments {tool_args}", "metadata": {"title": "Tool
Call", "parent_id": active_agent}, }) elif event.item.type ==
"tool_call_output_item": raw_out = event.item.raw_item out_text =
```

```
raw_out.get("output") if isinstance(raw_out, dict) else str(raw_out)
responses.append({ "role": "assistant", "content": f"Tool output: {out_text}", "metadata": { "title": "Tool Output", "parent_id": active_agent}, })
elif event.item.type == "handoff_call_item":
responses.append({ "role": "assistant", "content": f"Handoff to: {event.item.raw_item.name}", "metadata": { "title": "Handing Off Request", "parent_id": active_agent}, })
yield responses
```

3) Disable caching examples (this is what triggers the crash during startup)

In your `gr.ChatInterface(...)`, add:

```
cache_examples=False
```

So:

```
demo = gr.ChatInterface( chat_with_agent, title="DigiPen Campus Agent",
... examples=[ ... ], cache_examples=False, type="messages",
save_history=False, )
```

Why this works

- Gradio's `type="messages"` is happiest when you return lists of dicts `({"role", "content"})`.
- Disabling `cache_examples` prevents Gradio from executing your generator during startup, which is where your error is happening.

If you paste your current `chat_with_agent()` after edits, I'll quickly check that there's zero chance of a nested `ChatMessage` slipping into `content`.

You said:

```
import os
import gradio as gr
from gradio import ChatMessage
```

```
from agents import Agent, Runner, function_tool, WebSearchTool, FileSearchTool
```

```
# -----
# Environment (Hugging Face Secrets)
# -----
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
if not OPENAI_API_KEY:
    raise RuntimeError(
        "OPENAI_API_KEY is not set. Add it in your Hugging Face Space: "
        "Settings → Variables and secrets → New secret → OPENAI_API_KEY"
    )
```

```
# Optional: put your vector store ID as a Secret/Variable too.
# If missing or invalid, we will fall back to WebSearchTool for those agents.
VECTOR_STORE_ID = os.getenv("VECTOR_STORE_ID") # e.g. "vs_..."
USE_VECTOR_STORE = bool(VECTOR_STORE_ID)
```

```
# -----
# Tools
# -----
@function_tool
def get_bytes_cafe_menu(date: str) -> any:
    # Demo/stub data (no web needed)
    return {
        f'{date}': {
            "daily byte": {
                "name": "Steak Quesadilla",
                "price": 12,
                "description": "Flank steak, mixed cheese in a flour tortilla served with air fried potatoes, sour cream and salsa",
            },
            "vegetarian": {
                "name": "Impossible Quesadilla",
                "price": 12,
                "description": "Impossible plant based product, mixed cheese in a flour tortilla served with air fried potatoes, sour cream and salsa",
            },
            "international": {
                "name": "Chicken Curry",
                "price": 12,
```

```
        "description": "Chicken thighs, onion, carrot, potato, curry sauce served over
rice",
    },
}
}
```

```
def make_file_search_tool(max_num_results: int):
    """
    Returns a FileSearchTool if VECTOR_STORE_ID is provided, otherwise a
    WebSearchTool fallback.
    This prevents your Space from crashing when the vector store isn't available.
    """
    if USE_VECTOR_STORE:
        return FileSearchTool(
            max_num_results=max_num_results,
            vector_store_ids=[VECTOR_STORE_ID],
            include_search_results=True,
        )
    # Fallback: still works on Spaces without a vector store
    return WebSearchTool()
```

```
# -----
# Sub-agents
# -----
cafe_agent = Agent(
    name="Cafe Agent",
    instructions="You help students locate and provide information about the Bytes
Cafe.",
    tools=[get_bytes_cafe_menu],
)

building_agent = Agent(
    name="Building Agent",
    instructions=(
        "You help students locate and provide information about buildings and rooms on
campus. "
        "Be descriptive when giving locations. If you cannot access internal campus maps,
say so and "
    )
)
```

```
    "suggest official DigiPen resources."
),
tools=[make_file_search_tool(max_num_results=3)],
)

course_agent = Agent(
    name="Course Agent",
    instructions=(  

        "You help students find information about DigiPen courses (prerequisites,  

descriptions, etc.)."  

        "If internal course data isn't available, use official DigiPen web pages."  

),
tools=[make_file_search_tool(max_num_results=5)],
)

handbook_agent = Agent(  

    name="Handbook Agent",
    instructions=(  

        "You help students navigate the DigiPen student handbook and policies. "  

        "Prefer official DigiPen sources; quote short excerpts only when necessary."  

),
tools=[make_file_search_tool(max_num_results=5)],
)

counseling_service_agent = Agent(  

    name="Counseling Service Agent",
    instructions=(  

        "You assist DigiPen students by providing clear, supportive information about the  

DigiPen Counseling Center. "  

        "You explain available services, eligibility, appointment scheduling, confidentiality,  

and how to contact the Counseling Center. "  

        "You do NOT provide therapy, diagnose conditions, or give medical advice. "  

        "If the user expresses stress, overwhelm, anxiety, burnout, loneliness, or  

homesickness, prioritize directing them to "  

        "how to book counseling and what options exist."  

),
tools=[WebSearchTool()],
)
```

```

# -----
# Main router agent
# -----
agent = Agent(
    name="DigiPen Campus Agent",
    instructions=([
        "You are a helpful campus agent that routes requests to the most appropriate specialist agent. ",
        "Be concise and accurate. ",
        "Routing rules: If the user mentions stress, anxious/anxiety, overwhelmed, burnout, panic, depression, loneliness, homesick, ",
        "can't cope, or asks for emotional support → hand off immediately to Counseling Service Agent. ",
        "If the user asks about buildings/rooms → Building Agent. ",
        "Courses/prerequisites → Course Agent. ",
        "Policies/handbook → Handbook Agent. ",
        "Bytes Cafe menu → Cafe Agent."
    ]),
    handoffs=[building_agent, course_agent, handbook_agent, cafe_agent,
counseling_service_agent],
)

```

```

# -----
# Gradio chat function
# -----
async def chat_with_agent(user_msg: str, history: list):
    # history comes in as list of {"role": "...", "content": "..."} when type="messages"
    messages = []
    for msg in history:
        # Some gradio builds pass ChatMessage-like objects; handle both safely
        role = msg.get("role") if isinstance(msg, dict) else getattr(msg, "role", None)
        content = msg.get("content") if isinstance(msg, dict) else getattr(msg, "content",
None)
        if role and content is not None:
            messages.append({"role": role, "content": content})
    messages.append({"role": "user", "content": user_msg})
    responses = []

```

```
reply_created = False
active_agent = None

result = Runner.run_streamed(agent, messages)

async for event in result.stream_events():
    if event.type == "raw_response_event":
        if event.data.type == "response.output_text.delta":
            if not reply_created:
                responses.append(ChatMessage(role="assistant", content=""))
                reply_created = True
            responses[-1].content += event.data.delta

    elif event.type == "agent_updated_stream_event":
        active_agent = event.new_agent.name
        responses.append(
            ChatMessage(
                content=event.new_agent.name,
                metadata={"title": "Agent Now Running", "id": active_agent},
            )
        )

    elif event.type == "run_item_stream_event":
        if event.item.type == "tool_call_item":
            raw = event.item.raw_item

            if getattr(raw, "type", None) == "file_search_call":
                responses.append(
                    ChatMessage(
                        content=f"Query used: {getattr(raw, 'queries', None)}",
                        metadata={"title": "File Search Completed", "parent_id": active_agent},
                    )
                )
            else:
                tool_name = getattr(raw, "name", "unknown_tool")
                tool_args = getattr(raw, "arguments", {})
                responses.append(
                    ChatMessage(
                        content=f"Calling tool {tool_name} with arguments {tool_args}",
                        metadata={"title": "Tool Call", "parent_id": active_agent},
                    )
                )

    else:
        print(f"Unknown event type: {event.type}")
```

```

        )
    )

    elif event.item.type == "tool_call_output_item":
        # Some tool outputs are dict-like; keep it safe
        raw_out = event.item.raw_item
        out_text = raw_out.get("output") if isinstance(raw_out, dict) else str(raw_out)
        responses.append(
            ChatMessage(
                content=f"Tool output: {out_text}",
                metadata={"title": "Tool Output", "parent_id": active_agent},
            )
        )

    elif event.item.type == "handoff_call_item":
        responses.append(
            ChatMessage(
                content=f"Handoff to: {event.item.raw_item.name}",
                metadata={"title": "Handing Off Request", "parent_id": active_agent},
            )
        )

yield responses

```

```

# -----
# Gradio UI
# -----
demo = gr.ChatInterface(
    chat_with_agent,
    title="DigiPen Campus Agent",
    theme=gr.themes.Soft(
        primary_hue="red", secondary_hue="slate", font=[gr.themes.GoogleFont("Inter")]
    ),
    examples=[
        "I'm trying to find the WANIC classrooms. Can you help?",
        "What's the policy for eating in auditoriums?",
        "What's today's vegetarian dish at the Bytes Cafe?",
        "What are the prerequisites for FLM201?",
        "I'm stressed with GAM541. What should I do?"
    ]
)

```

```

    ],
    submit_btn=True,
    flagging_mode="manual",
    flagging_options=["Like", "Spam", "Inappropriate", "Other"],
    type="messages",
    save_history=False,
)

if name == "main":
    demo.launch()

```

please update the whole thing

ChatGPT said:

Here's a fully updated working `app.py` version based on your code, with the two key fixes:

1. No `ChatMessage` objects (use plain dict messages only) → avoids the Pydantic `FileMessage.alt_text` crash
2. Disable example caching (`cache_examples=False`) → prevents Gradio from running your generator at startup

Copy-paste this whole file into `app.py`.

```

import os import gradio as gr from agents import Agent, Runner,
function_tool, WebSearchTool, FileSearchTool #

----- # Environment (Hugging Face Secrets) #
----- OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")

if not OPENAI_API_KEY: raise RuntimeError( "OPENAI_API_KEY is not set. Add it in your Hugging Face Space: " "Settings → Variables and secrets → New secret → OPENAI_API_KEY" ) # Optional: Put your vector store ID as a Secret/Variable too. # If missing or invalid, we will fall back to WebSearchTool for those agents. VECTOR_STORE_ID =
os.getenv("VECTOR_STORE_ID") # e.g. "vs_..." USE_VECTOR_STORE =
bool(VECTOR_STORE_ID) # ----- # Tools #
----- @function_tool def get_bytes_cafe_menu(date: str) -> dict: # Demo/stub data (no web needed) return { f"{date}": {

```

```
"daily byte": { "name": "Steak Quesadilla", "price": 12, "description":  
"Flank steak, mixed cheese in a flour tortilla served with air fried  
potatoes, sour cream and salsa", }, "vegetarian": { "name": "Impossible  
Quesadilla", "price": 12, "description": "Impossible plant based product,  
mixed cheese in a flour tortilla served with air fried potatoes, sour  
cream and salsa", }, "international": { "name": "Chicken Curry", "price":  
12, "description": "Chicken thighs, onion, carrot, potato, curry sauce  
served over rice", } } def make_search_tool(max_num_results: int): """  
Prefer FileSearchTool when VECTOR_STORE_ID exists; otherwise fall back to  
WebSearchTool. This avoids crashing in Spaces when no vector store is  
configured. """ if USE_VECTOR_STORE: return FileSearchTool(  
max_num_results=max_num_results, vector_store_ids=[VECTOR_STORE_ID],  
include_search_results=True, ) return WebSearchTool() #  
----- # Sub-agents # -----  
cafe_agent = Agent( name="Cafe Agent", instructions="You help students  
locate and provide information about the Bytes Cafe.",  
tools=[get_bytes_cafe_menu], ) building_agent = Agent( name="Building  
Agent", instructions=( "You help students locate and provide information  
about buildings and rooms on campus. " "Be descriptive when giving  
locations. If you cannot access internal campus maps, say so and "  
"suggest official DigiPen resources." ),  
tools=[make_search_tool(max_num_results=3)], ) course_agent = Agent(  
name="Course Agent", instructions=( "You help students find information  
about DigiPen courses (prerequisites, descriptions, etc.). " "If internal  
course data isn't available, use official DigiPen web pages." ),  
tools=[make_search_tool(max_num_results=5)], ) handbook_agent = Agent(  
name="Handbook Agent", instructions=( "You help students navigate the  
DigiPen student handbook and policies. " "Prefer official DigiPen sources;  
quote short excerpts only when necessary." ),  
tools=[make_search_tool(max_num_results=5)], ) counseling_service_agent =  
Agent( name="Counseling Service Agent", instructions=( "You assist DigiPen  
students by providing clear, supportive information about the DigiPen
```

```
Counseling Center. " "You explain available services, eligibility,  
appointment scheduling, confidentiality, and how to contact the Counseling  
Center. " "You do NOT provide therapy, diagnose conditions, or give  
medical advice. " "If the user expresses stress, overwhelm, anxiety,  
burnout, loneliness, or homesickness, prioritize directing them to " "how  
to book counseling and what options exist." ), tools=[WebSearchTool()], )  
# ----- # Main router agent #  
----- agent = Agent( name="DigiPen Campus Agent",  
instructions=( "You are a helpful campus agent that routes requests to the  
most appropriate specialist agent. " "Be concise and accurate. " "Routing  
rules: If the user mentions stress, anxious/anxiety, overwhelmed, burnout,  
panic, depression, loneliness, homesick, " "can't cope, or asks for  
emotional support → hand off immediately to Counseling Service Agent. "  
"If the user asks about buildings/rooms → Building Agent. "  
"Courses/prerequisites → Course Agent. " "Policies/handbook → Handbook  
Agent. " "Bytes Cafe menu → Cafe Agent." ), handoffs=[building_agent,  
course_agent, handbook_agent, cafe_agent, counseling_service_agent], ) #  
----- # Gradio chat function (streaming) #  
IMPORTANT: Return ONLY plain dict messages, not ChatMessage objects. #  
----- async def chat_with_agent(user_msg: str,  
history: list): # history is list of {"role": "...", "content": "..."}  
when type="messages" messages = [] for msg in history: if isinstance(msg,  
dict) and "role" in msg and "content" in msg: messages.append({"role":  
msg["role"], "content": msg["content"]}) messages.append({"role": "user",  
"content": user_msg}) responses = [] reply_created = False active_agent =  
None result = Runner.run_streamed(agent, messages) async for event in  
result.stream_events(): if event.type == "raw_response_event" and  
event.data.type == "response.output_text.delta": if not reply_created:  
responses.append({"role": "assistant", "content": ""}) reply_created =  
True responses[-1]["content"] += event.data.delta elif event.type ==  
"agent_updated_stream_event": active_agent = event.new_agent.name  
responses.append({ "role": "assistant", "content": f"Running:
```

```

{active_agent}", "metadata": {"title": "Agent Now Running", "id": active_agent}, }) elif event.type == "run_item_stream_event": if event.item.type == "tool_call_item": raw = event.item.raw_item tool_type = getattr(raw, "type", None) if tool_type == "file_search_call": responses.append({ "role": "assistant", "content": f"File search query: {getattr(raw, 'queries', None)}", "metadata": {"title": "File Search Completed", "parent_id": active_agent}, }) else: tool_name = getattr(raw, "name", "unknown_tool") tool_args = getattr(raw, "arguments", {}) responses.append({ "role": "assistant", "content": f"Calling tool {tool_name} with arguments {tool_args}", "metadata": {"title": "Tool Call", "parent_id": active_agent}, }) elif event.item.type == "tool_call_output_item": raw_out = event.item.raw_item out_text = raw_out.get("output") if isinstance(raw_out, dict) else str(raw_out) responses.append({ "role": "assistant", "content": f"Tool output: {out_text}", "metadata": {"title": "Tool Output", "parent_id": active_agent}, }) elif event.item.type == "handoff_call_item": responses.append({ "role": "assistant", "content": f"Handoff to: {event.item.raw_item.name}", "metadata": {"title": "Handing Off Request", "parent_id": active_agent}, }) yield responses #
----- # Gradio UI #
----- demo = gr.ChatInterface( chat_with_agent, title="DigiPen Campus Agent", theme=gr.themes.Soft( primary_hue="red", secondary_hue="slate", font=[gr.themes.GoogleFont("Inter")] ), examples=[ "I'm trying to find the WANIC classrooms. Can you help?", "What's the policy for eating in auditoriums?", "What's today's vegetarian dish at the Bytes Cafe?", "What are the prerequisites for FLM201?", "I'm stressed with GAM541. What should I do? " ], cache_examples=False, # IMPORTANT: avoids startup crash with generator outputs submit_btn=True, flagging_mode="manual", flagging_options=["Like", "Spam", "Inappropriate", "Other"], type="messages", save_history=False, ) if __name__ == "__main__":
demo.launch()

```

Also ensure your requirements.txt (root) contains:

```
openai-agents==0.4.2  
gradio==5.49.1
```

If you still see a crash after this, paste the *new* error — but this version specifically addresses the `FileMessage alt_text` / startup caching issue you hit.