```python
from langchain_core.prompts import ChatPromptTemplate
from langchain_openai import AzureChatOpenAI
import os
from dotenv import load_dotenv
import json
from langchain_core import output_parsers

load_dotenv()
os.environ['AZURE_OPENAI_API_KEY'] = os.getenv("AZURE_OPENAI_API_KEY")
os.environ['AZURE_OPENAI_ENDPOINT'] =
os.getenv("AZURE_OPENAI_ENDPOINT")
api_version = os.getenv("API_VERSION")

model = AzureChatOpenAI (api_version=api_version, model= "finance-
gpt")

system_template = """you are a world class text summarizer.
Given a plain text {meeting_transcript} , you perform the following
tasks very accurately,
1. Create a 2 sentence summary of the meeting's main discussion points
2. list all the action items, importantly each action item should
start/prefaced with a "-"

Return the result strictly as a JSON with keys "summary" and
"action_items. the values of
"action_items" should be a list of strings even if there is only one
action item.
"""

user_template = """Please summarize and list down the key action items
of the
following {meeting_transcript}"""

prompt = ChatPromptTemplate([('system', system_template),
('user',user_template)])


parser = output_parsers.StrOutputParser()
# output = model.invoke(final_prompt)
# Step 1: Convert JSON string to dict
def final_output (output):
    final_output = parser.invoke(output)
    parsed_json = json.loads(final_output)
    return parsed_json

chain = prompt | model | final_output

from pydantic import BaseModel, ValidationError
from typing import List

# Step 1: Define your model
```

```python
class MeetingNotes(BaseModel):
    summary: str
    action_items: List[str]

# Wrap everything in a function
def extract_meeting_notes(prompt: str) -> MeetingNotes:
    max_retries = 2
    for attempt in range(max_retries):
        output = chain.invoke(prompt)

        try:
            _ = MeetingNotes(**output)
            return output

        except (json.JSONDecodeError, ValidationError) as e:
            print(f"□ Attempt {attempt + 1} failed: {e}")
            prompt = "Please output valid JSON only.\n\n" + prompt  #
Retry with system prompt

    raise ValueError("□ Failed to extract valid structured data after
retries.")


#####################  Sample 01
########################################

final_result = extract_meeting_notes(""""

Alice: Welcome everyone. Today we need to finalize the Q3 roadmap.

Bob: I've emailed the updated feature list—please review by Friday.

Carol: I'll set up the user-testing sessions next week.

Dan: Let's push the new UI mockups to staging on Wednesday.

Alice: Great. Also, can someone compile the stakeholder feedback into
a slide deck?

Bob: I can handle the slide deck by Monday.

Alice: Thanks, team. Meeting adjourned.""")
print(json.dumps(final_result, indent=2))
```

```
{
  "summary": "The team discussed finalizing the Q3 roadmap, reviewing
the updated feature list, and preparing user-testing sessions. Action
items include pushing UI mockups to staging and compiling stakeholder
feedback into a slide deck.",
  "action_items": [
    "- Review the updated feature list by Friday.",
    "- Set up user-testing sessions next week.",
    "- Push the new UI mockups to staging on Wednesday.",
    "- Compile the stakeholder feedback into a slide deck by Monday."
  ]
}


final_result = extract_meeting_notes(""""" Host: Let’s kick off our
marketing sync.

Emma: The social campaign draft is 80% done; I’ll share it today.

Frank: I spoke with the design team—they’ll deliver assets by Tuesday.

Emma: Once we have assets, I’ll schedule the ads for next week.

George: Reminder: submit your budget requests before end of day.

Host: Noted. I’ll send out the final budget spreadsheet.
""")
print(json.dumps(final_result, indent=2))

{
  "summary": "The team discussed the progress of the social campaign
draft and the timeline for asset delivery from the design team.
Additionally, there were reminders about budget submissions and the
distribution of the final budget spreadsheet.",
  "action_items": [
    "- Emma will share the social campaign draft today.",
    "- Frank will ensure the design team delivers assets by Tuesday.",
    "- Emma will schedule the ads for next week after receiving the
assets.",
    "- Team members should submit their budget requests before the end
of the day.",
    "- The host will send out the final budget spreadsheet."
  ]
}


final_result = extract_meeting_notes(""""Manager: Let’s start our
weekly check-in.

Dev1: The frontend bug is resolved and deployed.

Dev2: Backend caching is in progress, should be done by Thursday.

Dev3: I’m blocked on API access—waiting on credentials.

Manager: Noted. I’ll follow up on the API issue.

Dev1: Also, starting work on the onboarding flow.
```

```
Manager: Great. Let’s review progress on Friday.""")
print(json.dumps(final_result, indent=2))
```

```
{
  "summary": "During the weekly check-in, the team discussed the
resolution of a frontend bug and ongoing backend caching. Developer 3
is blocked on API access, and the manager will follow up on this issue
while the onboarding flow work begins.",
  "action_items": [
    "- Follow up on the API access issue for Developer 3.",
    "- Review progress on Friday."
  ]
}
```

```
final_result = extract_meeting_notes(""""Lead: We need a quick sync on
the client escalation.

PM: The root cause was a data pipeline delay.

Engineer: Logs show retry failures at 3 AM.

PM: I’ve already updated the RCA document.

Lead: Let’s make sure the fix is included in the next release.

Engineer: Scheduled for tomorrow’s deploy.

Lead: Good. Let’s keep the client updated.""")
print(json.dumps(final_result, indent=2))
```

```
{
  "summary": "The team discussed a client escalation caused by a data
pipeline delay, with the engineer noting retry failures in the logs.
The fix has been scheduled for deployment tomorrow and the lead
emphasized keeping the client updated.",
  "action_items": [
    "- Ensure the fix is included in the next release.",
    "- Keep the client updated."
  ]
}
```