The messages list in your code snippet is used to manage the conversation history with the Azure OpenAI API in the run_session function

This is the initial state of the messages list, created at the start of the run_session function before any API calls are made. Specifically, it occurs in this line:

This happens immediately after the MCP client session is initialized (await session.initialize()) and before the first Azure OpenAI API call (oai_client.chat.completions.create).

```
messages = [
    {
        "role": "system",
        "content": "You have access to two tools via MCP: search_by_image and image_search..."
    },
    {
        "role": "user",
        "content": "Find 5 similar images for this image: https://cloudinary.com/image123.jpg"
    }
]
```

This represents the evolved state of the messages list after one or more iterations of processing user queries, tool calls, and model responses.

```
messages = [
    {"role": "system", "content": "prompt."},
    {"role": "user", "content": "Find images like this: url1"},
    {"role": "assistant", "tool_calls": [...]},
    {"role": "tool", "content": "{results}"},
    {"role": "assistant", "content": "Here are similar images..."},
    {"role": "user", "content": "Now find images like this: url2"},  # New request
    # AI remembers previous context
]
```

the msg object, which is a ChatCompletionMessage extracted from the Azure OpenAI API response in the run_session function. This specific output occurs when the model decides to invoke tools (search_by_image and image_search) in response to the user's query. Let's break down when and under what conditions this exact msg output is generated.

```
print(msg)
ChatCompletionMessage(content=None, role='assistant', tool_calls=[ToolCall(id='call_abc123',
type='function', function=FunctionCall(name='search_by_image',
arguments='{"query_image_path": "https://example.com/image.jpg", "k": 5}')),
ToolCall(id='call_def456', type='function', function=FunctionCall(name='image_search',
arguments='{"image_url": "https://example.com/image.jpg", "num": 5}'))])
```

Why mcp?

1) If one tool fails, we can still run the rest of the tools
2) If we want to migrate from azure open ai to some other service, its easy because server would stay the same
3) Reusable tools
4) Llm smarty decides if or not to call the tools
5) Can easily add new tools because ecosystem is not fragmented
6) The code uses asyncio and MCP's ClientSession to handle tool calls asynchronously (await session.call_tool(name, args)). This is critical for tools like image_search, which may involve slow network requests to external APIs (e.g., SerpAPI).

After tool execution, tool_payloads might look like:

```
tool_payloads = {
  "search_by_image": [
    {"decoded_path": "/local/db/image1.jpg", "description": "Red apple", "score": 0.92},
    {"decoded_path": "/local/db/image2.png", "description": "Fruit basket", "score": 0.85}
  ],
  "image_search": [
    {"title": "Red Apple Stock Photo", "url": "https://example.com/red-apple.jpg"},
    {"title": "Fruit Gallery", "url": "https://stock.example.com/fruit.png"}
  ]
}
```