ESPC
Amsterdam23
52° 20' 29" N
4° 53' 18" E

Unlocking Scenarios with Microsoft Graph Using OpenAI and ChatGPT

Expanding Microsoft Search Extensibility with AI and beyond

Fabian Williams

Sr. Product Manager, Microsoft, USA

Gabrielle Williams

Microsoft Student Ambassador, USA

- ✓ Overview
- ✓ Current State
- ✓ Demos – current state
- ✓ Near Future State
- ✓ Roadmap

TL;DR

- All about Extensibility Options
  - Building In
  - Building With

Overview

What is the Microsoft Search API?

The one unified API search endpoint allowing you to query data from M365

Single search endpoint

Allows **search across all M365 workloads**

Exposes Microsoft Search **intelligence**

graph.microsoft.com/{version}/search/query

Microsoft Search Mission

Microsoft Search connects you to the **knowledge** and **expertise** across your organization and beyond, helping you to be more productive, take action, and save time.

Microsoft Search API Options

What we know and what we use now! Looks familiar?

REST/CSOM

*.sharepoint.com/_api/search/postquery
*.sharepoint.com/_api/search/query

Graph Sites and Files

graph.microsoft.com/{version}/drives/{drive-id}/root/search(q='{search-text}')
graph.microsoft.com/{version}/groups/{group-id}/drive/root/search(q='{search-text}')
graph.microsoft.com/{version}/me/drive/root/search(q='{search-text}')
graph.microsoft.com/{version}/sites/{site-id}/drive/root/search(q='{search-text}')
graph.microsoft.com/{version}/users/{user-id}/drive/root/search(q='{search-text}')
graph.microsoft.com/{version}/sites?search={query}

Transition to Microsoft Graph Search endpoint for OneDrive and SharePoint

Patrick Rodgers

October 30th, 2023 | 0 | 2

As the Microsoft 365 service evolves, we review opportunities to improve the experiences of both users and developers. Aligned to the vision, we are moving to a unified search solution that spans all of Microsoft 365 and announcing that we will no longer be making improvements to the OneDrive and SharePoint search APIs in favor of Microsoft Search APIs in Microsoft Graph. This decision will allow focused development of the best search experience possible in a single coherent API.

This change should encourage you to plan code updates, though we are not announcing an end-of-life date for the existing APIs. However, we will no longer invest in improvements or updates to these existing APIs.

Our guidance is to **transition your applications to call the Microsoft Graph Search API**, and you will find examples of how to find OneDrive and SharePoint content later in this article.

What’s been delivered in 2023 (v1.0)

Shipped (CY 23)

Select Properties
(ListItem, DriveItem)

Multiple Requests
(e.g, Site, ListItem, Bookmarks)

Interleave Specific Connections

(Files, Connectors)

Support via Microsoft Graph Toolkit

Coming Soon

Semantic Search

Current State

What If I want to use Natural Language against my M365 data now?

What task did I do with Gabby this week

Get me all the touchpoints I had around the Roundtree Project

What meetings and files did I work with Adele on this week

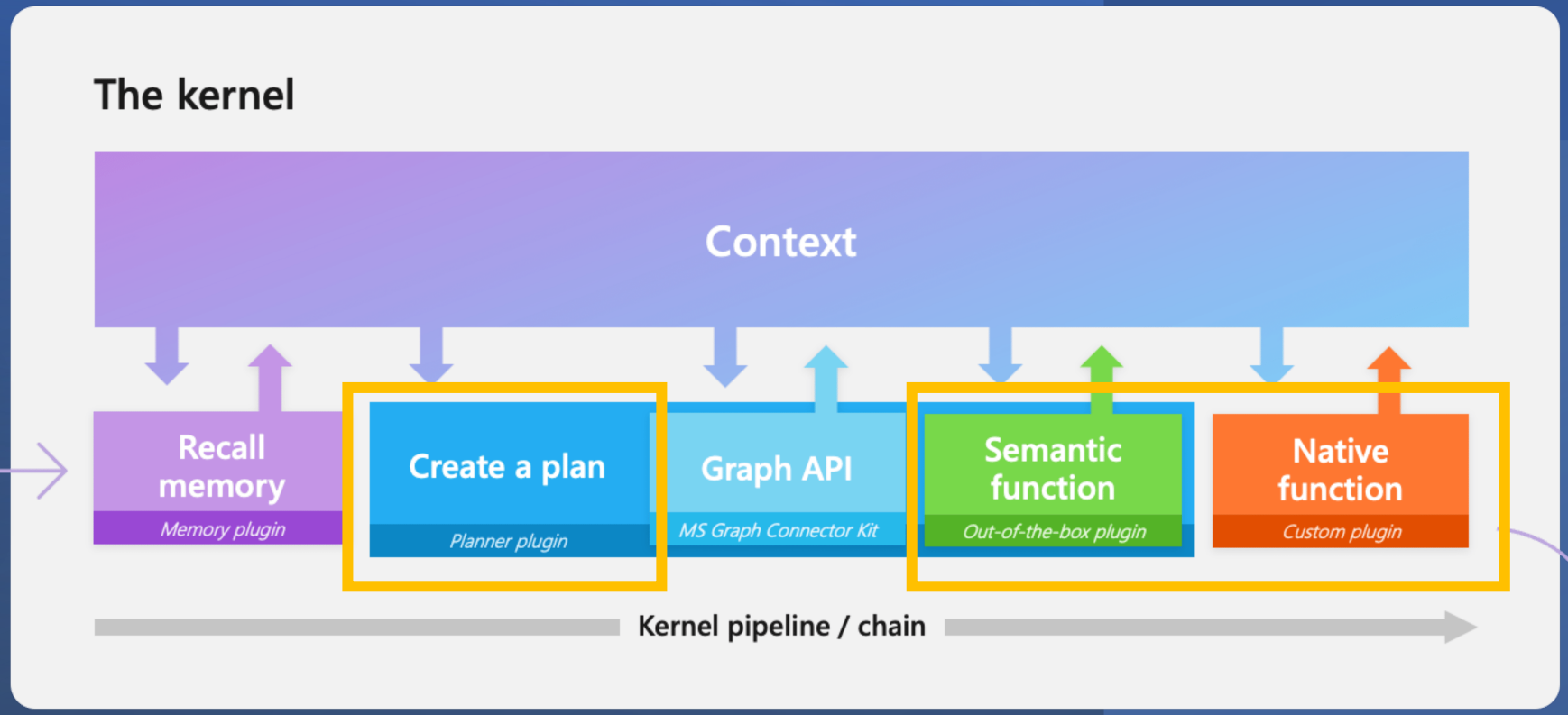
?

?

Cross reference for me feature requests in CRM with emails this week

?

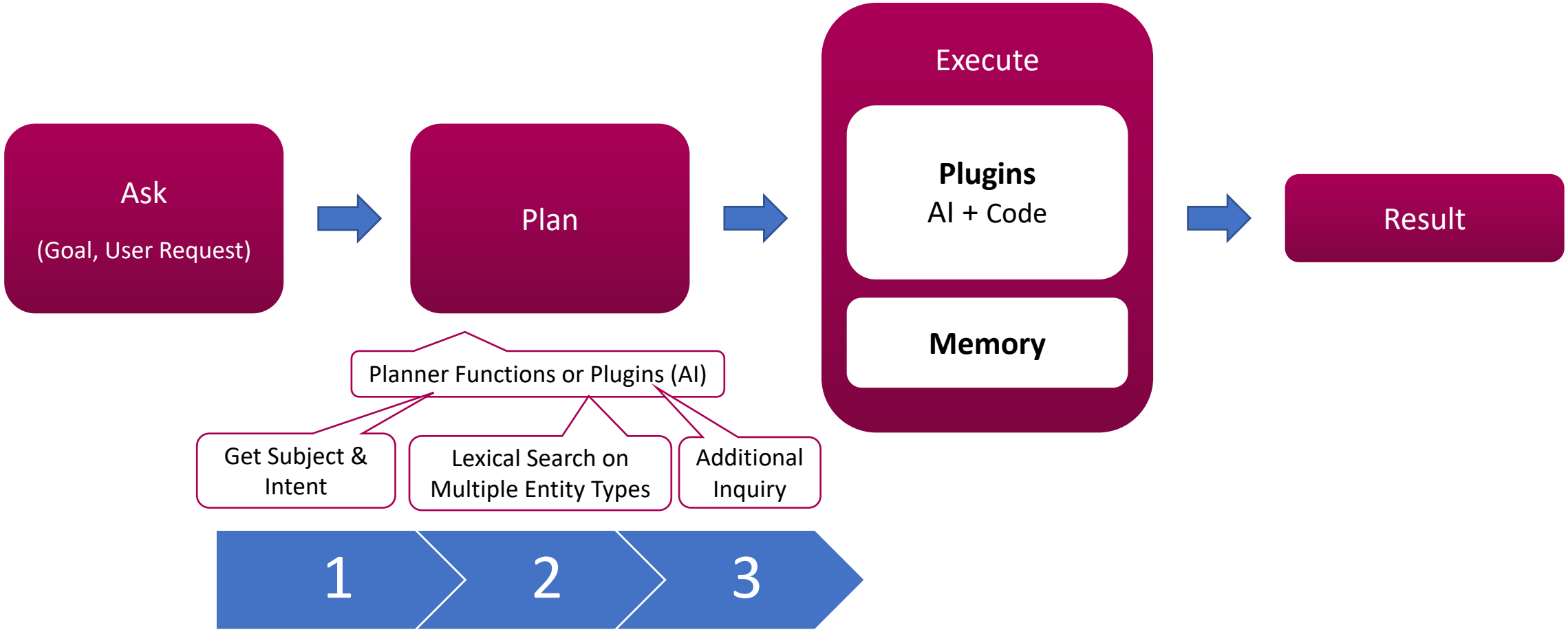
Semantic Kernel with Azure Open AI



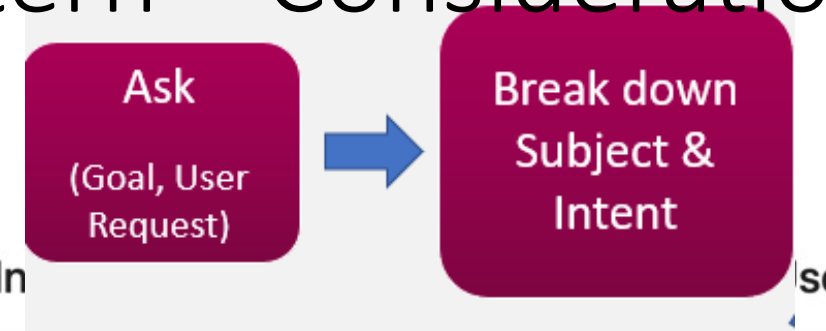
Get me all the touchpoints I had around the Roundtree Project with Adele

Graph call against /Query endpoint and multiple Entity Types sent to SK Memory & vector searched

Basic/Sequential/Stepwise Planner & More



Process Flow/Pattern – Considerations for Tokens



Do the Query and Get the In serPrompt)

```
# note: Using a File to Run these Plugins instead of Inline to
# show a different way
plugins_directory = "Plugins/"
intentFunctions = kernel.import_semantic_skill_from_directory(plugins_directory, "HoldMyHandPlugin")

getIntentFunction = intentFunctions["GetIntent"]
```

```
userIntent = getIntentFunction("What did I accomplish this weekend with Gabby in Antwerp regarding our ESPC demo?")

print(userIntent)
```

```
{ "subject": "Gabby", "action": "know what was accomplished during the ESPC demo in Antwerp"}
```

```
subject_data = json.loads(str(userIntent))
print(subject_data)
```

```
{'subject': 'Gabby', 'action': 'know what was accomplished during the ESPC demo in Antwerp'}
```

```
subject = subject_data['subject']
action = subject_data['action']
```

```
print(subject)
```

```
Gabby
```

```
print(action)
```

```
know what was accomplished during the ESPC demo in Antwerp
```

Semantic Kernel Function for gauging Intent

espc23 / GraphOpenAI / Plugins / HoldMyHandPlugin / GetIntent / **skprompt.txt**

fabianwilliams Everything WORKS! All Test Success for Demo 0472f1b · 14 hour

Code Blame 11 lines (7 loc) · 355 Bytes Raw

```
{{$history}}
User: {{$input}}

----------------------------------------

Provide the intent of the user. The intent should include the subject name and the best one of the following: {{$options}} as
The format should read:

{ "subject": "The name of the person as the subject", "action": "The intent of the user"}

INTENT:
```

Process Flow/Pattern – Load up on Lexical Results

```
request_body = QueryPostRequestBody(
        requests = [
                SearchRequest(
                        entity_types = [
                                EntityType.ChatMessage
                        ],
                        query = SearchQuery(
                                query_string = subject,
                        ),
                ),
        ],
)

chat_result = await client.search.query.post(request_body)
```

Chat

Subject

```
chat_data = chat_result.value
```

```
]

 store summaries, chat IDs, and generated GUIDs
= []

s in the SearchResponse object
response.hits_containers:
n the container
.hits:

        summary = hit.summary

        # Extract the chatId from additional_data
        chat_id = hit.resource.additional_data.get('chatId', 'Unknown')

        # Generate a unique GUID for each hit
        guid = str(uuid.uuid4())

        # Append the chatId, summary, and GUID as a tuple to the list
        summaries_chat_ids_and_guids.append((guid, summary))

# Now 'summaries_chat_ids_and_guids' contains the chatId, summary, and a unique GUID for each search hit
for item in summaries_chat_ids_and_guids:
    await kernel.memory.save_information_async("ESPC23", id=guid, text=summary)
    print(item)
```

Payload from Chat Entity on Query Endpoint

```
('485dbc48-5d9b-4441-9985-4c42d5a186bb', 'I prefer dinner in thanksgiving dinner amsterdam this year over thanksgiving dinner in
year...')
('592f1dfb-f17a-4e09-892f-3a1d7b508634', "I can't get over the thanksgiving dinner we had in amsterdam.")
('0a84505d-bc0c-477e-8efd-9bc2046cf04b', '"id": "9c72876c-b0ec-41b3-8079-8ebe216daa34"')
('105a47a1-da79-421c-9629-4a60d55227ce', 'Fabian Williams  lets practice the demo while we are in Antwerp  sounds like a plan')
('03aef79f-f71b-4ef8-b69a-feeced02443f', "Fabian Williams     hello Gabby, are you prepared for the ChatGPT session with Python?
i'm very excited.")
('8354e0dc-2f5e-4528-beeb-7ef688b61f89', 'lets practice the demo while we are in Antwerp...')
('b0379c7f-6090-489b-9087-bf97e0e8ca42', 'hello Gabby, are you prepared for the ChatGPT session with Python?')
```

Process Flow/Pattern – Load up on Lexical Results

```
request_body = QueryPostRequestBody(
    requests = [
        SearchRequest(
            entity_types = [
                EntityType.DriveItem
            ],
            query = SearchQuery(
                query_string = subject,
            ),
        ),
    ],
)

drive_result = await client.search.query.post(request_body)
```

DriveItem

Subject

```
drive_data = drive_result.value
```

```
se = drive_data[0]                                  nly one SearchResponse object in the list

ty list to store su                                 ion, and generated GUIDs
and_guids = []
]

onse:
hits_containers in the SearchResponse object
container in drive_search_response.hits_containers:
ver each hit in the container
it in drive_hits_container.hits:
            # Extract the summary
            drive_summary = drive_hit.summary

            # Extract the 'from' information from additional_data
            from_data = drive_hit.resource.additional_data.get('from', {}).get('emailAddress', {})
            from_name = from_data.get('name', 'Unknown')
            from_email = from_data.get('address', 'Unknown')
            file_name = getattr(drive_hit.resource, 'name', 'Unknown')
            user_email = from_data.get('email', 'Unknown')

            # Generate a unique GUID for each hit
            drive_guid = str(uuid.uuid4())

            # Append the summary, 'from' information, and GUID as a tuple to the list
            drive_summary_from_and_guids.append((drive_guid, drive_summary, from_name, from_email))
            drive_narrative.append((drive_guid, file_name))
except Exception as e:
    print(f"An error occurred: {e}")

# Now 'drive_summary_from_and_guids' contains the summary, from information, and a unique GUID for each search hit
for item in drive_narrative:
    await kernel.memory.save_information_async("ESPC23", id=drive_guid, text=file_name)
    print(item)
```

Payload from Drive Item Entity on Query Endpoint

```
('b5e3ee37-4cfe-43eb-9f83-788127f7fba2', 'CollabSummit2024_GabbySpeaking.docx')
('bbd25f11-4f13-4b44-98f9-175590bf6c31', 'Apps')
('fec858b9-abdf-479a-8e6f-2af806d4903c', 'ProjectRoundtreeStatusReport.docx')
('6f484c77-000a-45d5-b01c-f5c53c5d8ab9', 'HowToGainAccessToStudyGroup.docx')
```

Process Flow/Pattern – Load up on Lexical Results

```
request_body = QueryPostRequestBody(
    requests = [
        SearchRequest(
            entity_types = [
                EntityType.Message
            ],
            query = SearchQuery(
                query_string = subject,
            ),
        ),
    ],
)

mail_result = await client.search.query.post(request_body)
```

Messages

Subject

```
mail_data = mail_result.value
```

```
mail_data[0]  # Ass                    earchResponse object in the list

                    and generated GUIDs
uids = []

:
_containers in the SearchResponse object
iner in mail_search_response.hits_containers:
each hit in the container
 mail_hits_container.hits:
he summary
y = mail_hit.summary

            # Extract the 'from' information from additional_data
            from_data = mail_hit.resource.additional_data.get('from', {}).get('emailAddress', {})
            from_name = from_data.get('name', 'Unknown')
            from_email = from_data.get('address', 'Unknown')

            # Generate a unique GUID for each hit
            guid = str(uuid.uuid4())

            # Append the summary, 'from' information, and GUID as a tuple to the list
            mail_summary_from_and_guids.append((guid, mail_summary, from_name, from_email))
            mailconstruct = mail_summary + " from " + from_name
except Exception as e:
    print(f"An error occurred: {e}")

# Now 'mail_summary_from_and_guids' contains the summary, from information, and a unique GUID for each search hit
for item in mail_summary_from_and_guids:
    await kernel.memory.save_information_async("ESPC23", id=guid, text=mailconstruct)
    print(item)
```

Payload from
Messages Entity
on Query Endpoint

```
('ddcb7e8d-7051-4cb0-ae77-ac0e7712e1d9', "Fabian Williams shared a file with you  Here's the document that Fabian Wi
ProjectRoundtreeStatusReport   This link will work for anyone.  Open   Privacy...", 'Unknown', 'Unknown')
('4829ff66-4231-4646-a19e-623e8444b585', "Fabian Williams shared a file with you   Here's the document that Fabian W
HowToGainAccessToStudyGroup  1 min to read this.   This link will work for...", 'Unknown', 'Unknown')
('97f7e954-dc85-488e-b76e-05b20ae97967', 'From: Microsoft <microsoft-noreply@microsoft.com> Sent: Thursday, November
an Williams <fabian@fabster.onmicrosoft.com> Subject: Your Microsoft invoice...', 'Unknown', 'Unknown')
('d37abc85-d6a6-4a6d-aba9-8d526c1fe699', 'Private to you Hi, <c0>Gabby</c0> Gabgab, Welcome to your new digest from
```

Test Semantic Memory Recall/Search

```python
async def search_memory_examples(kernel: sk.Kernel) -> None:
    questions = [
        "What did I accomplish this weekend with Gabby in Antwerp regarding our ESPC demo?",
        "What ESPC23 artifacts do I have?",
    ]

    for question in questions:
        results = await kernel.memory.search_async("ESPC23", question, limit=3)
        for result in results:
            print(f"Question: {question}")
            print(f"Answer: {result.text}\nRelevance: {result.relevance}\n")
```

```python
await search_memory_examples(kernel)
```

```
Question: What did I accomplish this weekend with Gabby in Antwerp regarding our ESPC demo?
Answer: hello Gabby, are you prepared for the ChatGPT session with Python?
Relevance: 0.7862505624190529

Question: What did I accomplish this weekend with Gabby in Antwerp regarding our ESPC demo?
Answer: hello Gabby, are you prepared for the ChatGPT session with Python?
Relevance: 0.7862505624190529

Question: What did I accomplish this weekend with Gabby in Antwerp regarding our ESPC demo?
Answer: Ok, So make sure you know your lines, study your answers to the questions. Get your demo ready to go. MOST IMPORTANT:
ifices to the demo gods!!! Dad. from Unknown
Relevance: 0.7582602447035468

Question: What ESPC23 artifacts do I have?
Answer: HowToGainAccessToStudyGroup.docx
Relevance: 0.7296164736930801

Question: What ESPC23 artifacts do I have?
Answer: HowToGainAccessToStudyGroup.docx
Relevance: 0.7296164736930801
```

RAG and Run

```
[56]: print("Simulate Chat experience with SK and RAG")
      rag_func, context = await setup_RAG(kernel)
```

```
Simulate Chat experience with SK and RAG
```

```
[60]: print("Ask a question (type 'exit' to exit):\n")
      answer = await RAG(kernel, rag_func, context)
```

```
Ask a question (type 'exit' to exit):

User:>  What did fabian and gabby work on

ChatBot:> Based on the information provided, Fabian and Gabby worked on a ChatGPT session with Python.

 Source: hello Gabby, are you prepared for the ChatGPT session with Python?
 Relevance: 0.7917456309860029
```

Demo Semantic Kernel against Search Endpoint

Behinds the Scenes & associated Costs

Azure Open AI | **Microsoft Graph** | **Azure Open AI** | **Azure Open AI**

Ask (Goal, User Request) → Break down Subject & Intent → Call Into Search and Execute Lexical Query → Add to SK Memories → Vectorized Results and Text Completed Response → Result to User

Break down Subject & Intent → /DriveItem /Events /Messages /Chat → Add to SK Memories

Resource	Resource type	Location	Resource group ...
fabsopenaialpha	Azure AI service	US East	openaialpha

Service name	Service tier	Meter
Cognitive Services	Azure OpenAI	Text-Davinci Tokens
Cognitive Services	Azure OpenAI	GPT-3.5-turbo Tokens
Cognitive Services	Azure OpenAI	Embeddings-Ada Tokens

Resource	Resource type	Location	Resource group ...
fabsopenaialpha	Azure AI service	US East	openaialpha

Service name	Service tier	Meter
Cognitive Services	Azure OpenAI	Text-Davinci Tokens
Cognitive Services	Azure OpenAI	GPT-3.5-turbo Tokens
Cognitive Services	Azure OpenAI	Embeddings-Ada Tokens

Ask (Goal, User Request)

Multiple Items are returned here and Per Entity → For every item returned it is sent to Semantic Kernel Memory as an Embedding → In the Recall Function we send the Ask as an embedding to the Results In Memory for nearest neighbor search

Potential Pitfalls if you Roll on your own

Potential for Throttling

Additional Token Costs

No Secret Sauce [Insights & Signals]

Manage your own Security

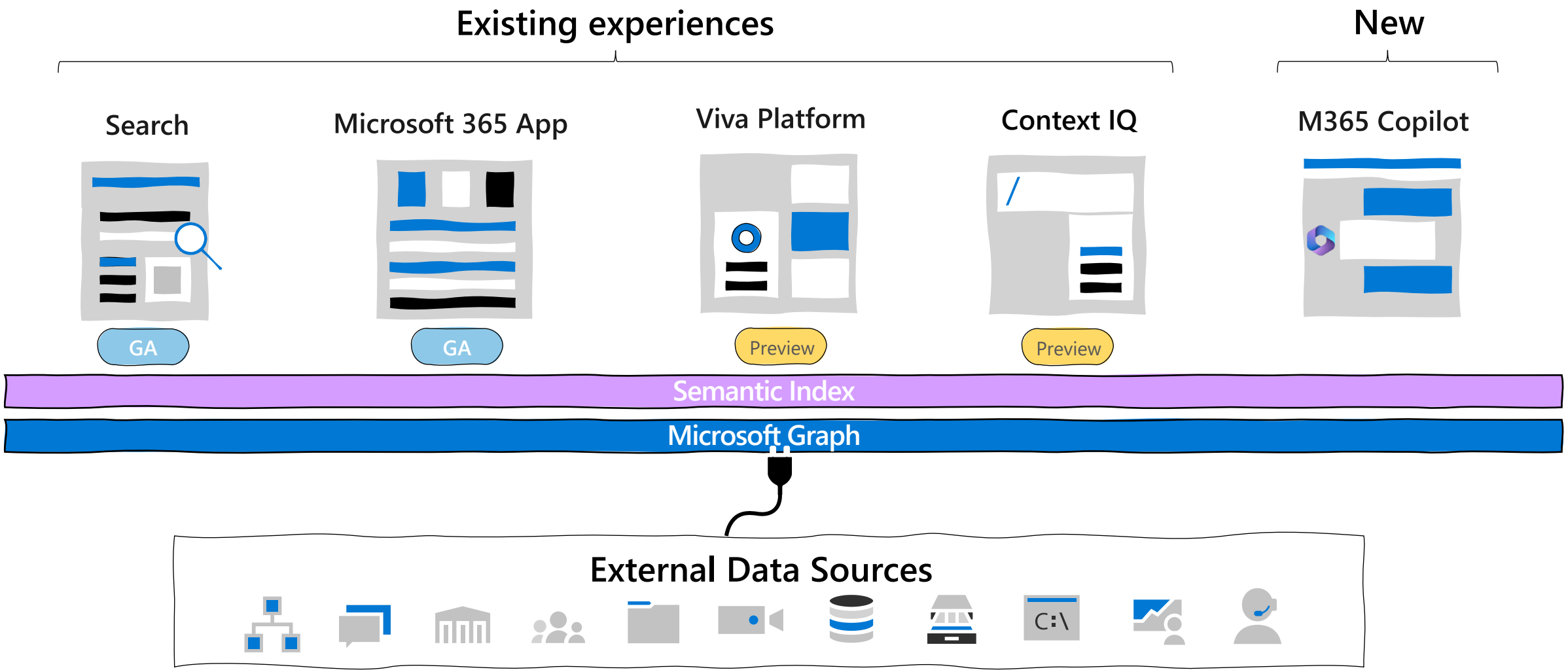
Constantly keeping Data in Sync

Benefits if you Roll with Microsoft 365

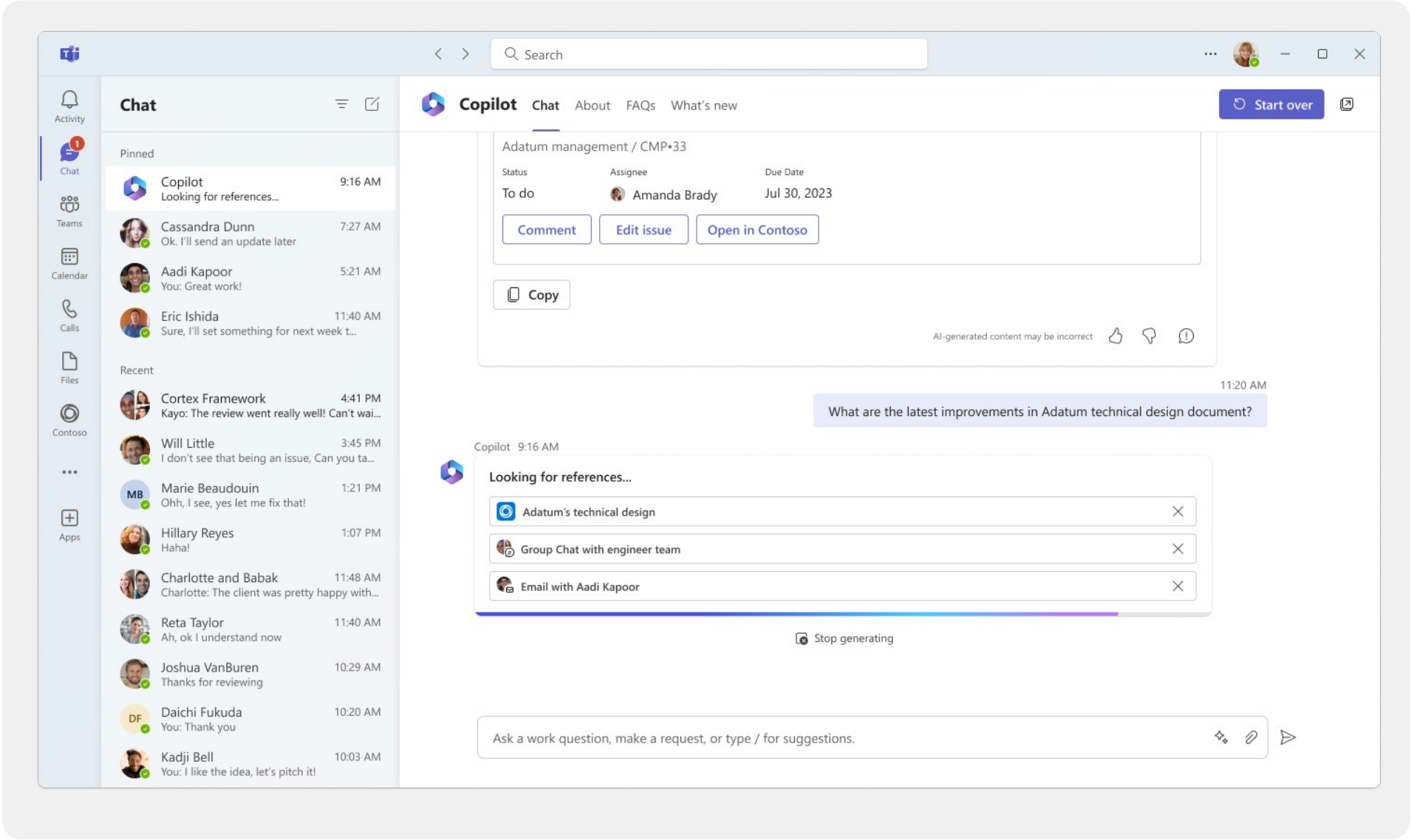
- Reduce Time to Market
- Reduce/Eliminate Token Costs
- Reduce Complexity
- Value from MSFT Insights & Signals
- No need to Sync Data /Vectors
- Inherent Trust & Security

Near Future State!

You can bring any data source in to M365 and Copilot



Copilot uses semantic indexing



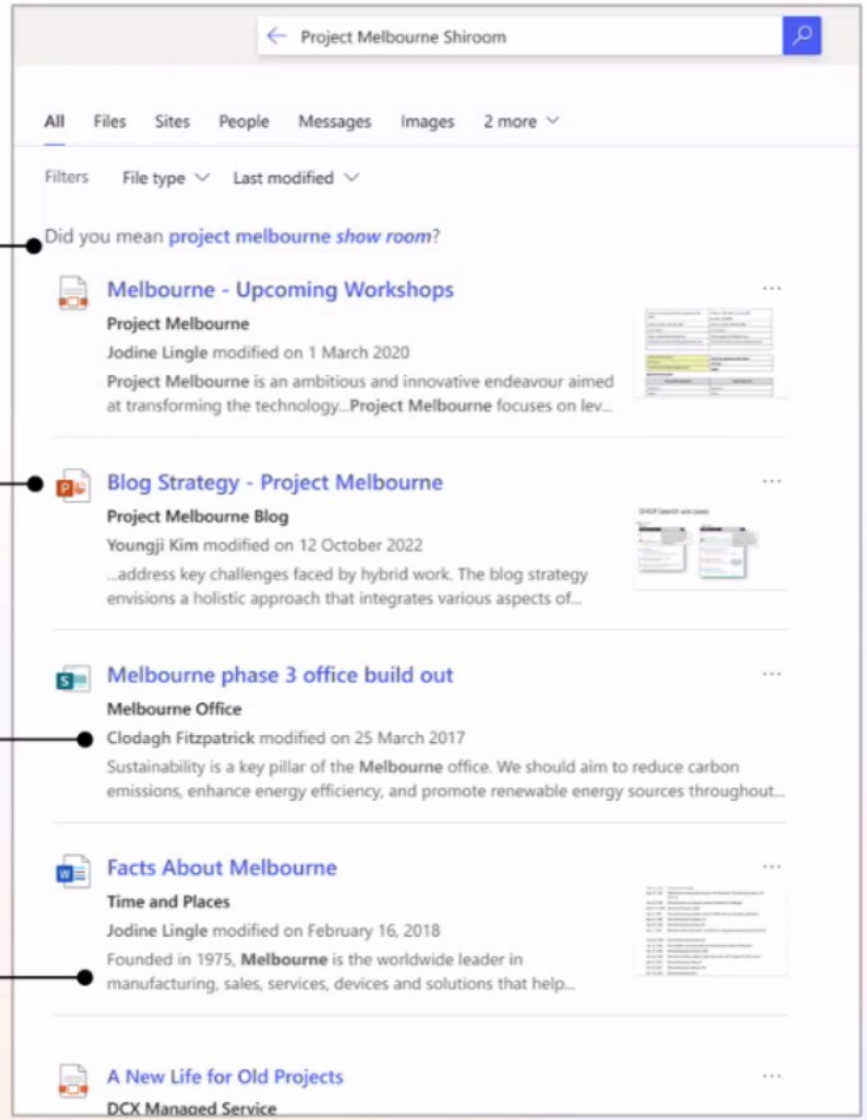
Semantic Index for Copilot Search Improvements

keyword matched results & suggestions

closest matched content ranked in order

last modified date and person adds relevance

document summary provides context



Without Semantic Index for Copilot

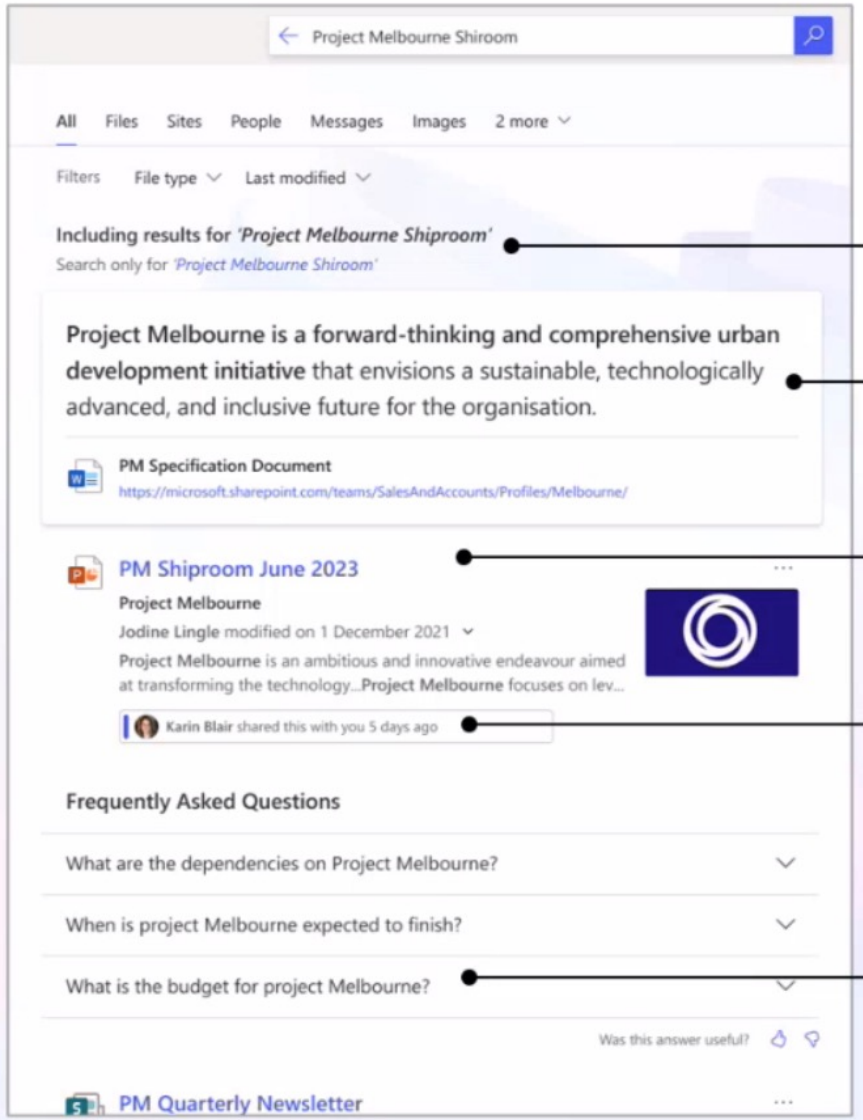
autocorrected spelling of organizational jargon

informative answers directly from content

more relevant results, based on search intent

enhanced content sharing insights

AI generated questions and answers on topic



With Semantic Index for Copilot

What’s Next for exposing semantic in Graph Search API

- **Key Benefits**:
- **Delivers more personalized results** in semantic search scenarios compared to keyword-based search scenarios.
- Supports leveraging **Graph connectors to enrich the Copilot experience with M365 and external data**
- **Enables** Retrieval Augmented Generation (**RAG**) **to get extract accurate,** precise answers from the Index that can enrich and personalize Generative AI use cases.

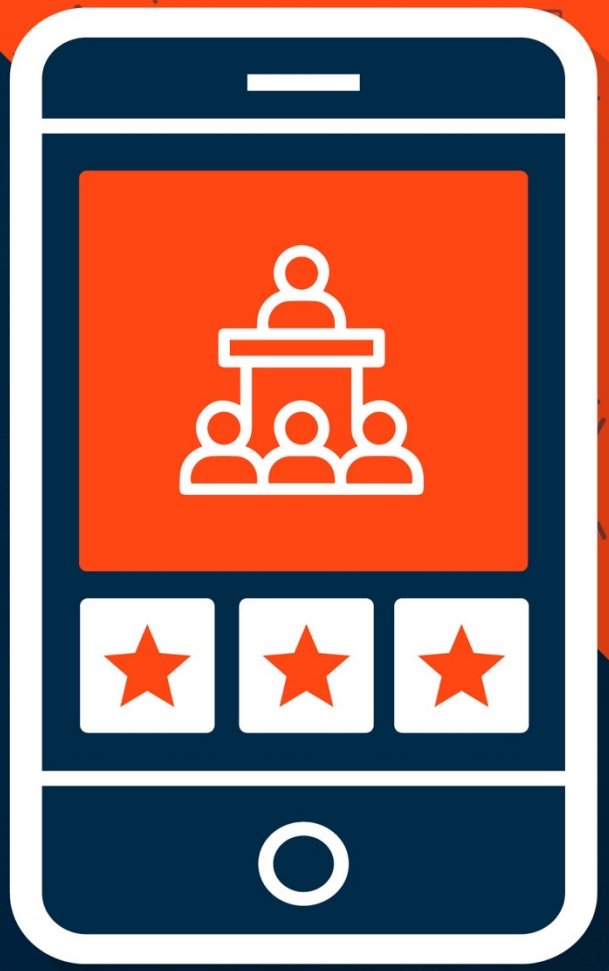
- **Specifications**
- Will support M365 and Graph Connector File types initially (other file types TBD)
- Optional parameter in API request body to enable/disable
- Programmatic access via API requires Copilot License

Roadmap

High Level Roadmap Timelines

CY23H2	EOY 2023	CY2024
Finalize Development **Gather Use cases**	**Release Dogfood version Internally**	**GA *** By Build 2024

Please rate this session on the app



GET IT ON Google Play

Download on the App Store