



Part of the AgentCon World Tour by the Global AI Community

AI Agents Developer Conference

agentcon.city/oslo



TypeAgent: From RAG to Structured RAG



Bernhard Merkle

Made Possible by



Microsoft

audiocodes

**global AI
community**



vivicta*



ProcessPilotene

bouvet

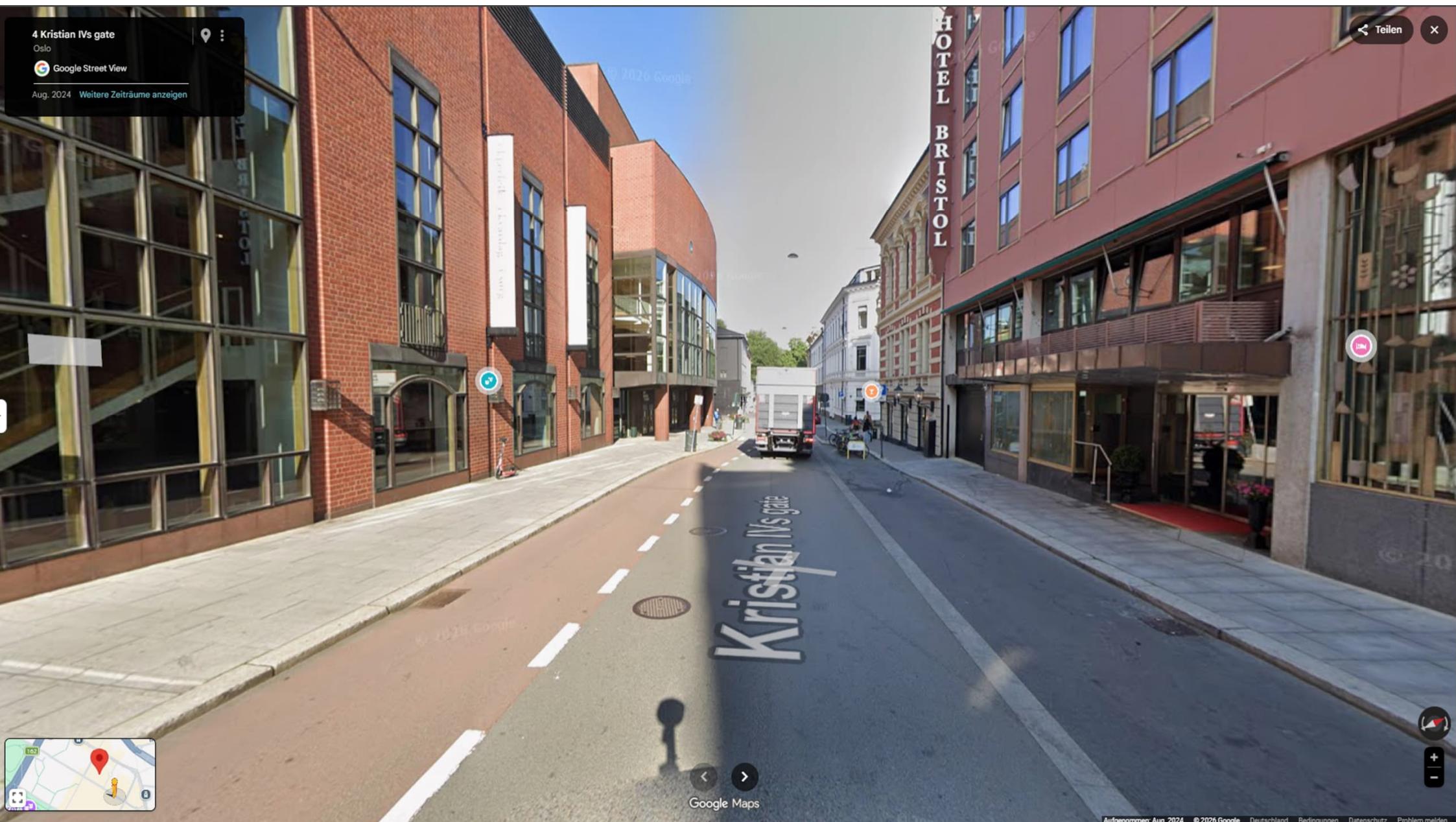
q.beyond

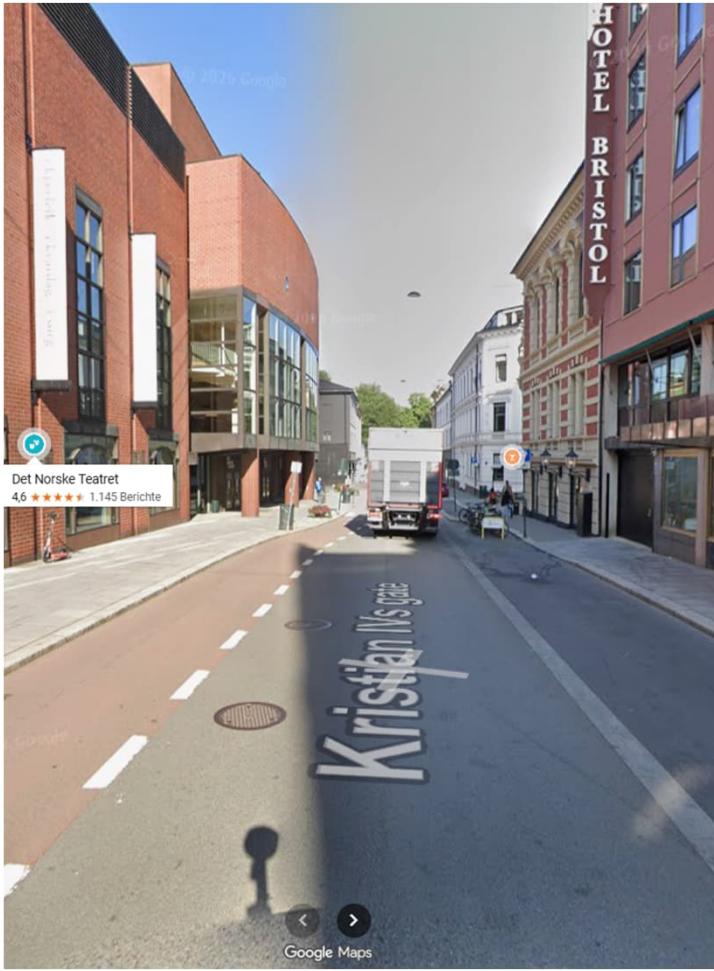
**THRIVE
CONFERENCE**

About me...



SICK
Sensor Intelligence.

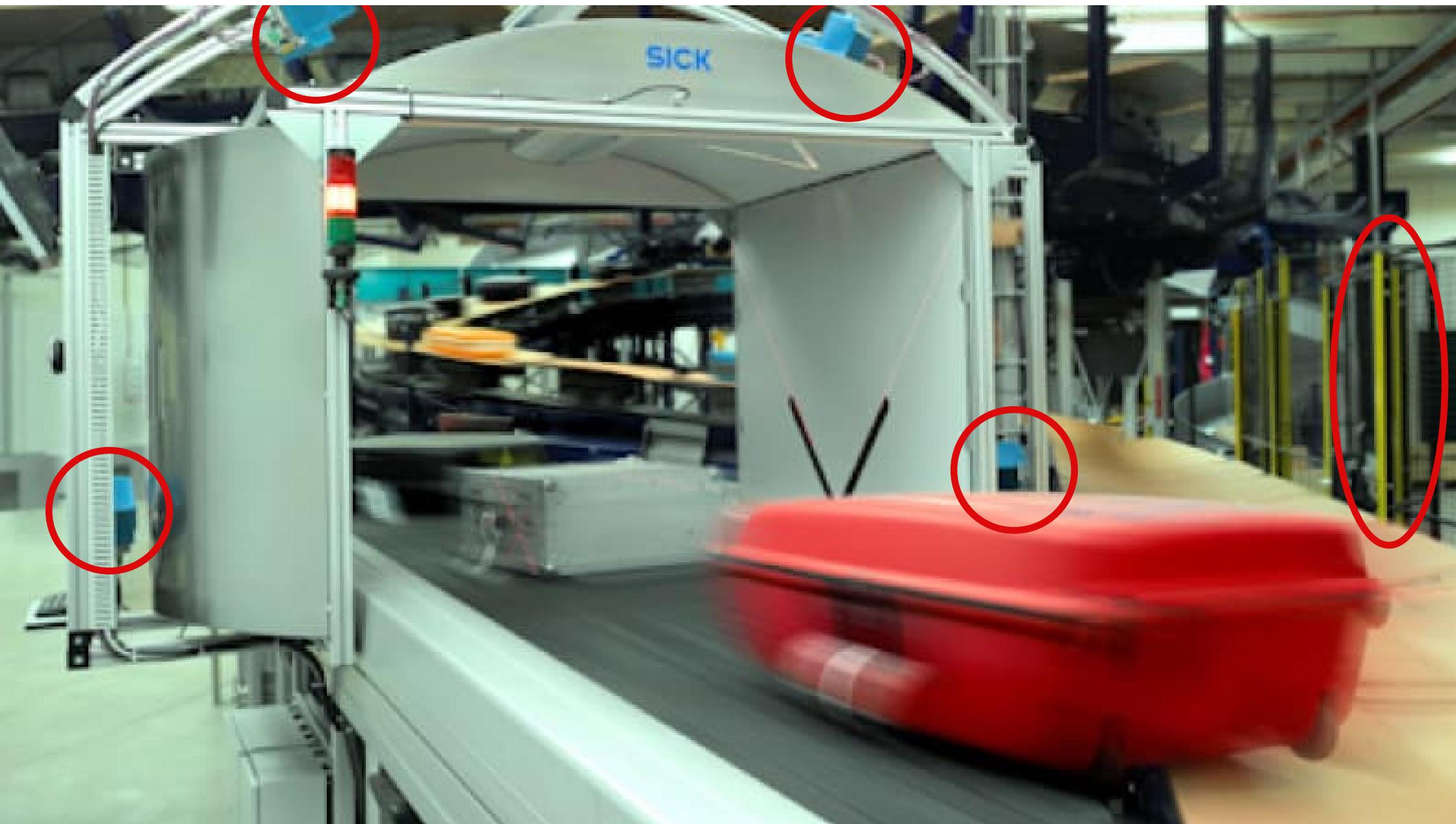


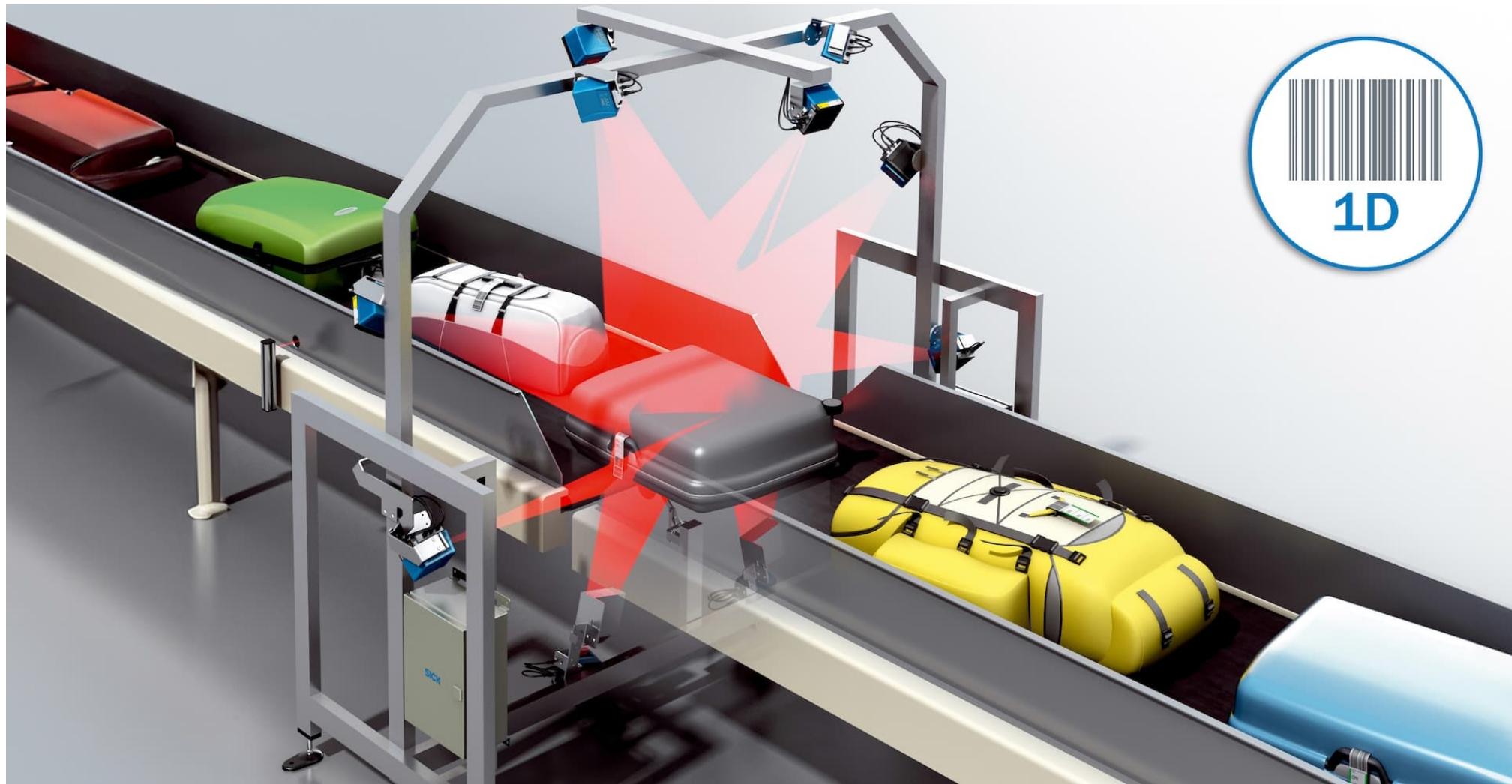














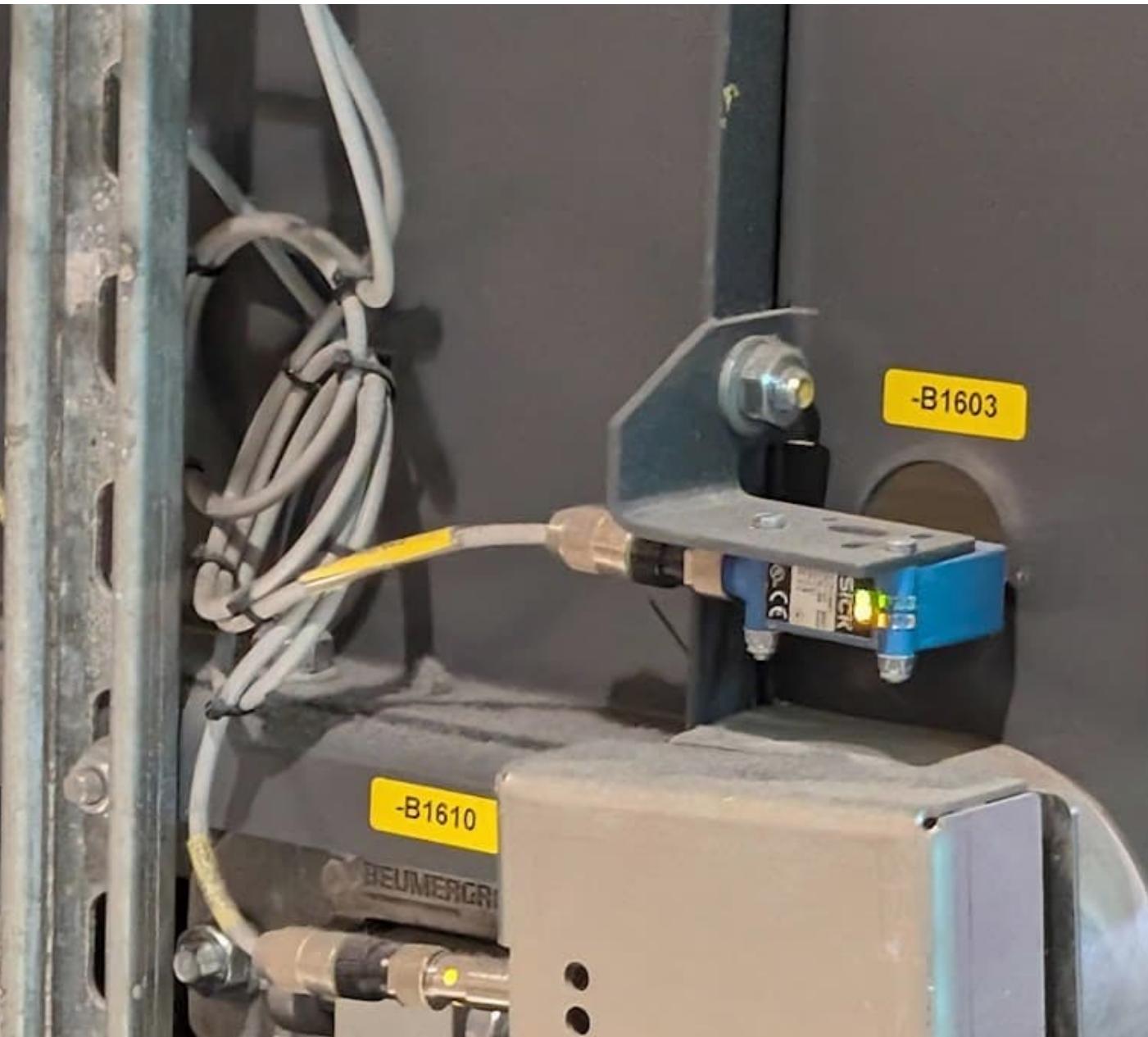


=TRC06.017



Part No. TP13170

-B1601



THE PARROT
IS DEAD

TS





It's bleedin' demised!

No, no! It's resting!

NORWEGIAN
BLUE - PINEIS
FURJORDE



**AGENTIC
WORKFLOWS**



GITHUB & AI AGENTIC
WORKFLOW



LLM vs RAG vs AI Agent vs Agentic AI

Aspect	LLM	RAG	AI AGENT	Agentic Ai
Definition	A smart text generator. You ask it something, it writes back an answer.	An AI that searches through your documents before answering your question.	An AI helper that can plan steps, use different tools, and complete whole tasks for you.	A team of AI workers where each one has a different job, and they work together to finish big projects.
Architecture	Your question → AI brain → Answer	Your question → Searches your files → Reads relevant parts → Writes answer using what it found	Your goal → Makes a plan → Uses tools it needs → Checks results → Adjusts and continues → Finishes task	Your goal → Manager assigns tasks → Researcher AI does its part → Writer AI does its part → Reviewer AI checks → Team delivers final result
Core Components	One AI model that learned from tons of text.	AI model plus a searchable library of your documents.	AI brain plus access to tools like search, calculator, file creators.	Multiple AI workers, shared memory they all access, manager that coordinates them.
Primary Functionality	Creates text by guessing what word should come next, like autocomplete on your phone.	Answers questions using information from your actual files and documents.	Breaks down your goal into steps, picks the right tools, and works until it's done.	Different AI specialists work together, pass information between them, and coordinate like a real team.
Typical Use Case	Writing emails, posts, stories, or getting quick explanations.	Answering questions about company policies, product guides, or research papers.	Research projects, collecting and organizing data, creating content packages.	Marketing campaigns, big research projects, running ongoing business processes.
Tool Integration	Works alone. Can't connect to	Can read your documents but	Can use web search, spreadsheets, calculators	Each team member uses whatever tools it needs for its



Problems with Classic-RAG

- ⚠ The "Lost in the Middle" Problem
- ⚠ Lack of Global Context (The "Needle in a Haystack")
- ⚠ Semantic Disconnect (The "Keyword Trap")
- ⚠ Retrieval Noise and "Garbage In, Garbage Out"

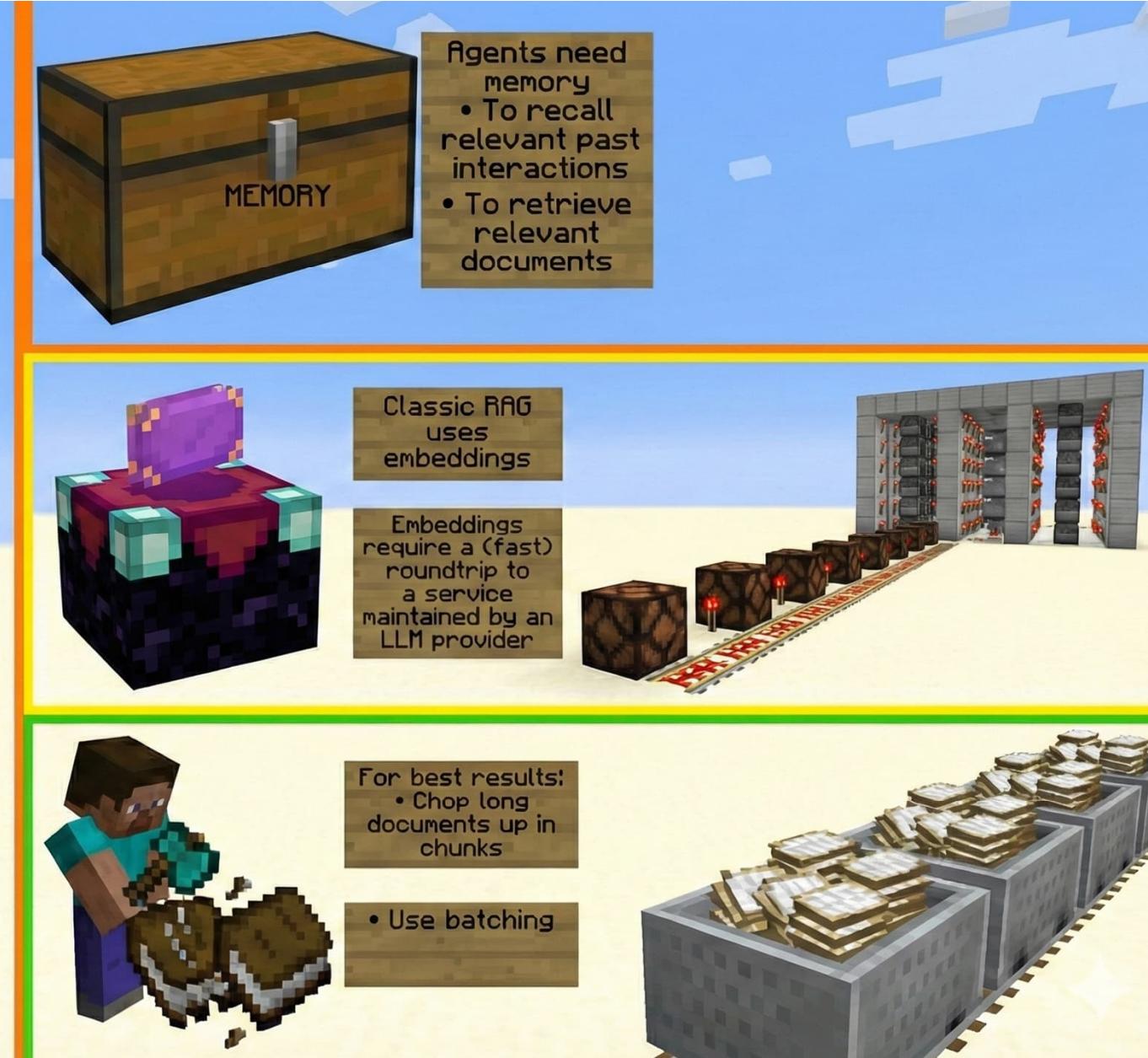
- ⚠ Retrieval recall and precision are not really good
- ⚠ Useage of much compute and data for....
- ⚠ tokens, embeddings with no real semantic...
- ⚠ context windows is limited, workaround, not efficient...

RAG

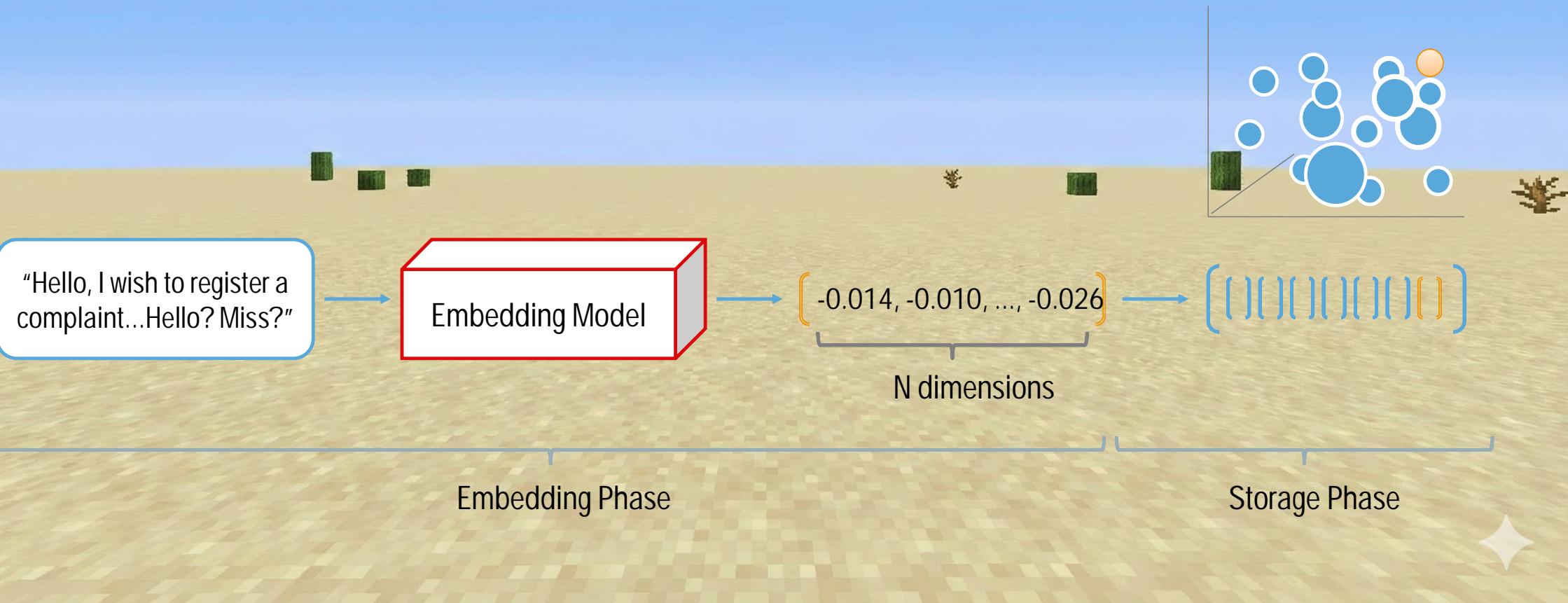
Structured-RAG



Review: Classic RAG (Retrieval- Augmented Generation)

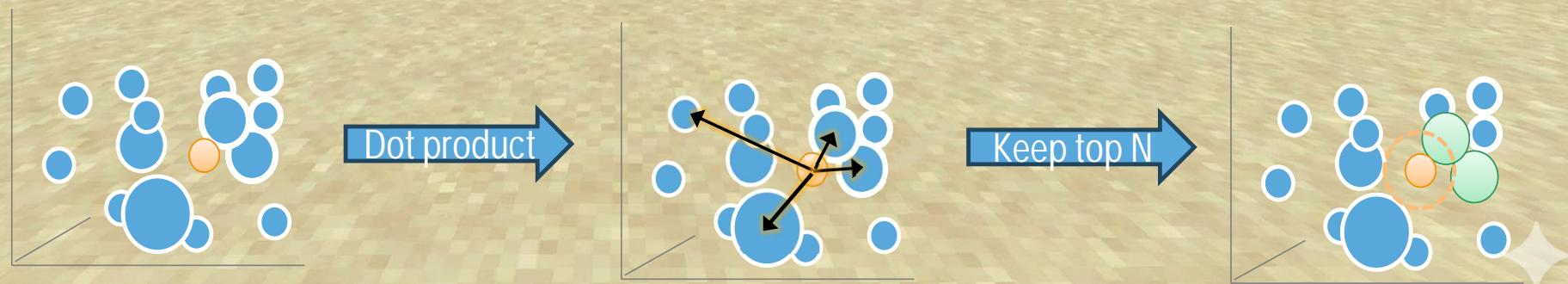
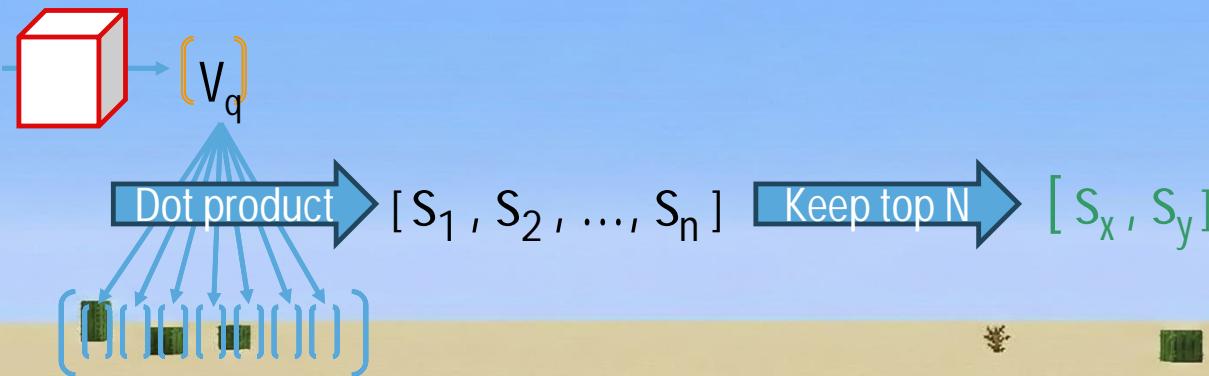


Classic RAG – Embedding Storage



Classic RAG – Embedding Retrieval

"What's the matter
with the parrot?"



Classic RAG:

Pros	Well understood (mostly).
	Easy to deploy.
Cons	Expensive and complicated.
	Take up lots of space.
	Large input strings give “mushy” results, near many things, not all that near anything in your query.



AND NOW FOR
SOMETHING
COMPLETELY
DIFFERENT ...

WANDA

DEAD PARROT



WANDA

DEAD PARROT



WANDA

DEAD PARROT

OMG META OBJECT FACILITY STACK

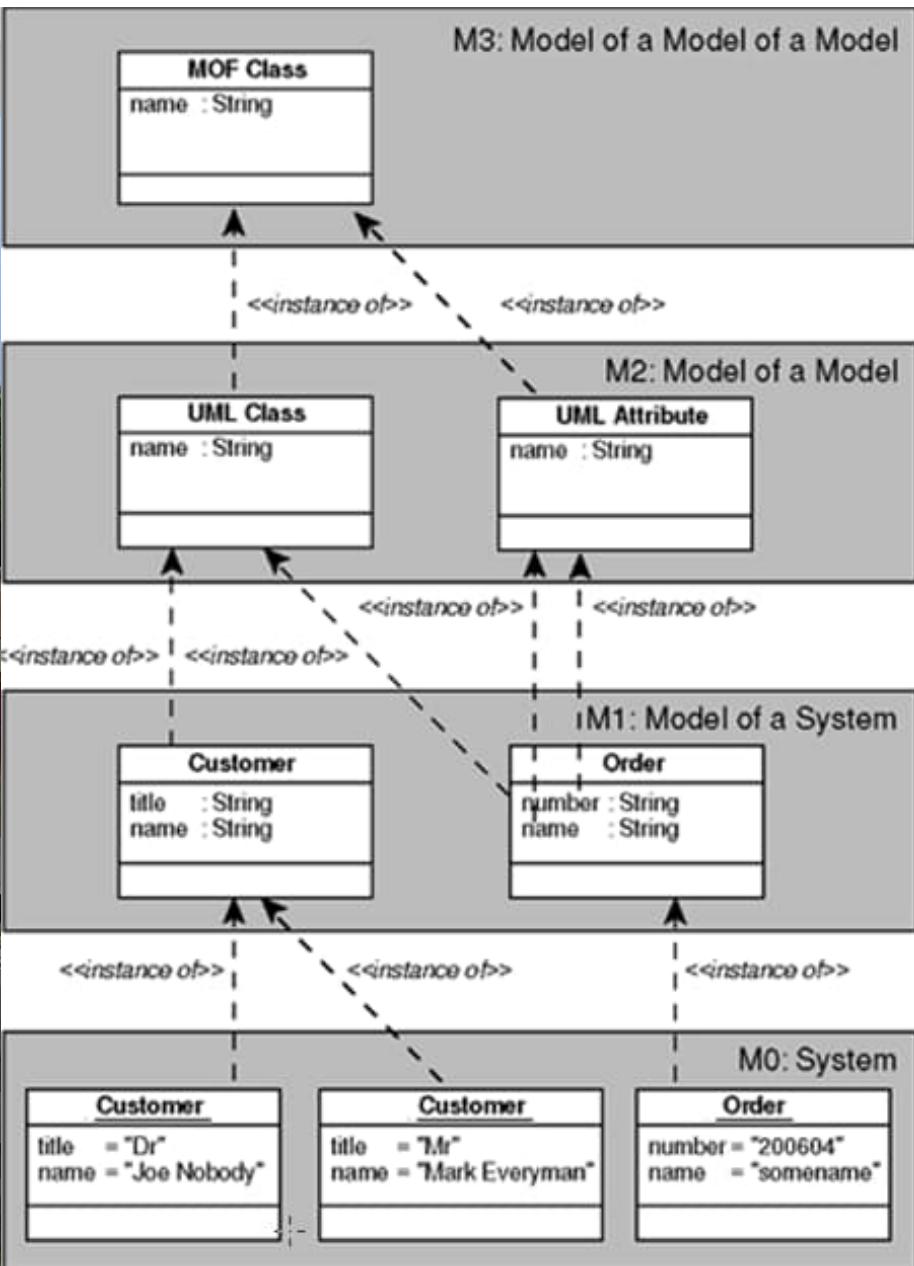
M3: MOF
(META-METAMODEL)

M2: METAMODEL (UML, CWM)

M1: MODEL (USER MODEL)

M0: DATA (INSTANCE)





Structured RAG:

1. Ingestion

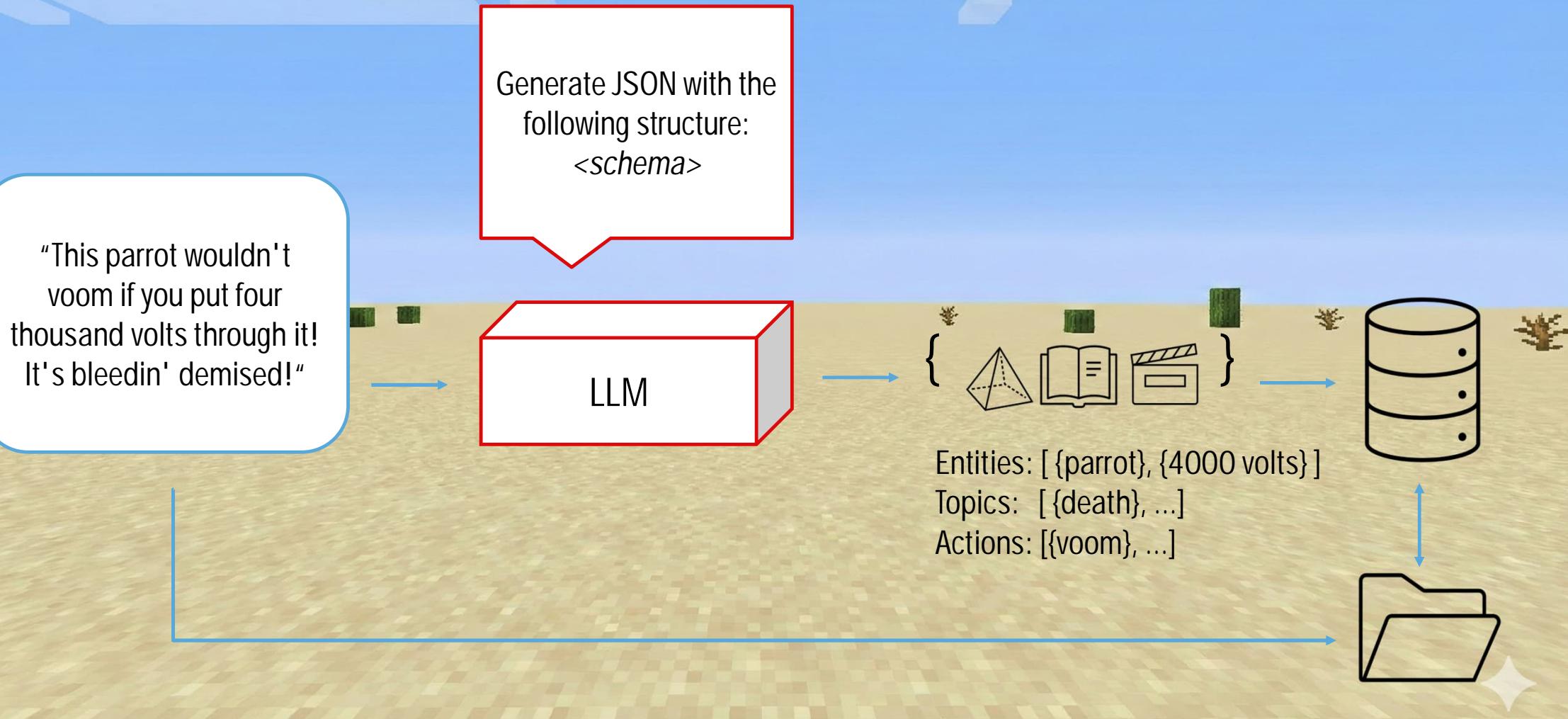
Run each input string through an LLM that extracts “knowledge” from it.

Knowledge: entities (people, places, etc.), actions, topics, relationships, etc.

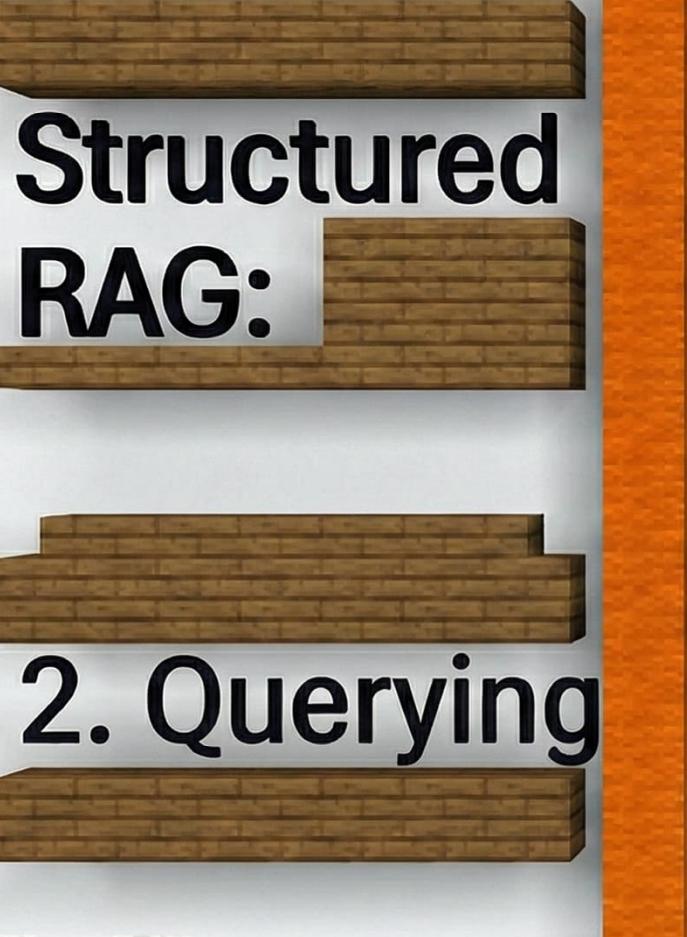
Store knowledge nuggets in a classic database and create indexes over them.

The database is easily queried – it’s classic computer science.

Structured RAG – Ingestion Pipeline



Structured RAG:



2. Querying

Turn user question into an (abstracted) database query.

Run the query, producing {Entities, Topics, Actions}.

Produce answer from best query results.

Structured RAG – Query Pipeline



"What's wrong
with the
parrot?"

Given the user
question and the
knowledge nugget,
write an answer.

Turn this into a query
for our <schema>

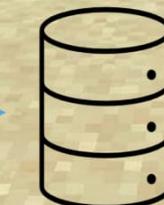
LLM

"The parrot is
deceased."



LLM

<?>



Structured RAG evaluated

Retrieval recall and precision are better due to extraction and storage of knowledge nuggets, which can be queried more precisely, generating higher information density, leading to better answers.

Scalable to much larger conversation histories: indexing approach needs less compute and data.

Can do inference over knowledge nuggets. E.g. $\text{artist}(\text{Palin}) \rightarrow \text{person}(\text{Palin})$, which helps with e.g. “what people did we talk about yesterday?”

Some knowledge can be extracted without consulting an LLM (e.g. email headers).

Bulk ingestion is slower, due to the use of an LLM instead of just embeddings.

pwsh in demo

merklbel ➜ ♦ demo ➜ ♦ load-dotenv-lib ➜ ♦ ➜ in pwsh at 01:23:35

Ingest Messages 1/2

examples > demo >  ingest.py >  main

```
1  from dotenv import load_dotenv
2
3  from typeagent import create_conversation
4  from typeagent.transcripts.transcript import TranscriptMessage, TranscriptMessageMeta
5
6  load_dotenv()
7
8
9  def read_messages(filename) -> list[TranscriptMessage]:
10     messages: list[TranscriptMessage] = []
11     with open(file=filename, mode="r") as f:
12         for line in f:
13             # Parse each line into a TranscriptMessage
14             speaker: str, text_chunk: str = line.split(sep=None, maxsplit=1)
15             message: ConversationMessage = TranscriptMessage(
16                 text_chunks=[text_chunk],
17                 metadata=TranscriptMessageMeta(speaker=speaker),
18             )
19             messages.append(message)
20
21     return messages
```

Ingest Messages 2/2

```
examples > demo > 🗂 ingest.py > 📄 main
22
23  async def main():
24      conversation: ConversationBase[ConversationMessage] = await create_conversation(dbname="demo.db", TranscriptMessage)
25      messages: list[ConversationMessage] = read_messages(filename="testdata.txt")
26      print(f"Indexing {len(messages)} messages...")
27      results: AddMessagesResult = await conversation.add_messages_with_indexing(messages)
28      print(f"Indexed {results.messages_added} messages.")
29      print(f"Got {results.semrefs_added} semantic refs.")
30
31
32  if __name__ == "__main__":
33      import asyncio
34
35      asyncio.run(main())
36
```

Query database

```
examples > demo > 🗂 query.py > ...
1  from dotenv import load_dotenv
2
3  from typeagent import create_conversation
4  from typeagent.transcripts.transcript import TranscriptMessage
5
6  load_dotenv()
7
8
9  async def main():
10     conversation: ConversationBase[ConversationMessage] = await create_conversation(dbna..."demo.db", TranscriptMessage)
11     question = "Who volunteered to do the python library?"
12     print("Q:", question)
13     answer: str = await conversation.query(question)
14     print("A:", answer)
15
16
17 if __name__ == "__main__":
18     import asyncio
19
20     asyncio.run(main())
21
```

```
pwsh in typeagent-py X + v
typeagent-py python .\tools\ingest_vtt.py .\tests\testdata\Parrot_Sketch.vtt .\tests\testdata\Confuse-A-Cat.vtt .\tests\testdata\Short.vtt -d mp.db
Using Azure OpenAI... 1.227s
Processing 152 messages in batches of 4...
4/152 messages | 29 refs | 10.4s/batch | 10.4s elapsed
8/152 messages | 59 refs | 7.8s/batch | 18.2s elapsed
12/152 messages | 83 refs | 6.2s/batch | 24.4s elapsed
16/152 messages | 112 refs | 6.6s/batch | 31.0s elapsed
20/152 messages | 142 refs | 7.9s/batch | 39.0s elapsed
24/152 messages | 172 refs | 7.1s/batch | 46.0s elapsed
28/152 messages | 207 refs | 8.0s/batch | 54.1s elapsed
32/152 messages | 233 refs | 5.4s/batch | 59.5s elapsed
36/152 messages | 252 refs | 7.8s/batch | 67.4s elapsed
40/152 messages | 272 refs | 7.5s/batch | 74.9s elapsed
44/152 messages | 283 refs | 6.4s/batch | 81.3s elapsed
48/152 messages | 308 refs | 10.4s/batch | 91.7s elapsed
52/152 messages | 327 refs | 6.7s/batch | 98.5s elapsed
56/152 messages | 351 refs | 6.3s/batch | 104.8s elapsed
60/152 messages | 375 refs | 7.1s/batch | 111.9s elapsed
64/152 messages | 384 refs | 3.5s/batch | 115.5s elapsed
68/152 messages | 402 refs | 3.7s/batch | 119.2s elapsed
72/152 messages | 418 refs | 3.8s/batch | 122.9s elapsed
76/152 messages | 435 refs | 5.2s/batch | 128.2s elapsed
80/152 messages | 453 refs | 4.7s/batch | 132.8s elapsed
84/152 messages | 468 refs | 5.4s/batch | 138.3s elapsed
88/152 messages | 488 refs | 4.3s/batch | 142.6s elapsed
92/152 messages | 500 refs | 4.0s/batch | 146.6s elapsed
96/152 messages | 519 refs | 7.4s/batch | 154.0s elapsed
100/152 messages | 532 refs | 4.0s/batch | 158.0s elapsed
104/152 messages | 551 refs | 5.3s/batch | 163.3s elapsed
108/152 messages | 573 refs | 5.7s/batch | 169.0s elapsed
112/152 messages | 593 refs | 4.6s/batch | 173.6s elapsed
116/152 messages | 607 refs | 4.0s/batch | 177.6s elapsed
120/152 messages | 620 refs | 3.3s/batch | 180.9s elapsed
124/152 messages | 644 refs | 8.9s/batch | 189.8s elapsed
128/152 messages | 670 refs | 10.0s/batch | 199.8s elapsed
132/152 messages | 688 refs | 7.5s/batch | 207.3s elapsed
136/152 messages | 713 refs | 7.9s/batch | 215.2s elapsed
140/152 messages | 731 refs | 6.1s/batch | 221.3s elapsed
144/152 messages | 759 refs | 8.4s/batch | 229.7s elapsed
148/152 messages | 778 refs | 5.6s/batch | 235.3s elapsed
152/152 messages | 805 refs | 6.8s/batch | 242.2s elapsed
Imported 152 messages from 3 file(s) to mp.db
All indexes built successfully

To query the transcript, use:
python tools/query.py --database 'mp.db' --query 'Your question here'
```

pwsh in typeagent-py X + v

merkbel ➤ ♦ typeagent-py ➤ ♦ ♦fixDemosForConfTalk ↑1 ➤ ♦

♦ in pwsh at 23:02:18

pwsh in typeagent-py X + v

merklbel ➤ ♦ typeagent-py ♦ ♦fixDemosForConfTalk ↑1 ➤ ♦

♦ in pwsh at 22:51:23

Classic RAG vs Structured RAG

Feature	Classic RAG	Structured RAG
Logic	Semantic similarity (Vectors)	Typed Conversations & Schema-based indexing
Memory	Flat document chunks	Hierarchical, human-like memory
Precision	Lower (suffers from "context drift")	Higher (resolves pronouns + state)
Efficiency	Often requires large contexts	Distills logic into faster, smaller models

Structured RAG vs (Knowledge) Graph RAG

Feature	Structured RAG (TypeAgent)	Knowledge Graph RAG (GraphRAG)
Core Structure	Typed Schemas (TypeScript/JSON)	Nodes & Edges (Triplets)
Philosophy	"Logical distillation" of conversation.	"Relational mapping" of a corpus.
Primary Goal	Precise agent memory & task state.	Multi-hop reasoning & global summaries.
Indexing Unit	Conversations, turns, and entities.	Entities, relationships, and communities.
Search Method	Schema-driven lookup & pronoun resolution.	Graph traversal & community summarization.
Best For...	"What was the third book we discussed?"	"What are the common themes in these 1,000 files?"

Structured RAG vs (Knowledge) Graph RAG

- 1. The Core Mechanism
 - ▶ Structured RAG: uses KnowPro to extract a "logical structure" from conversation
 - ▶ Graph RAG: It builds a network of entities (nodes) and their interactions (edges)

- 2. Information Density vs. Connectivity
 - ▶ Structured RAG prioritizes density control. It distills a long conversation into a compact, structured representation that fits within the "attention budget" of smaller, faster models. It is about making the model's memory more efficient.
 - ▶ GraphRAG prioritizes connectivity. It is less about saving tokens and more about ensuring that the model can find "hidden" relationships. For example, if Document A mentions a person and Document B mentions a company they own, GraphRAG links them so the LLM can answer questions about the person's business interests.

Structured RAG vs (Knowledge) Graph RAG

■ 3. Use Case Focus

- ▶ Structured RAG is built for agents that do work.
superior because it maintains the "logical thread."
- ▶ GraphRAG is built for analysis.
standard for "Tell me the overall sentiment of these research papers"

■ 4. Developer Experience

- ▶ Structured RAG relies on Schema-as-Code.
You define what your memory looks like using TypeScript interfaces.
The system ensures the LLM populates these structures strictly.
 - ▶ GraphRAG relies on Ontology Extraction.
The system often discovers the structure itself (unsupervised) or
uses a pre-defined graph schema to map out the world.
-

A screenshot of a Windows desktop environment. At the top, a browser window titled "New Tab" is open, displaying a search bar with "https://example.com" and a blue "Go" button. The browser has standard window controls (minimize, maximize, close) and a "+" button for new tabs. Below the browser is the taskbar, which includes the "TypeAgent Shell v0.0.1 - dispatcher" icon, a file explorer icon, and other system icons. The main workspace shows a messaging application. A message from "greeting" at 1:28:56 AM reads: "Bonsoir, Bernhard! Hope you're having a productive night working on your presentation. Need a break or maybe some inspiration?". A hand icon is next to the message. A horizontal line with the text "yesterday" spans the width of the messages. Another message from "greeting" at 1:30:14 AM reads: "Ahoy, Bernhard! Working late on this chilly Monday night? Let's make sure those conference slides are as smooth as Bach's finest composition!". A yellow hand icon is next to this message. A horizontal line with the text "a few minutes ago" spans the width of the messages. At the bottom right of the screen, there is a toolbar with several icons: a magnifying glass, a camera, a microphone, a red circle with a slash, and a right-pointing arrow.

New Tab

https://example.com Go

TypeAgent Shell v0.0.1 - dispatcher - [] Zoom: 140%

yesterday

greeting - 1:28:56 AM

Bonsoir, Bernhard! Hope you're having a productive night working on your presentation.
Need a break or maybe some inspiration?

a few minutes ago

greeting - 1:30:14 AM

Ahoy, Bernhard! Working late on this chilly Monday night? Let's make sure those
conference slides are as smooth as Bach's finest composition!

TypeAgent Shell v0.0.1 - dispatcher - [] Zoom: 140%

calendar.scheduleEvent - 1:35:34 AM

✓ Event created: **meeting** on 02/02/2026 at 09:00

- 1:37:31 AM

clear

system history clear - 1:37:31 AM

Chat history cleared.

system.history.clearHistory - 1:37:35 AM

Action system.history.clearHistory completed.

New Tab

https://example.com

Go

Kalender – Bernhard Merkle

https://outlook.office.com/calendar/0/view/week?deepLink...

Outlook

Startseite Anzeigen Hilfe

Neues Ereignis Tag Arbeitswoche Woche Monat Geteilte Ansicht ...

Heute < > 02–08 Februar, 2026 Im Büro

	02 Mo	03 Di	04 Mi	05 Do	06 Fr	07 Sa	08 So
6							
7							
8							
9							
10							
11							
12							
13							
14							
15							

KnowPro" system for extracting logical structure from text

```
// Define the logical structure of a memory "Entity"
export interface Entity {
  name: string;
  type: "person" | "place" | "organization" | "concept" | "task";
  attributes: string[]; // Key-value or descriptive traits
}

// Define the conversation "Index"
export interface SemanticMemory {
  entities: Entity[];
  facts: string[];
  // Links current conversation turns to specific logical objects
  relationships: Array<{
    subject: string;
    predicate: string;
    object: string;
  }>;
}

// The "KnowPro" logical structure for a past turn
export interface ConversationMemory {
  timestamp: string;
  summary: string;
  entitiesExtracted: string[]; // Links to global Entity store
  resolvedPronouns: Map<string, string>; // e.g., "he" -> "John"
}
```

```
{
  "entities": [
    {"name": "John", "type": "person"},
    {"name": "lake trip", "type": "event"}
  ],
  "task": {
    "action": "finish photo montage",
    "deadline": "Friday",
    "assignedBy": "John"
  },
  "disambiguation": {
    "He": "John",
    "the pictures": "pictures from lake trip"
  }
}
```

KnowPro Folder (TypeScript): Look for index.ts and ontology.ts. These files define how the system builds "Indices" of conversations that are logically linked rather than just vector-indexed.

typeagent-py (Python): Look at the typeagent/know_pro directory. It translates these TypeScript interfaces into Pydantic models, which act as the structured schema for Python-based agents.

Knowpro: Datastructures

```
src > typeagent > transcripts > transcript.py > ...
11  from ..knowpro import secindex, serialization
12  from ..knowpro.conversation_base import ConversationBase
13  from ..knowpro.convsettings import ConversationSettings
14  from ..knowpro.interfaces import ConversationDataWithIndexes, SemanticRef, Term
15  from ..knowpro.universal_message import ConversationMessage, ConversationMessageMeta
16  from ..storage.memory.convthreads import ConversationThreads
17  from ..storage.memory.messageindex import MessageTextIndex
18
19 # Type aliases for backward compatibility
20 TranscriptMessage: <class 'ConversationMessage'> = ConversationMessage
21 TranscriptMessageMeta: <class 'ConversationMessageMeta'> = ConversationMessageMeta
22
23
24 # TypedDict for serialization (kept for backward compatibility with saved files)
25 class TranscriptMessageMetaData(TypedDict):
26     speaker: str | None
27     listeners: list[str] # Must match serialized ConversationMessageMeta
28
29
30 class TranscriptMessageData(TypedDict):
31     metadata: TranscriptMessageMetaData
32     textChunks: list[str]
33     tags: list[str]
34     timestamp: str | None
35
```

Summary: Structured RAG

- ✓ Retrieval recall and precision are better (knowledge nuggets)
- ✓ can be queried more precisely, generating higher information density
- ✓ → Better Answers
- ✓ Scalable to much larger conversation histories
- ✓ indexing approach needs less compute and data
- ✓ Some knowledge can be extracted without consulting an LLM
- ✓ More lightweight than (Knowledge) GraphRAG
- ⚠ research project, early stage...
- ⚠ Bulk ingestion is slower (LLM usage)...

Sources:

- Code
 - Structured-RAG
 - ▶ <https://github.com/microsoft>TypeAgent> (Initial TS Implementation Umesh Madan and Steven Lucco)
 - ▶ <https://github.com/microsoft/typeagent-py> (Python Port of TypeAgent Guido van Rossum)
 - Graph-RAG
 - ▶ <https://github.com/microsoft/graphrag>
 - ▶ <https://github.com/neo4j/neo4j-graphrag-python>
 - Collections
 - ▶ <https://github.com/graphrag>
 - ▶ <https://github.com/Andrew-Jang/RAGHub>
 - ▶ <https://github.com/DEEP-PolyU/Awesome-GraphRAG>

Sources:

- Presentations:
 - Microsoft Build 2025: "*Making Agent Memory Better and Agent Actions Faster with TypeAgent*"
<https://www.youtube.com/watch?v=acbUy2oZqmA>
 - PyBay 2025: "*Structured RAG is better than RAG*"
<https://www.youtube.com/watch?v=-kIESD7iB-s>
- Guidos Talk: <https://github.com/microsoft/typeagent-py/blob/main/docs/StructuredRagPyBay25.pdf>
- Images generated via <https://gemini.google.com/>

