#### The Alan Turing Institute

Data-Centric Engineering at Scale: Developing an Al Data Engineer

**Adam Sobey** 

**Programme Director of Data-Centric Engineering, The Alan Turing** Institute

Professor of Data-Centric Engineering, Maritime Engineering, University of Southampton

27<sup>th</sup> March 2025

#### **Digitalisation for Decarbonisation**







#### JAWS

- Power prediction software
- Commercialized by Shell Shipping and Maritime, licenced to Kongsberg
- Trim and draft optimisation
- 5% Fuel Saving (when combined with T-VOS potential for 18%)
- 62 vessels

#### T-VOS

- Voyage Optimisation Software
- Commercialised through Theyr
- Clients include Euronav, Spire, OpenMI and Lloyd's Register
- 5% fuel saving, 7% earlier arrival, 8% TCE – Compared to other software
- 2,151 vessels, potential for 30,000

#### Data Pipeline

- Developed data pipeline
- Bluebox: active data collection
- Developing reinforcement learning controller
- Increased data engineers from 1 to 16 (6 PhDs)

#### **Digital Backbone**

 Development of data models is becoming increasingly simple

However,

- Companies don't have the digital backbone to take advantage of these methods
- Scale-up is a challenge, how do we provide a bespoke data model to 10,000 ships?
- Need to automate the training, quality assurance and communication
- Can we develop an AI Data Engineer to create a bespoke data pipeline?



AI Data Engineer having a lunch break

# The AI Data Pipeline







# Automated data pipeline



A simple solution?



User prompt



Integrated development environment



**Cloud service** 

#### What if we don't have an expert?



#### **Requested Output**

- Produce a concise summary of the agreed-upon pipeline architecture, highlighting its key components and connections.

- Provide a high-level plan and rationale for the design, explaining why it is wellsuited for the given data and use case.

- Estimate the cloud resources, implementation efforts, and associated costs, providing a rough breakdown and complexity rating.

- Generate a `PIPELINE\_OVERVIEW.json` file, detailing the proposed complete architecture in JSON format with the following fields:

- "Platform": A cloud service provider's name if the cloud solution is the best, or "local server" if locally hosted servers are preferred.

- "Component 1": The first component in the pipeline framework, with AWS official name.

- "Component 2": The second component in the pipeline framework, with AWS official name. Continue until all required components are listed.

- "Implementation difficulties": A rating from 1 to 10 (lowest to highest).

- "Maintenance difficulties": A rating from 1 to 10 (lowest to highest).





Al Data Engineer



JSON with pipeline specification

User prompt

#### Single agent simulation

data-engineer ( to primary\_assistant )

exitcode: 1 (execution failed) Code output: ERROR: Could not find a version that satisfies the requirement pcl-python (from versions: none) ERROR: No matching distribution found for pcl-python

primary\_assistant ( to data-engineer )

The error message indicates that there is no package named `pcl-python` available on PyPI.

After further research, I found that the Point Cloud Library (PCL) does not have a Python wrapper. However, you can use the Open3D library in Python to read and process point cloud data.

Here's an updated version of the script:



#### Single agent simulation

data-engineer ( to primary assistant )
auta-engineer (to primary_assistant)
exitcode: 0 (execution succeeded)
Code output:
nrimary assistant (to data-opginger)
primary_assistant ( to data-engineer )
It seems that the execution was successful, but there is no code output provided.
However, based on the updated code I provided earlier, here's what we can expect as output:
1. A "cameral image ing" file will be created in the current working directory with a random image
T. A camera_image.jpg me will be created in the current working directory with a random image.
2. A "lidar_data.pcd.bin" file will be created in the current working directory with some dummy lidar data.
3. The `process_data()` function will generate some dummy lidar data and return it along with the camera image.
4. The `store_data()` function will create a CSV file named "processed_data.csv" in the current working directory, containing the
processed data.
If you'd like to see more specific output, please let me know what you're looking for (e.g., the contents of the "camera_image.jpg" file,
the dummy lidar data, etc.).
data-engineer ( to primary_assistant )

TERMINATE



Data science engineer



User prompt

Infrastructure engineer

Business objective engineer

Machine learning engineer

Conversation delegation engineer



JSON with pipeline specification

DataEngineerAgent (to chat\_manager):

Final output doesn't include most components.

Can not remember the required JSON format

Longer conversation chain: 4 proposals + 12 discussions (maximum allowed by the prompt)

Using claude-3.5-haiku, it costs 0.27\$

#### I'll complete the PIPELINE\_OVERVIEW.json with the full details based on our previous consensus discussions: ```json "project": "Self-Driving Car Data Pipeline", "version": "1.0", "infrastructure": { "platform": "AWS Cloud-Native", "compute": { "type": "Kubernetes-Distributed", "primary\_hardware": "NVIDIA DGX Systems", "gpu\_specification": "A100 GPUs with Tensor Core Technology" }, "anomaly\_detection": { "methodology": "Multi-layered Unsupervised Approach", "techniques": [ "Autoencoder-based detection", "Isolation Forest algorithm" "anomalv indicators": | "Reconstruction error", "Statistical deviation" "data processing": { "sensor modalities": [ "LiDAR", "Camera", "Radar". "GPS" "processing\_framework": "Distributed Machine Learning" This JSON captures the key architectural decisions, compute infrastructure, anomaly detection strategy, and data peline. as discussed and agreed upon in our previous consensus steps.

# Intrinsic Memory Agents: Efficient and Cost Saving Agents



### **Memory Limitations**

- Most agents have limited context windows, meaning: they can only "remember" a certain amount of conversation history.
- As conversations grow longer, earlier exchanges get pushed out of their accessible memory.
- This creates a situation where agents might contradict their earlier statements or forget key information.
- They can also forget the users instructions.



Our AI Data Engineer is a goldfish

#### **Current Solutions**

RAG-based memory agents

Retrieve previous dialogues that are related to the current request.

Pros: Accurately reciting previous dialogues. Cons: Retrieving memories that are not integrated with other dialogues to form meaningful context.



RAG based memory agents

#### **Current Solutions**

RAG-based memory agents

Retrieve previous dialogues that are related to the current request.

Pros: Accurately reciting previous dialogues. Cons: Retrieving memories that are not integrated with other dialogues to form meaningful context.



RAG based memory agents

#### **Current Solutions**

RAG-based memory agents

Retrieve previous dialogues that are related to the current request.

Pros: Accurately reciting previous dialogues. Cons: Retrieving memories that are not integrated with other dialogues to form meaningful context.

– Summary based agents

Summarise all previous dialogues to create a contextual summary to be used as memory. Pros: contextual and meaningful Cons: summary doesn't have a specific aim, thus important information can be lost or not captured.



Summary based agents

### **Intrinsic Memory Agents**

- MemoryAgent retains structured contextual memory.
- Memory can be pre-defined to make sure they are aligned with their and conversational objectives.
- Memory is updated after the agent generates a new output, maintaining consistency with previous context while keeping up-to-date with the new output.
- Memory is maintained on a per-agent basis. Thus, it allows agents to retain their autonomy, while still allowing general summarisations if desired.
- Allows other context reduction methods to reduce the cost of using AI models.



Produce a concise summary of the agreed-upon pipeline architecture, highlighting its key components and connections.



"layer": "Storage" "technology" ("Amazon S3", "storage\_types". [

Provide a high-level plan and rationale for the design, explaining why it is wellsuited for the given data and use case.



"layer": "Storage", "technology": "Amazon S3",

Estimate the cloud resources, implementation efforts, and associated costs, providing a rough breakdown and complexity rating.

"estimated\_infrastructure": {
 "compute\_units": "Elastic",
 "storage\_capacity": "Petabyte-scale",
 "monthly\_cost\_estimate": "\$15,000 - \$25,000"

"Platform": A cloud service provider's name if the cloud solution is the best, or "local server" if locally hosted servers are preferred.

"pipeline\_architecture": {
 "name": "Self-Driving Car ML Data Pipeline",
 "platform": "AWS Cloud",
 "primary\_components": [

Individual components: "Component 1": The first component in the pipeline framework, with AWS official name.

```
"layer": "Data Ingestion",
"technology": "Amazon Kinesis Data Streams",
"sources": [
        "6 Camera Streams",
        "1 LiDAR Point Cloud Stream"
],
"capabilities": [
        "Real-time sensor data streaming",
        "Millisecond-level processing",
        "High-frequency data capture",
        "High-frequency data capture",
        "Hulti-modal sensor integration"
],
"performance_metrics": {
        "throughput": "High-volume streaming",
        "latency": "<50ms",
        "data_types": ["Image", "Point Cloud"]
}
```

"Component 2": The second component in the pipeline framework, with AWS official name. Continue until all required components are listed.







Reached consensus faster, with shorter conversation chain: 4 proposals + 3 discussions (minimum required by the prompt)

Using claude-3.5-haiku, it costs 0.16\$

```
DataEngineerAgent (to chat manager):
I'll complete the PIPELINE_OVERVIEW.json file using the compret
```json
    "pipeline architecture": {
        "name": "Self-Driving Car ML Data Pipeline",
        "platform": "AWS Cloud".
        "primary components": [
                "layer": "Data Ingestion",
                "technology": "Amazon Kinesis Data Streams",
                "sources": [
                    "6 Camera Streams",
                    "1 LiDAR Point Cloud Stream"
                "capabilities": [
                     "Real-time sensor data streaming",
                    "Millisecond-level processing",
                    "High-frequency data capture",
                    "Multi-modal sensor integration"
                 "performance metrics": {
                    "throughput": "High-volume streaming",
                    "latency": "<50ms",
                    "data types": ["Image", "Point Cloud"]
                "layer": "Data Processing",
                "technology": ["AWS Glue", "Apache Spark"],
                "transformations": [
                    "Image normalization",
                    "Point cloud alignment".
                    "Metadata extraction",
                    "Data validation"
                "processing capabilities":
                    "distributed computing": true,
                    "scalability": "Horizontal scaling",
                    "data preparation": "ML-ready datasets"
   This comprehensive JSON provides a detailed overview of the self-dr
```

"layer": "Storage", "technology": "Amazon S3", "storage\_types": [ "Raw sensor data". "Processed datasets". "Model artifacts" "storage\_characteristics": { "durability": "99.999999999%", "scalability": "Unlimited", "cost optimization": "Tiered storage" "layer": "Machine Learning", "technology": "Amazon SageMaker", "ml\_workflow": [ "Model training", "Distributed training", "Real-time inference", "Model monitoring" "ml capabilities": { "multi modal\_processing": true, "inference\_endpoints": "Low-latency", "model management": "Automated versioning" "architectural principles": [ "Scalability", "Real-time processing", "Cost-effectiveness", "High reliability" "compliance standards": [ "GDPR", "CCPA", "Automotive data protection" "estimated\_infrastructure": { "compute\_units": "Elastic", "storage\_capacity": "Petabyte-scale", "monthly\_cost\_estimate": "\$15,000 - \$25,000"

### **Conversation Delegation Agent**

Without intrinsic memory:

- Can't follow propose → discussion → consensus steps.
- 2. Always allows only 1 proposal instead of the requested 4
- 3. It jumps back and forth between the propose and discussion phases.

With intrinsic memory:

- 1. Can follow the steps
- 2. Manage to count the number of proposals and discussions correctly.
- 3. Doesn't jumps between steps.



Conversation delegation is more consistent

# **Working Agents**

Without intrinsic memory:

- Individual agents' personal opinions are not maintained – other agents easily influence them.
- 2. During the proposal step, all the follow-up proposals are extended from the first one.
- 3. During the discussion step, they can't remember their proposal, but again, always follow up with the previous discussions

With intrinsic memory:

- 1. Able to create their proposal, I can see different components being proposed.
- 2. Discussion can compare their own proposals and others.



Agents can focus more on their own proposals

### Conclusion

- Can maintain contextual consistency and retain relevant and important pre-defined information;
- Can improve efficiency in using the group chat based agentic AI framework – solves problems with a shorter amount of dialogues;
- Can improve conversational and output quality;
- Can reduce the cost of using LLM models in group chat based agentic Al framework.



#### The Alan Turing Institute



Data-Centric Engineering at Scale: Developing an Al Data Engineer

Thanks to: Ting Su, Sizhe Yuen and Ziyang Wang (they did all the hard work)

asobey@turing.ac.uk

