Context Graphs to bridge the Decision Intelligence gap and build a Digital Twin

Building dynamic local and global Context Graphs to empower Digital Twins for hyper-personalization in financial services

Persistent Systems Ltd. dattaraj_rao@persistent.com https://www.linkedin.com/in/dattarajrao/

We present the vision of the context graph as a unified layer of institutional knowledge to bridge the decision intelligence gap in an enterprise. Today, with the rise of large language models and agentic AI, we see high adoption rate of agents to perform diverse enterprise tasks. However, for these agents to deliver exponential value, it is especially important to curate business context and provide it in a structure that agents can consume. This is where the context layer and the context graph play a significant role. The context graph curates knowledge from different source systems, data lake-houses, business documents, and institutional knowledge maintained in the Enterprise in different forms. Using agentic perception, this knowledge is analyzed, connected to each other, and a dynamic context graph is developed. The dynamic nature of the graph helps adapt to ever-changing business dynamics and proactively respond with right course of action. Given the right context, the agents are empowered to make decisions and drive value beyond traditional automation with high-value reasoning and decision making. A key element of the context graph is decision intelligence. Historical decision traces and reasoning steps of agents that have driven decisions, serve as very useful knowledge which is often not considered while bringing context together. We envisage decision and reasoning traces as a key element of the context graph and show how these can ground future decisions made by agents. Taking an example of a finance digital twin, curating domain knowledge using an open knowledge format, we represent it as a graph and use this connectivity information to enable customer specific hyper personalization. We envision the context graph to grow into the core heart of the Enterprise turning into a true "moat" for the modern Enterprise.

CCS CONCEPTS •**Applied computing~Electronic commerce~E-commerce infrastructure •Information systems~Information retrieval~Retrieval models and ranking~Language models•Information systems~Information retrieval~Retrieval models and ranking~Probabilistic retrieval models•Information systems~Information retrieval~Document representation~Ontologies•Information systems~Information retrieval~Specialized information retrieval~Environment-specific retrieval~Web and social media search**

Additional Keywords and Phrases: Open Knowledge Format, Personalization, Graph

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

1 INTRODUCTION

In today's world, as models are becoming more commoditized, organizations are seeing the Context layer as the true "moat" that will provide differentiation. True measure of value from artificial intelligence (AI) will come not from highly generalized benchmarks, but on highly specialized outcomes specific to your line of business. No two organizations are the same. A unique Context layer is what will enable specific value offering for the business and we will see that dependency on a particular commercial large language model (LLM) may not be as stringent as was thought couple of years back. In short, a commodity cheap LLM with the right organizational Context (both in content and structure) can almost always exceed the performance of a costlier model working on generic data.

Here we discuss details on building a context layer which curates data from multiple diverse data sources (structured, unstructured, SaaS, events) in an organization. Using methods like mapping to a domain or business ontology and human-in-loop verification, the context is validated and curated for use by agents. Because a typical organizational context will include many concepts and relationships, we propose a graph data structure to store these entities and relations. Context expressed in form of a graph can be easily consumed by AI agents powered by LLMs. Instead of systems like retrieval augmented generation (RAG) [1] where we give raw text chunks to LLMs to figure out a response, in our context engineering approach, we curate a context and pass on to agents in formats like JSON or YAML. Embedding this context into the LLM input token stream greatly improves the response quality and helps capture essential domain logic. Moreover, enhancing the context with decision intelligence and history of decisions and their impact further empowers the agents to take more accurate decisions. We also propose a separation of local and global context by the use case – to enable a focused approach to passing intelligence to the LLM and avoiding any hallucinations with the model trying to figure out these relations and decision logic.

The curated context is then consumed in a digital twin layer which emulates real world objects like humans, organizations, products, etc. Creating a digital twin using a context encapsulation format like JSON-LD or YAML-LD (LD = linked data) or the recently published open knowledge format (OKF), helps work with context specific to the entity represented by the twin. This enables connecting specific machine learning (ML) models to the twin for specific forecasting and predictions. Also, by capturing empirical rules, decision points, reasoning traces and feedback from production helps enrich the twin to reflect real-world scenarios and dynamically capture the organizational domain knowledge. The twin thus developed is true representation of the asset and can be executed in any agent runtime powered by commodity LLMs. Responsible AI guardrails around security, privacy and access control are enabled in the agent harness. A representation is shown in fig 1 below.

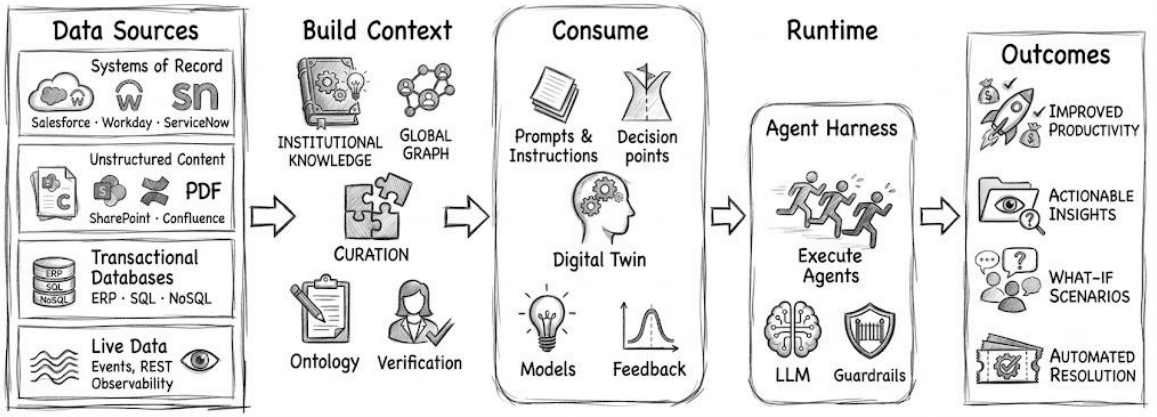

Figure 1. Conceptual view of the Context curation and consumption pipeline (Source: by Author)

2 UNDERSTANDING AGENTS

Agentic AI has transformed modern enterprises, and we see agents enabling many key tasks. Agents provide a distinct advantage over traditional methods like robotic process automation, and API based automation. Agents have the capability to do reasoning and self-critique [4]. Thus, they can understand if wrong analysis or part is taken and can self-correct themselves, as long as the right context is provided. Agents build on top of traditional LLM calls and they form reasoning loops with to enable advanced cognitive capabilities. Agentic cognitive model shown in Figure 2 below enables key patterns of perception, reasoning, tool calling and reflection. Perception enables agents to understand a variety of inputs like text, structured data, images and make sense of it. This is enabled via powerful and multi-modal models. This capability enables a chatbot to understand user intent and figure out what steps to take to solve a problem. Agentic reasoning helps create a plan on how to solve the problem at hand and what steps to take. Usually, the conversation history is passed to the agent as short term memory to assist in decision making. Then the agent executes a reasoning loop that keeps calling multiple tools as needed. The thought-action-observation loop executes, and the agent can understand tool responses and decides on what steps to take next. The agent also has access to new capability with the model context protocol (MCP) which allows it to call external tools. These tools are like superpowers for the agent, allowing it to do specialized tasks like create presentations, access the Internet, or send emails. Finally, when the task is completed, a response is compiled and presented to the user. This loop enables the agent to understand the response and reason over it, correct any errors that may have occurred and give a comprehensive response to the user or the next agent in the workflow.

A key pattern in the agent cognitive model is memory [4]. Memory is a dynamic construct for an agent that helps it provide necessary information for the large language model to do reasoning and generate accurate results. Memory could be a working memory, which includes conversations and chat histories. Memory includes past interactions by the user captures of state, etc. Semantic memory contains the different concepts and relationships usually captured and triples or knowledge graph which helps the agent understand complex real-world relations. Procedure memory is a key aspect of the agent memory which contains skills on performing domain specific tasks. This is often tied to the tools but captures the domain knowledge.

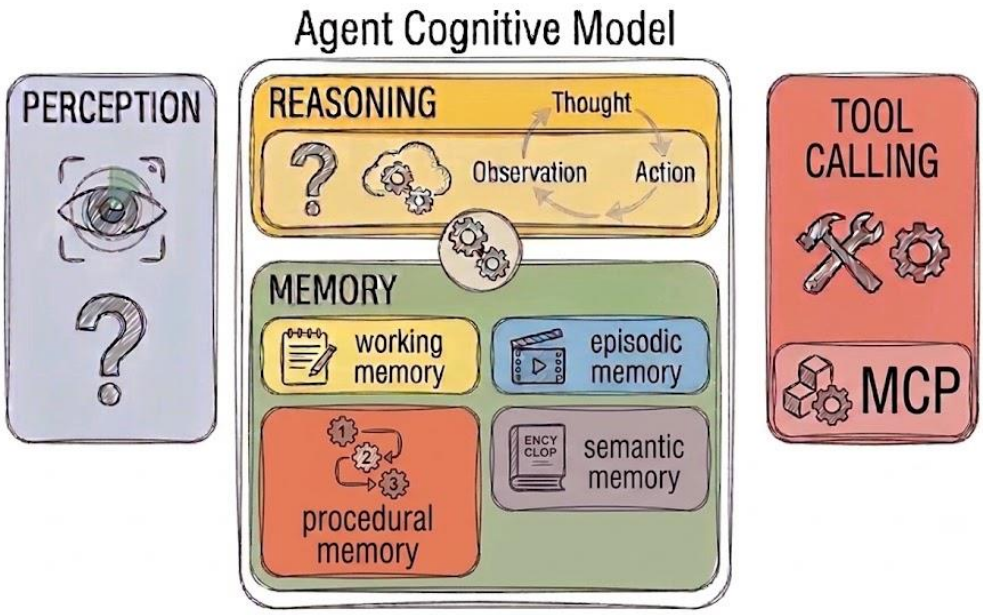

Figure 2. Agent cognitive model with 4 key patterns (Source: by Author)

3 COMPONENTS OF CONTEXT

Context is the input to an AI agent. It is the dynamic memory of the agent that helps it use the large language models to generate useful responses relevant to the business context. It encapsulates all details the agent needs to carry out a given task. The quality of context decides the quality of output generated by the agent.

In traditional retrieval augmented generation (RAG) [3] the context was pretty much prompt instructions, along with text chunks selected by searching a vector database and ranking chunks by relevance. However, in more mature agent systems context contains many elements organized to influence the agent outputs. The context is compiled by bringing together prompt instructions, conversation history, list of tools available, historical decisions and state, retrieved knowledge, etc. to help agent get a complete perspective for taking the next steps. Additional information can help enrich the prompt for agents like a history of decision traces which previous agents or runs followed to arrive at a decision. For example, if we have an agent that recommends risk of approving a credit decision, if there is validated information of past reasoning how agents arrived at a decision, that can help agents make better decisions. All these inputs can be arranged as a graph of connected data to enable easy understanding and analysis by agents. This approach is shown in figure 3 below.

As shown in the figure, different elements are combined to create context. Unlike prompt engineering which involved a lot of human cognitive load to create a prompt that the agent can benefit from, context engineering involves combining data together into a context graph. There are domain specific programs which are similar to traditional prompt engineering. Then there is the state and reasoning which is usually captured by taking production traces of agents to understand the state and putting it into the context graph. The structure and structure data becomes an important part of the concept graph. Here usually the approach is not to do any duplication and embedded relevant data as needed from the source system into the context. The context also contains a catalogue of tools that are available to the agent to take certain actions. This could also include specific skills which will augment the agent's procedural memory.

Having compiled this context is not enough. As in the RAG pattern if we just push this context into a prompt and send it to the agent, the agent will have difficulty looking at all the diverse information. The recommended approach is to leverage an industry or business standard ontology to create a graph of all the data so that the agent can know how to look for relevant information. This context graph becomes the dynamic memory of the agent and can be used by the different large language model calls to supplier relevant information. This pattern is the essence of the agent reasoning loop.

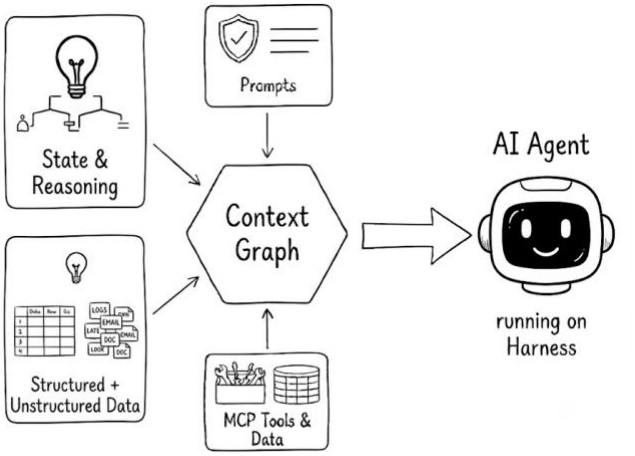

Figure 3. Different elements that combine to form the context (Source: by Author)

4 CONTEXT IN THE ENTERPRISE

In a real enterprise there are numerous systems of record, knowledge and engagement. These systems carry their own data and context specific to the problem they are solving. For example, a customer relationship management system (CRMS) like Salesforce may carry Sales context. ServiceNow may carry data and context on support tickets and state of the enterprise assets. Enterprise resource planning (ERP) systems carry finance context. Each system may employ their own analytics or agents to solve specific problems. But to answer enterprise level problems we need to combine this context into a global context used by agents to answer enterprise level questions. Traditional approaches involved using integration patterns like API or enterprise services bus (ESB) to integrate systems together. However, these often become point integration solutions which cannot really adapt and scale. With a unified context layer, the individual context from local context can be assembled to a global context. On top of that this can be enriched with additional knowledge like institution knowledge (rules of thumb), routing logic, domain skills and decision history. These can be out together as a context graph mapped to business KPI (key performance indicators). Reasoning traces from, historical decisions can help enrich context for agents to make cognitive decisions like a human. This concept is shown in figure 4 below.

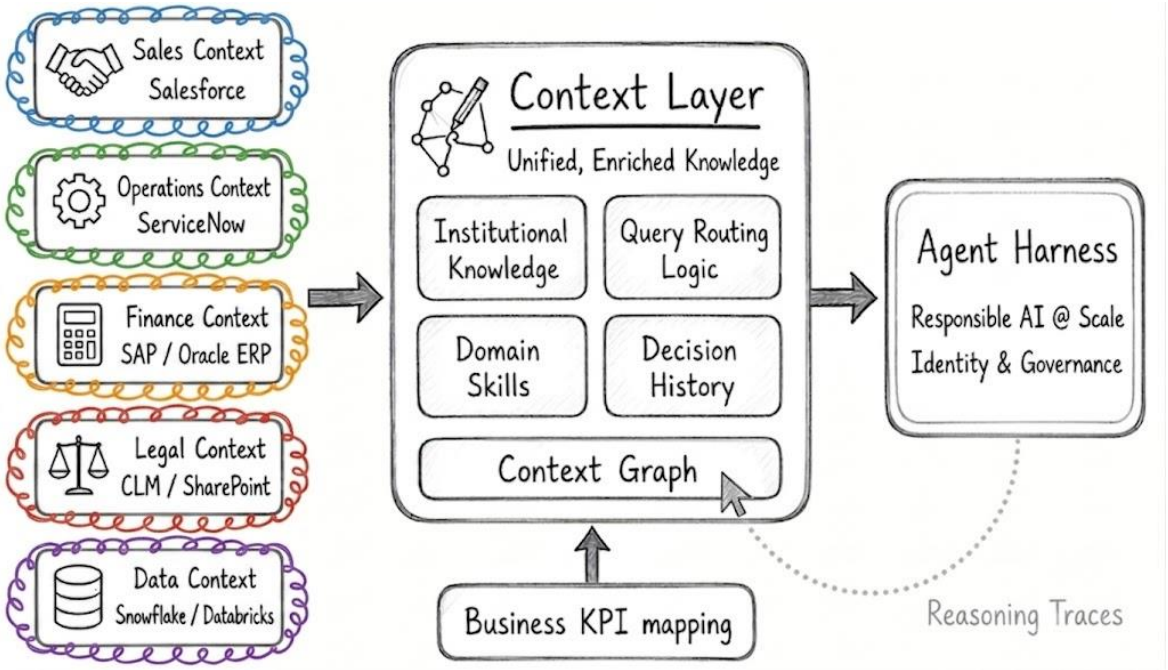

Figure 4. Context from multiple source systems combined into a context layer (Source: by Author)

5 CASE STUDY: FINANCE DIGITAL TWIN

Now let us take a real-world example of a finance digital twin and create a context graph to drive advanced analysis like what if scenarios and hyper personalization using AI agents. We consider a system where we have transactional data of several customers, which we use to create a profile or a digital twin of the customer. Objective will be to ask the digital twin specific questions on what recommendations we can provide to the customer. Unlike traditional, digital twins, which are heavily dependent on statistical models, our approach will leverage a context bundle, which will store the events and historical decisions of the subject. This knowledge will help us make informed decisions on what sort of recommendations to provide to the customer. First, let us look at the data of the customers, in a dashboard shown below in Figure 5. The customer data is assembled from several source systems and compiled together to provide a unified dashboard, as is a common pattern in different Enterprises.

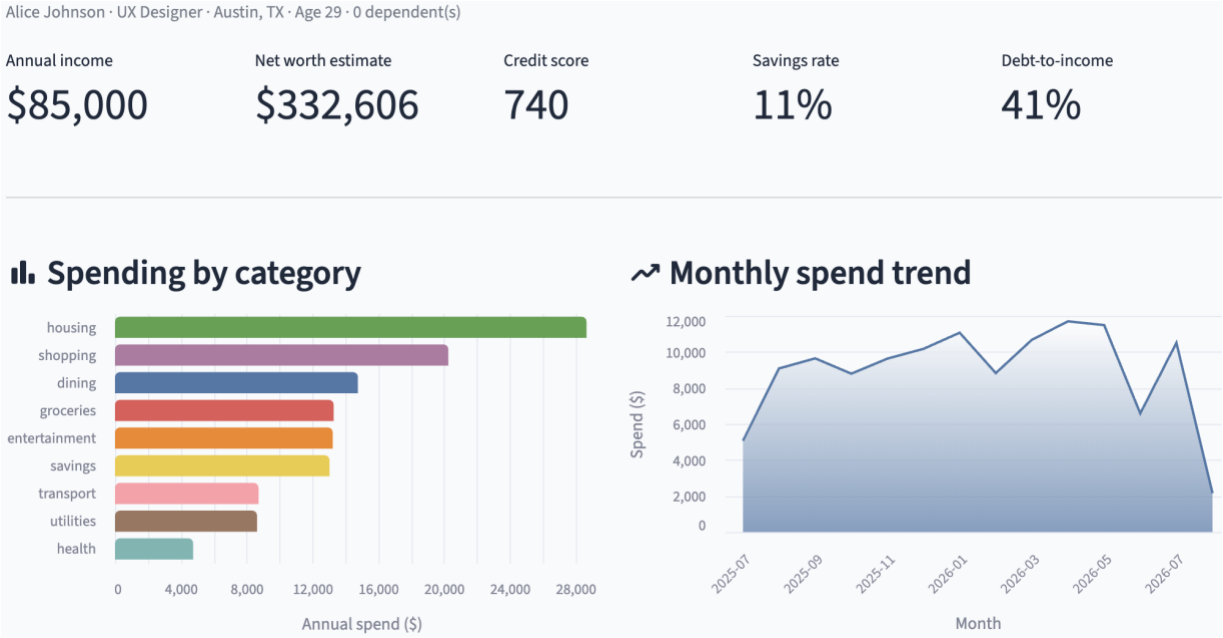

Fig 5. Dashboard showing data collected for a given customer – descriptive statistics

Next, the analyze the transactions for the particular customer and try to understand different events like job changes or major purchases that have taken place and try to capture it. To capture these events, we use a graph data structure, and represent this as a graph shown in figure 6 below. We see events like credit card applications, gym, memberships, loan applications, etc., which are flagged and recorded to help the agent make informed decisions. We can use an ontology to standardize on the definitions and data structure of these events and decisions. But the idea is to have a way to capture these decisions and events into a system, which can feed to AI agents. When these events and decisions seem trivial to a human, for an agent powered by LLMs, these are extremely valuable and can drive major insights into making a particular decision. A context graph bundle specific to the customer is ingested by the agent when making specific decisions on the

customer profile. Empowered with this context, the agent has enough background to make a highly hyper-personalized decision and suggest recommendations.

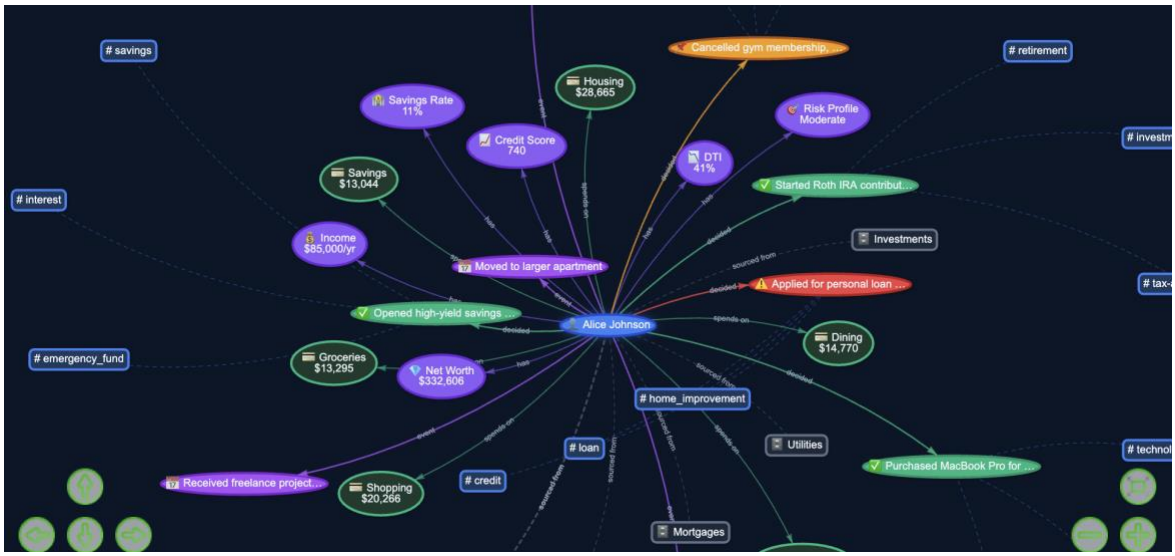

Figure 6. Building a context graph capturing major events and decisions

Let's look at specific scenario in the global context graph below in Figure 7. Major events and decisions are captured in eclipse nodes in graph. Red nodes are the decisions that have gone negative like rejection of credit decision for personal loan. Positive decisions are shown in green. This decision information is passed to the agents using a format like JSON or YAML. Further ahead we will show structure of the project in the open knowledge format (OKF). OKF is an open standard which is an implementation of the LLM Wiki pattern introduced by Andrej Karpathy [8]. OKF bundles are pretty flexible; you can design the schema as per the specific project that you are building. You can connect it to an existing ontology or build one. The key feature of OKF is that it is Agent and Human consumable [7]. Agents can read and write to OKF bundle, because essentially the format is JSON and YAML.

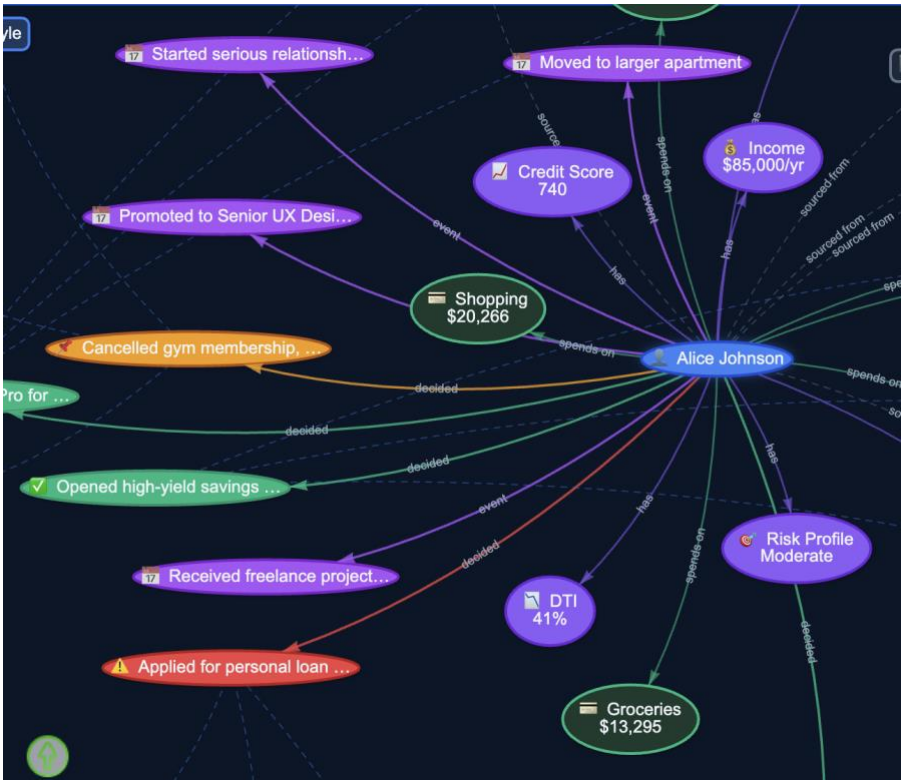

Figure 7. Building a context graph capturing major events and decisions

Now that we have this context graph curated, we can use this to take strategic decisions on the customer and ask questions and conduct what if scenarios. In this example, we will use this knowledge to suggest product recommendations based on existing catalogue of products known beforehand. Figure 7 shows a snapshot of some of the recommendations provided by agents running on this context graph. They look at the state and decision traces for the particular customer and reason to recommend matching products. Along with the matching products, a reasoning place is provided to suggest why the recommendation was given. Unlike statistical recommendation systems which provide a confidence score, here a full text trace is provided, which can be understood by a human. Along with the recommendation of propensity score is also provided to help enlist get a understanding on which product would be a better fit for the customer. The agent uses this context graph to calculate a propensity score for several products in our catalog like premium credit card, investment product, personal loan, etc. The scores are not only calculated using some unverifiable statistical measures. We use concrete concepts and relationships to understand dependencies and preferences. Agents use this knowledge and use empirical methods to calculate propensity scores for several products per customer. This level of hyper-personalization would not be possible without agentic AI. Also, Figure 9 below shows relationship of events to propensity scores, expressed as a graph.

We see events like getting a freelance project and promotion captured and their respective influence on recommending specific products. Unlike traditional recommender systems [9][10] which rely on statistics for calculating product propensity, here we leverage the power of unstructured text as graph nodes to calculate the propensity.

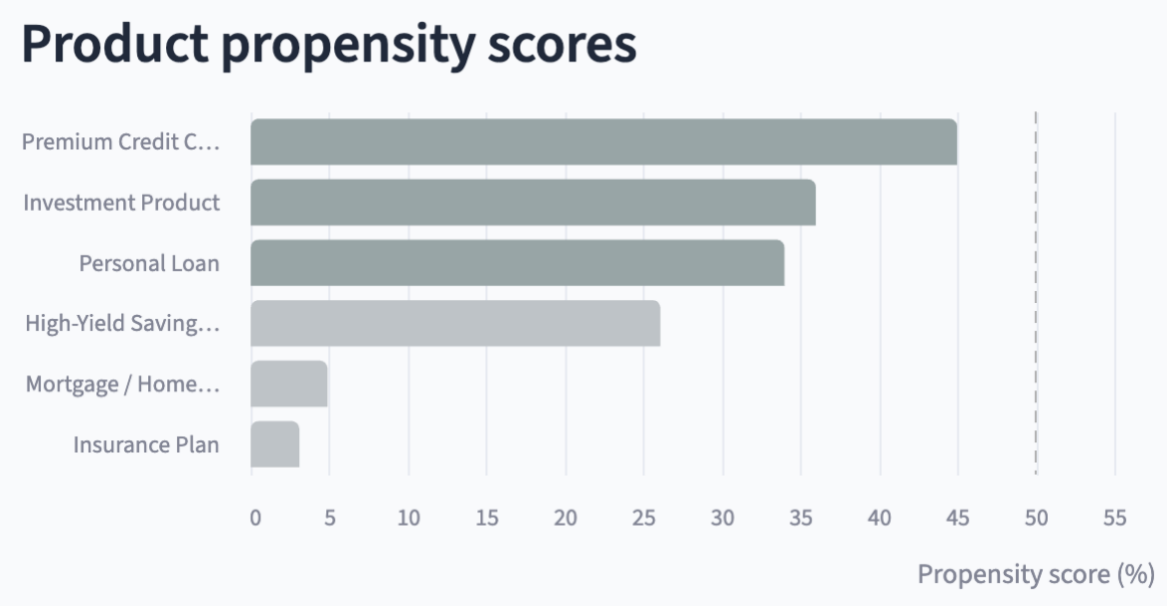

Figure 8. Propensity score for several product offerings with respect to the customer (Alice)

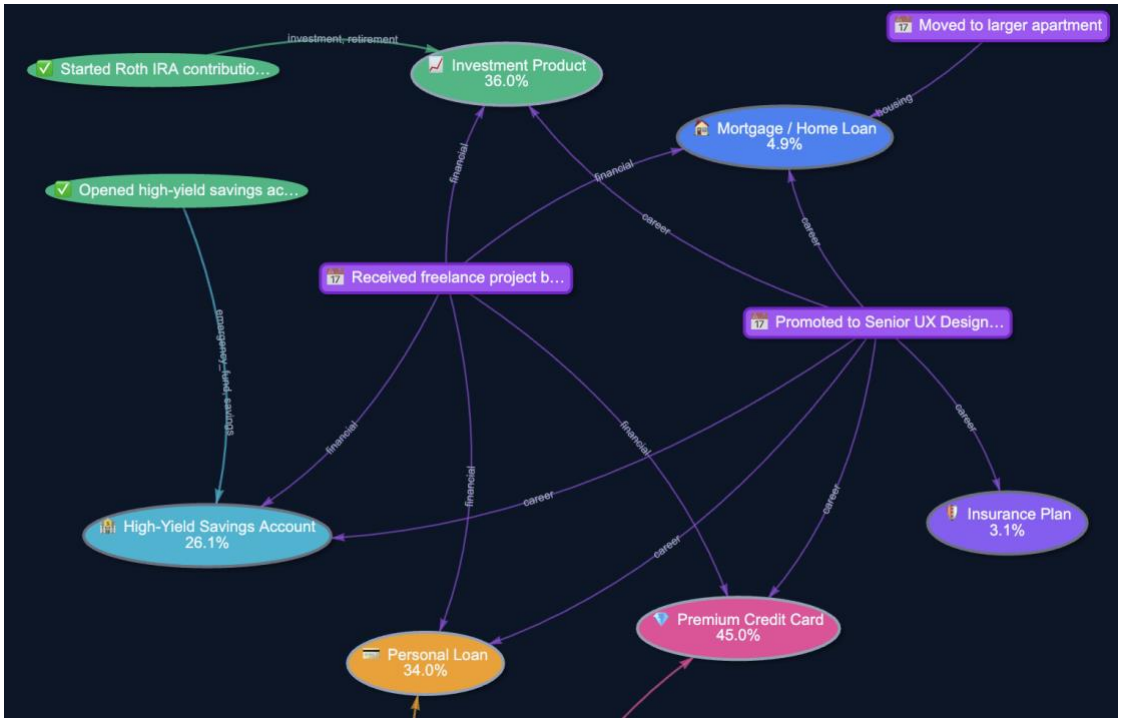

Figure 9. Explainability of propensity scores to events and decisions

6 DIGITAL TWIN CONTEXT BUNDLE

Next, we will use an agent harness to study transactional data from source systems by connecting via methods like model context protocol (MCP). We will extract knowledge and organize in a folder structure where we capture essential insights per user in JSON or YAML formats. We use open knowledge format (OKF) to use industry standard way to represent entities and relations. Figure 3 below shows an example of the folder structure we use. By analyzing data from databases, we generate insights like financial summaries, profile information and key decisions taken and store in knowledge base. This will be used by agents to search for knowledge and generate insights.

The knowledge base uses a hierarchical file structure to store insights at different levels. The hierarchy of markdown and yaml files make it easy for agents to consume this context. Also using progressive disclosure pattern [12], the agents can selectively load right files in context avoiding overrunning the context. For greater scalability, we can leverage graph database like Neo4j or AWS Neptune to store this context bundle. Also, there are specialized products like Google Dataplex [15] which can help manage a larger context graph.

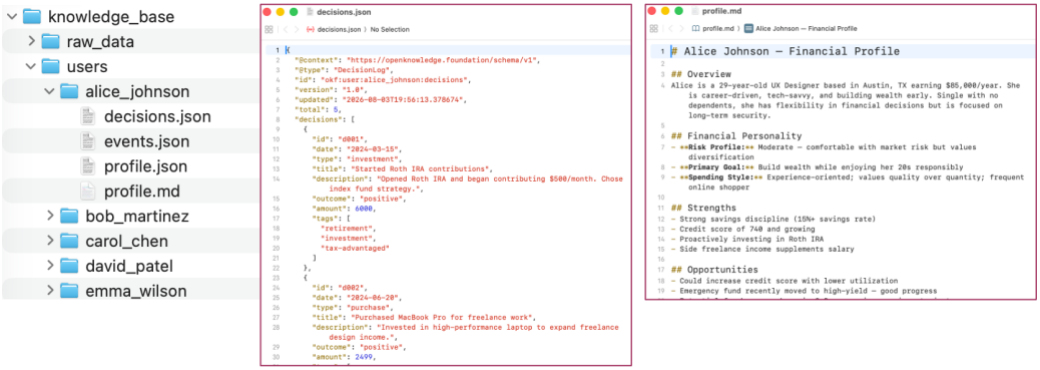

Figure 8. Example of digital twin bundle with curated JSON and YAML files (Source: by Author)

7 CONCLUSION

We see that using a context graph with agents enables encoding of complex domain knowledge as a graph data structure in formats like open knowledge format (OKF). This knowledge is agent as well as human consumable. Agents can read and write to the OKF bundles. We showed with an example of finance digital twin, how we can build a customizable customer specific OKF bundle which captures the events and decision logic mined from data sources. This context graph is complementary to the source systems and attempt should be not to duplicate the data. However, this additional decision intelligence greatly helps organize knowledge and empowers the agents to make more effective and accountable decisions with full lineage to sources.

REFERENCES

[1] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In Advances in Neural Information Processing Systems 33 (NeurIPS 2020). arXiv:2005.11401. https://doi.org/10.48550/arXiv.2005.11401

[2] Edge, D., Trinh, H., Cheng, N., et al. (2024). From Local to Global: A Graph RAG Approach to Query-Focused Summarization. arXiv. https://doi.org/10.48550/arXiv.2404.16130

[3] Lewis, P., Perez, E., Piktus, A., et al. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *arXiv*. https://doi.org/10.48550/arxiv.2005.11401

[4] Lilian Weng. 2023. LLM Powered Autonomous Agents. Lil'Log. Retrieved August 20, 2026 from https://lilianweng.github.io/posts/2023-06-23-agent

[5] Han, H., Wang, Y., Shomer, H., et al. (2024). Retrieval-Augmented Generation with Graphs (GraphRAG). *arXiv*. https://doi.org/10.48550/arxiv.2501.00309

[6] Fuller, A., Fan, Z., Day, C., & Barlow, C. (2020). Digital Twin: Enabling Technologies, Challenges and Open Research. *IEEE Access*, *8*, 108952–108971. https://doi.org/10.1109/access.2020.2998358

[7] Andrej Karpathy. 2026. LLM Wiki. *GitHub Gist*. Retrieved August 21, 2026, from https://gist.github.com/karpathy/442a6bf555914893e9891c11519de94f

[8] Google Cloud Data Cloud Team. 2026. How the Open Knowledge Format can improve data sharing. Google Cloud Blog (June 2026). https://cloud.google.com/blog/products/data-analytics/how-the-open-knowledge-format-can-improve-data-sharing

[9] Covington, P., Adams, J., & Sargin, E. (2016). Deep Neural Networks for YouTube Recommendations. In Proceedings of the 10th ACM Conference on Recommender Systems (pp. 191–198). ACM. https://doi.org/10.1145/2959100.2959190

[10] Jannach, D., Manzoor, A., Cai, W., & Chen, L. (2021). A Survey on Conversational Recommender Systems. *ACM Computing Surveys*, *54*(5), 1–36. https://doi.org/10.1145/3453154

[11] Hogan, A., Blomqvist, E., Cochez, M., D'amato, C., Melo, G. De, Gutierrez, C., Kirrane, S., Gayo, J. E. L., Navigli, R., Neumaier, S., Ngomo, A.-C. N., Polleres, A., Rashid, S. M., Rula, A., Schmelzeisen, L., Sequeda, J., Staab, S., & Zimmermann, A. (2021). Knowledge Graphs. *ACM Computing Surveys*, *54*(4), 1–37. https://doi.org/10.1145/3447772

[12] Sartaj, H., Ali, S., & Gjøby, J. M. (2024). MeDeT: Medical Device Digital Twins Creation with Few-shot Meta-learning. ACM Transactions on Software Engineering and Methodology. https://doi.org/10.1145/3708534

[13] *ACM Trans. Interact. Intell. Syst.* 10, 4, Article 29 (October 2020), 32 pages. https://doi.org/10.1145/3374218

[14] Google Cloud. 2026. Introduction to Dataplex. Retrieved August 5, 2026 from google.com