Russell Jurney
Founder

Created: June 2, 2022
Updated: August 21, 2022

1555 Botelho, Dr.
Ste 511
Walnut Creek, CA 94596

rjurney@graphlet.ai

Property Graph Factory

Extract, Transform, Resolve, Model, Predict, Explain

This document is an *essay about building and using enterprise knowledge graphs* in the form of property graphs after reflecting on being CTO at three knowledge graph companies and outlines a gap in the open source ecosystem and market where a *Property Graph Factory* fits to improve outcomes in risky, expensive large knowledge graph projects. These projects can cost eight figures and fail to provide a return on investment. An effective property graph factory can address the problems every enterprise knowledge graph AI project encounters to cut the costs of building applications based on graph intelligence in half. This document covers the process of building a knowledge graph using the most popular open source stack before outlining the problems in this process that kill projects and the features in Graphlet AI that will solve them.

I focus on process and technology rather than use cases because **this essay is a net with which to gather use cases that fit or alter the factory.** I would appreciate feedback on this document in the form of questions, comments and suggestions as I develop this idea. This is the philosophy of my new knowledge graph consultancy Graphlet AI (website in progress). It is the driving ethos of our practice, it could be an open source project under Apache governance or it could be a startup that disrupts the graph database and knowledge graph markets by unblocking their growth by offering a platform that cuts the cost of applications driven by enterprise knowledge graphs in half. **Which do you think it is?**

INTRODUCTION

The workflow for the property graph factory described below defines the process that enables enterprises to embrace and extend the data platforms they already use to build an enterprise knowledge graph with a uniform ontology that represents the domain in which their business operates. On top of this domain specific graph they can then use graph intelligence to add the types of edges in the solution space that solve their business problems. In addition to network construction, a KG factory would allow companies to easily do graph search, graph machine learning, and the rapid development of graph neural networks

(GNNs) to build a high fidelity model of their problem domain which can drive the models that automate their business processes under a Software 2.0 model as defined by Andrej Karpathy and Ratner, Hancock and Ré.

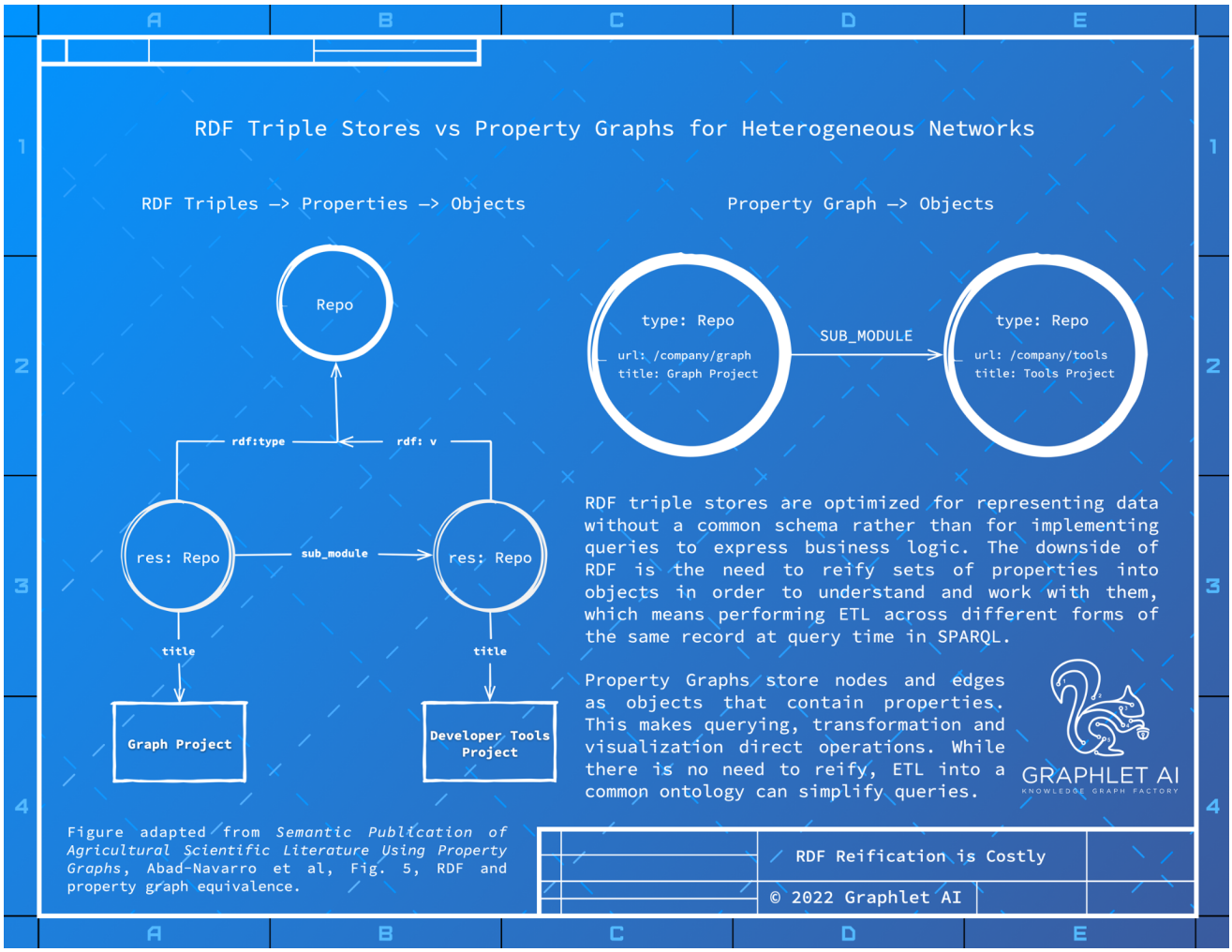
Heterogeneous knowledge graphs offer a rich domain in which to model your market and its problems. When combined with heterogenous graphlets and motifs, motif search against null models and Motif Graph Neural Networks, they offer an efficient way to create automated solutions using representation learning that builds on top of the domain expertise previously expressed by an organization in the form of graph queries. These distributed representations of the problem domain of a market are the core assets that drive FAANG companies... and is causing other enterprises to follow them in transforming their companies to be model first, rather than code first.

Without a factory to make the workflow more efficient, innovative enterprises spend eight figure budgets on knowledge graph projects that fail to provide a return on investment. If a property graph factory could make the process above 25%, 50% or 75% faster it would create an enormous amount of value for its users. Spending $5M instead of $10M to achieve an improved outcome constitutes enterprise value.

The graph database and knowledge graph markets have long been frustrated by the failure of the semantic web and many enterprise knowledge graph factories are closed silos based on legacy technology. I believe the market will explode once the Python open data stack is easy to apply to model a problem using graph machine learning. It is the mission of Graphlet AI to build a Property Graph Factory that brings network science and graph machine learning into the operations of enterprises across the globe.

PROBLEM DEFINITION

The knowledge graph and graph database markets have long asked themselves: why aren't we larger? The vision of the *semantic web* was that many datasets could be cross-referenced between independent graph databases to map all knowledge on the web from myriad disparate datasets into one or more authoritative ontologies which could be accessed by writing SPARQL queries to hop from one knowledge graph to another. The reality of dirty data made this vision impossible, as Cory Doctorow outlined in his essay Metacrap.



*Semantic Web was a dead end. People think in terms of objects - not attributes they must **reify***

In reality most time is spent cleaning data - graph data isn't in the format you need to solve your business problems. You have multiple datasets in different formats, each with its quirks. You need to deduplicate data using entity resolution - an unsolved problem in commercial tools for large graphs. Even once you merge duplicate nodes and edges, you rarely have the edge types you need to think in terms of your problem domain to make a problem easy to solve.

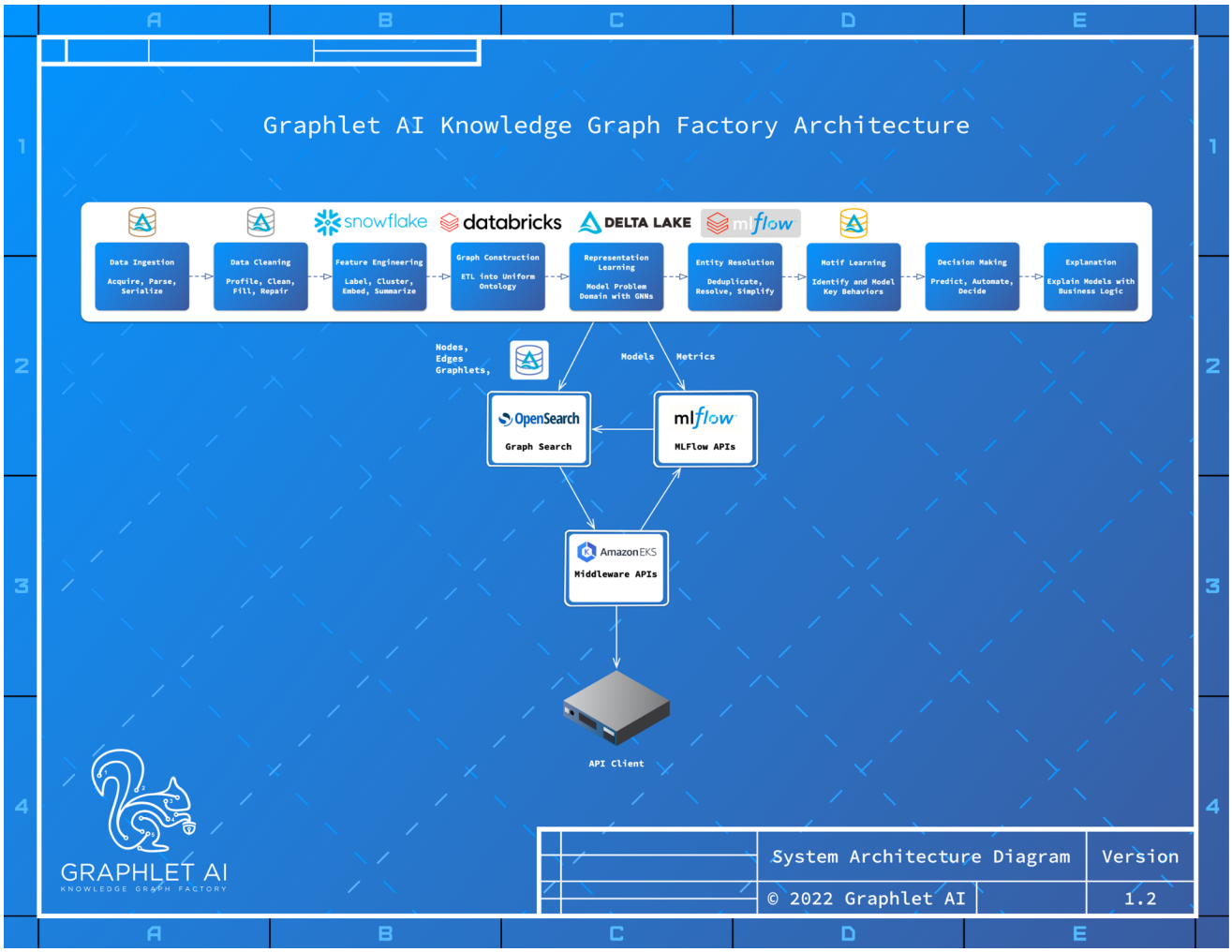
It turns out the most likely type of edge in a knowledge graph that solves your problem with analysis is defined by the output of an entire program - a program in Python which employs machine learning. For large graphs, this program needs to be run on a scalable platform based on commodity machines like Databricks or Snowflake (which are improving using GPUs) and extend rather than rebuild and isolate within to provide an excellent developer experience. To create new edges in the domain of your solution, you need to compare candidate pairs nodes that are candidates for edges in an efficient manner. Google Grale outlines a simple blocking or reduce mechanism for building a graph that solves your business problem

using simple graph algorithms like connected components or other types of querying or pattern matching. It can also be the basis for graph machine learning on a more refined graph.

This document outlines the broad requirements for a *property graph factory* that embraces the most popular open source technologies for machine learning, big data and information retrieval and that meets the needs of users in enterprises by extending the data platforms they already use. This confluence of factors are they key to unlocking the potential of the knowledge graph and graph database markets under the auspices of *graph intelligence*.

PLATFORM ARCHITECTURE

The system architecture for this platform is optimized for large scale datasets and their corresponding knowledge graphs by adopting the Delta Architecture, supported by cloud services for search, graph retrieval and machine learning operations. The data pipeline is operated under an ingest / build / refine / publish / query model of data processing. As much data processing as possible is performed in a horizontally scalable, batch processing system such as Databricks (PySpark) or Snowflake. The platform builds on top of the open Python data stack, which is utilized for the purposes of Extract Transform Load (ETL), feature extraction or engineering, construction of a knowledge graph with a uniform ontology for a given problem domain, graph representation learning, entity resolution and the computation of network motifs or *heterogenous graphlets* which are then folded into node-level representations as *heterogeneous graphlet minors*.



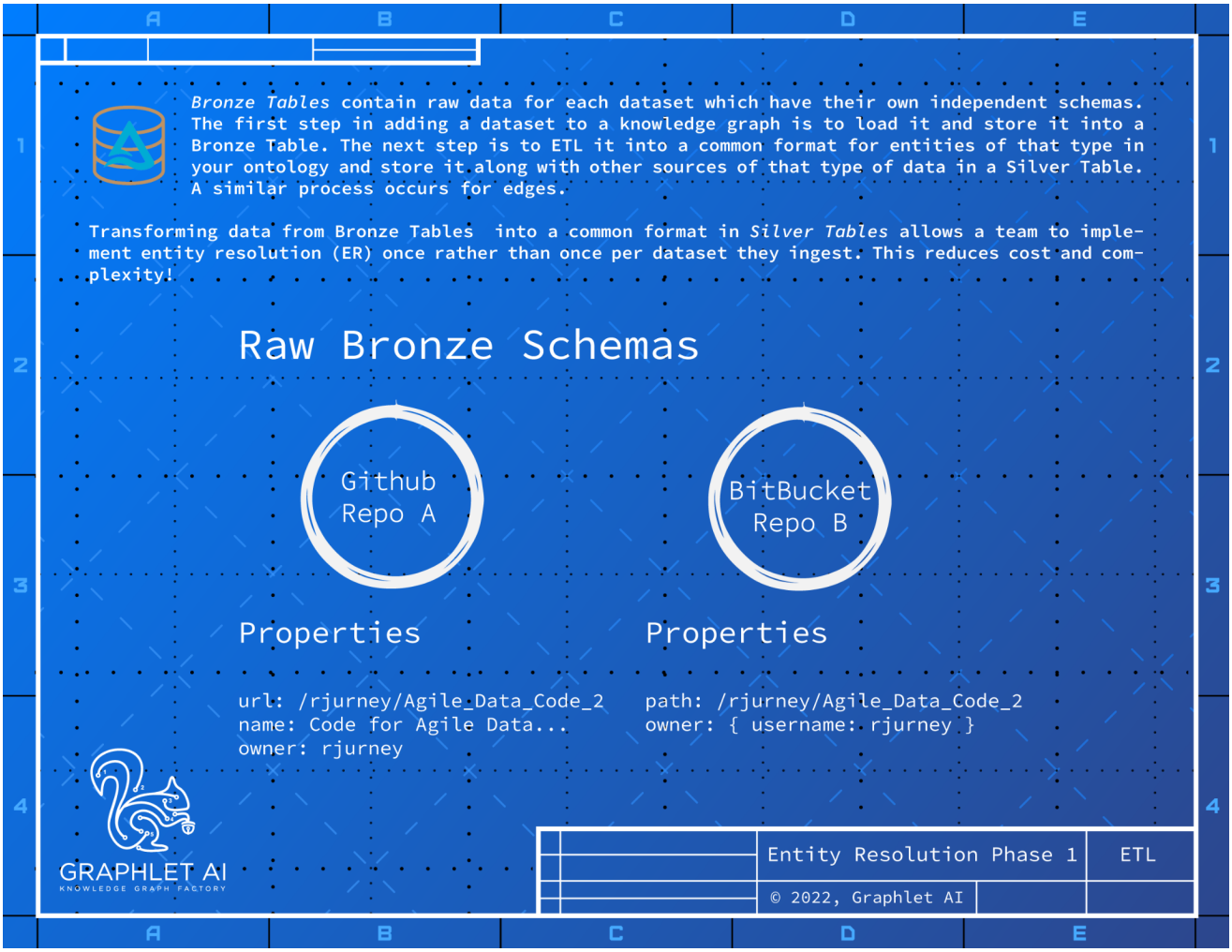
KNOWLEDGE GRAPH CONSTRUCTION

This section outlines the phases of the knowledge graph construction process for large knowledge graphs using distributed systems and deep learning. This process is echoed in Databricks marketing on using big data for financial services. Once we outline the process, in the next section we will discuss problems one encounters in knowledge graph construction before outlining a set of solutions to these problems Graphlet AI will build.

Ingestion

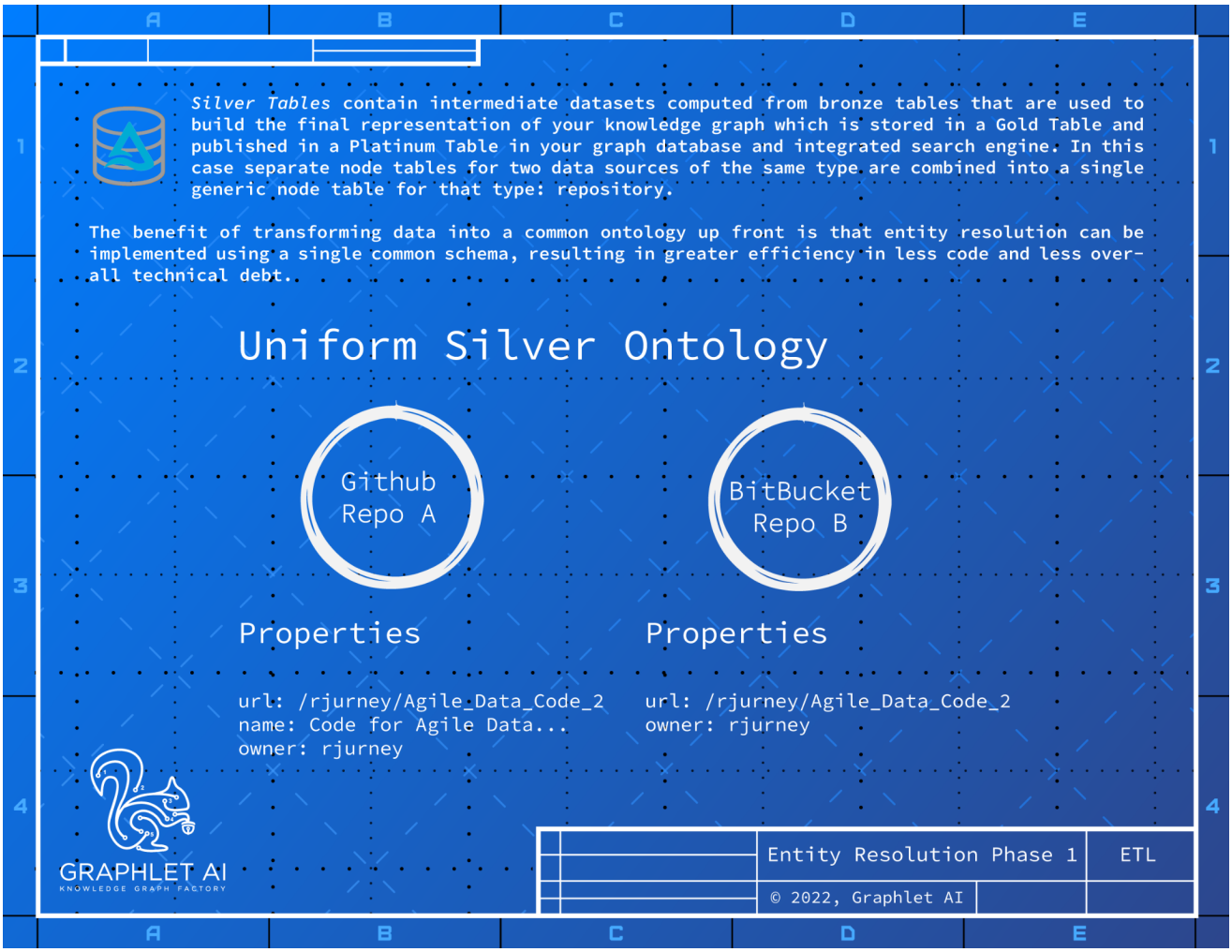
Datasets that make up a knowledge graph are ingested onto bulk storage systems such as Amazon S3, Google Cloud Storage (GCP) or onto parquet based tables with a version tracking component such as Delta Tables or Apache Iceberg, which has broad language and library support. This allows version tracking of datasets and their varying metadata along with source code and predictive models over time so the

pipeline producing the knowledge graph from scratch is always uniquely versioned from end-to-end. Nodes and edges can be stored in their own intermediate silver tables.



Data Cleaning and ETL

Data from different sources about the same thing often contain different schemas, and for efficiency's sake it is necessary to transform - rather than link - the data into a single ontology representing your problem domain. Multiple datasets need to be transformed into a single, generic form that fits the query and access patterns for your application - for example Github, GitLab and BitBucket repositories can become Repos with a type field referring to the source.



Tools for building Python classes for each domain's application ontology are required to make ETL'ing multiple bronze datasets into PySpark DataFrames or tables in SnowFlake. Classes for each type in the ontology can be created from base classes provided by the system that use Python decorator functions to define DataFrame or table schemas in a way that encourages code reuse.

Entity Resolution

The landmark paper *Deep Entity Matching with Pre-Trained Language Models* defined an encoding mechanism for semi-structured records and embedded these representations using Sentence Transformers to create vector representations of entities for entity matching. This turns out to be an excellent starting point as an encoding mechanism for many graph ML tasks. Matching code for entity resolution is available as part of the papers' authors' implementation ditto. This sentence transformer vector encoding can be used for blocking and even matching via semantic similarity between structured records using cosine similarity with a threshold, although a different model is used for matching via a fine-tuned classifier in the ditto paper.

Blocking with Locality Sensitive Hashing (LSH): Google Grale

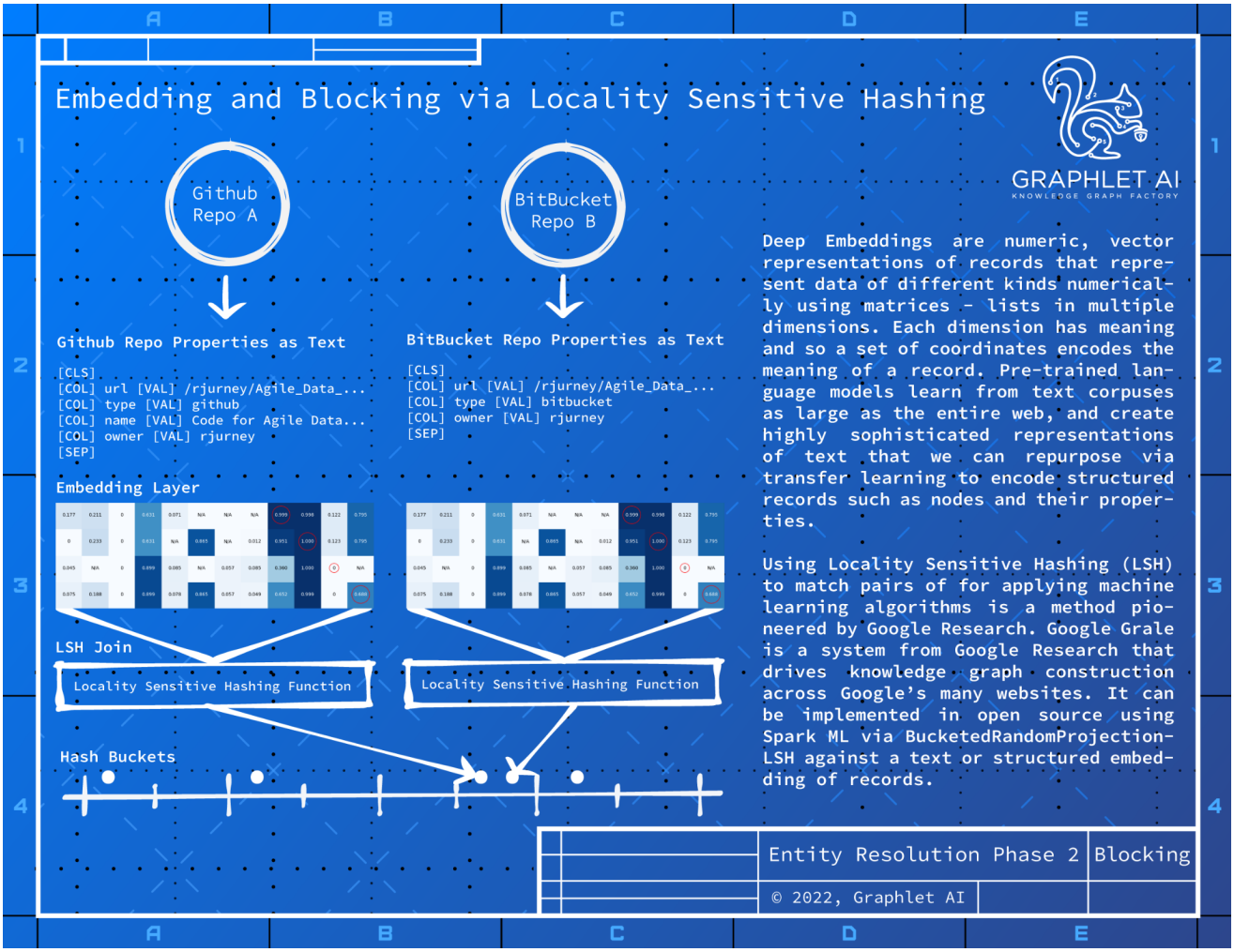
Google Research put out a paper in 2020 about a system called Grale - the system that drives Google's internal knowledge graph factory for all its web properties. Similar to the reduce phase of MapReduce, Grale uses a relatively simple, scalable algorithm called Locality Sensitive Hashing (LSH) to perform blocking of nodes into similar groups for comparing pairs of nodes to perform tasks such as blocking for entity resolution or link prediction. It can be thought of as MapLSH for the big graph ML space as compared to MapReduce for general purpose big data processing.

It is critical to ship the simplest thing that could possibly work to production before moving on to more sophisticated methods for any machine learning problem one pursues as premature optimization can be a terminal flight pattern for any ML project. A knowledge graph factory should support and enable this process through tools that enable stepwise implementation of solutions to link prediction problems.

A lightweight implementation of link prediction is possible for property graph nodes using the encoding mechanism for structured records from ditto, which can represent semi-structured data including lists, maps

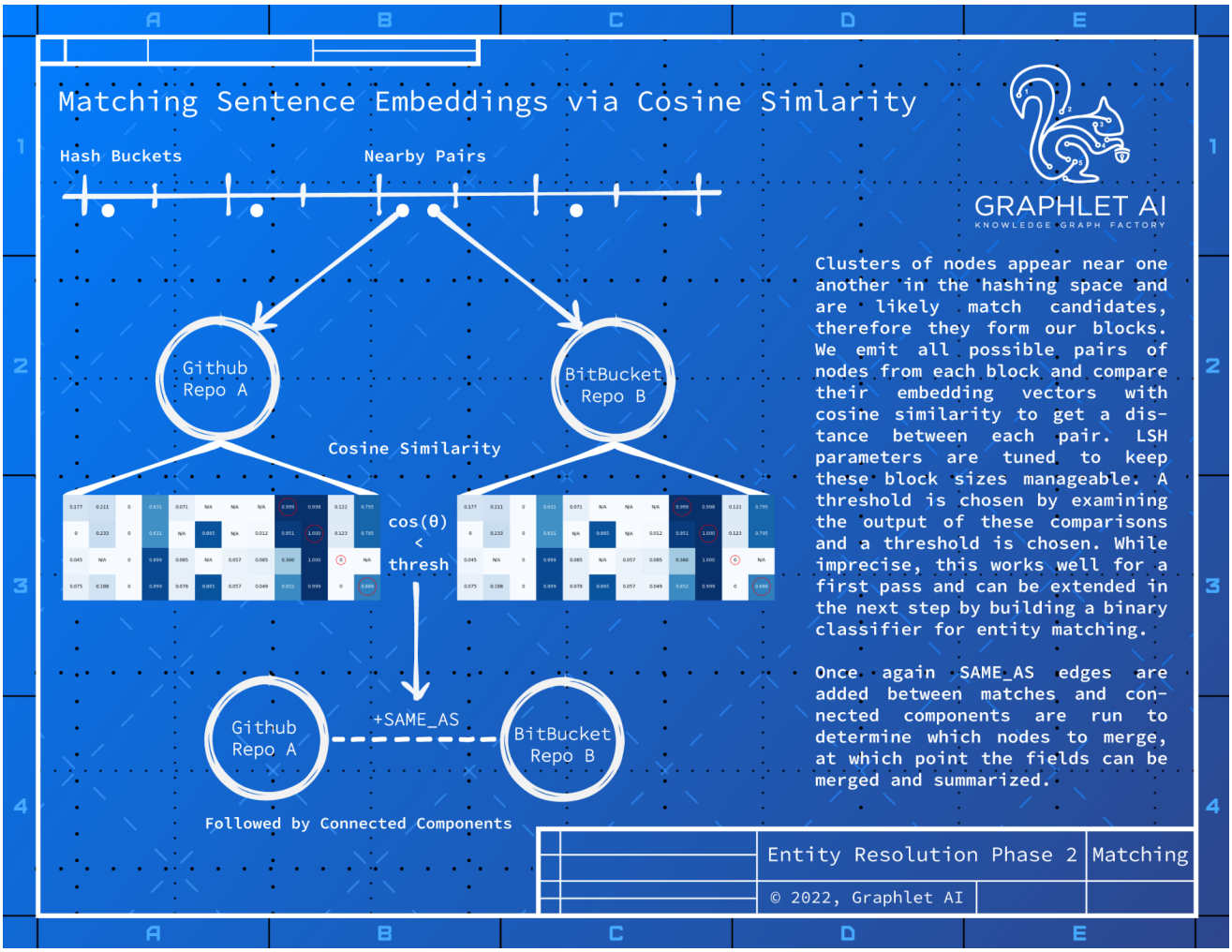
and trees. Sentence Transformers are used to encode the data. Spark ML's BucketedRandomProjectionLSH can act as the blocking mechanism for entity resolution. It offers a fuzzy join operator that can use any vector such as a deep embedding to perform self-joins for link prediction tasks.

Spark ML's LSH self-join on node records encoded using structural hints for pre-trained language models provide a base capability to create the edges in your enterprise knowledge graph that form part of the solution space via graph analytics whereas the provided edges usually lie in the problem space. *The job of a data science team using enterprise knowledge graphs is to transform a graph in the problem space to one describing the solution space.*



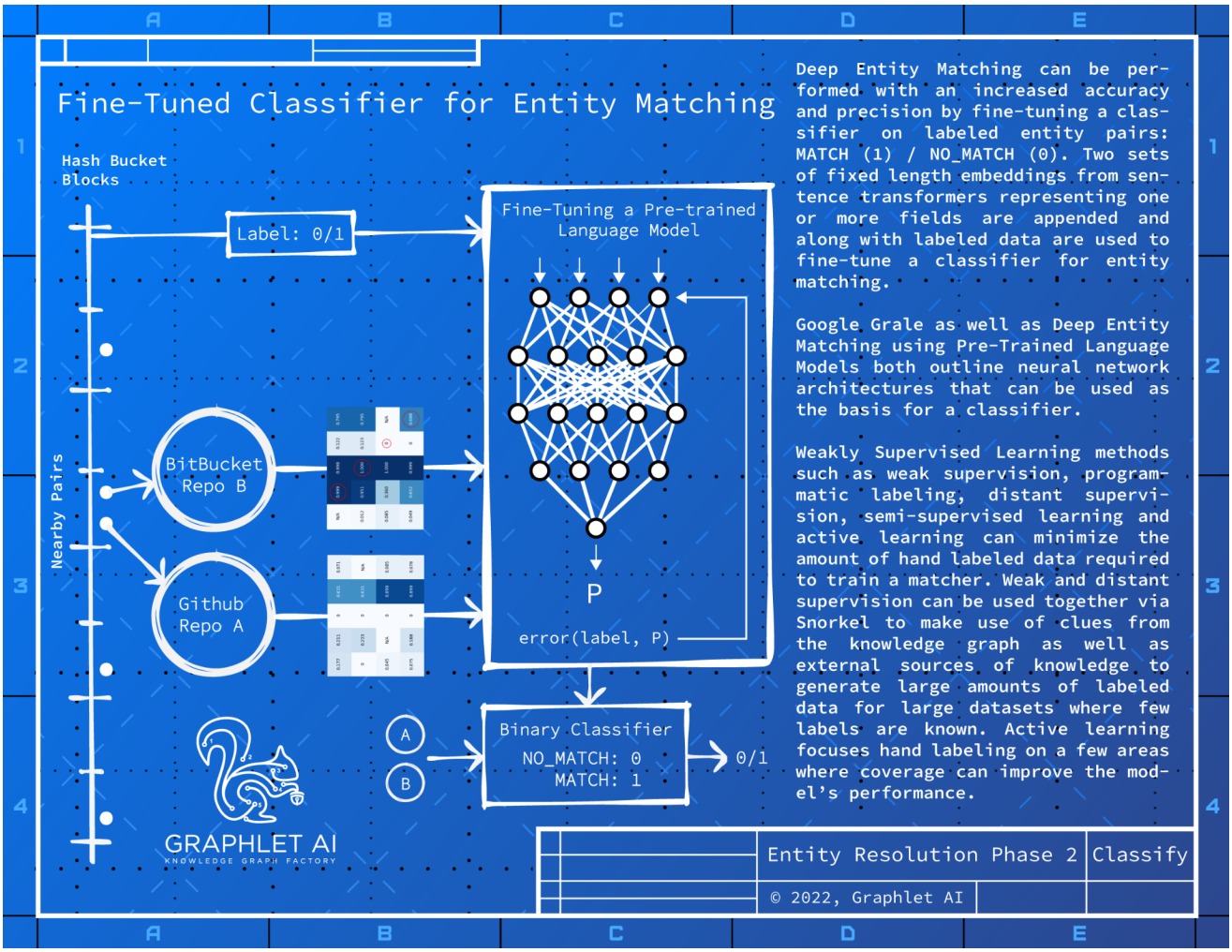
Entity Matching 1.0: Cosine Similarity Thresholds

An prototype for fuzzy matching of nodes in property graphs can be created using cosine similarity between the sentence embedding vectors and a hand-tuned threshold. This assures the practitioner of the technique's capabilities before investing more time and resources in a full implementation.



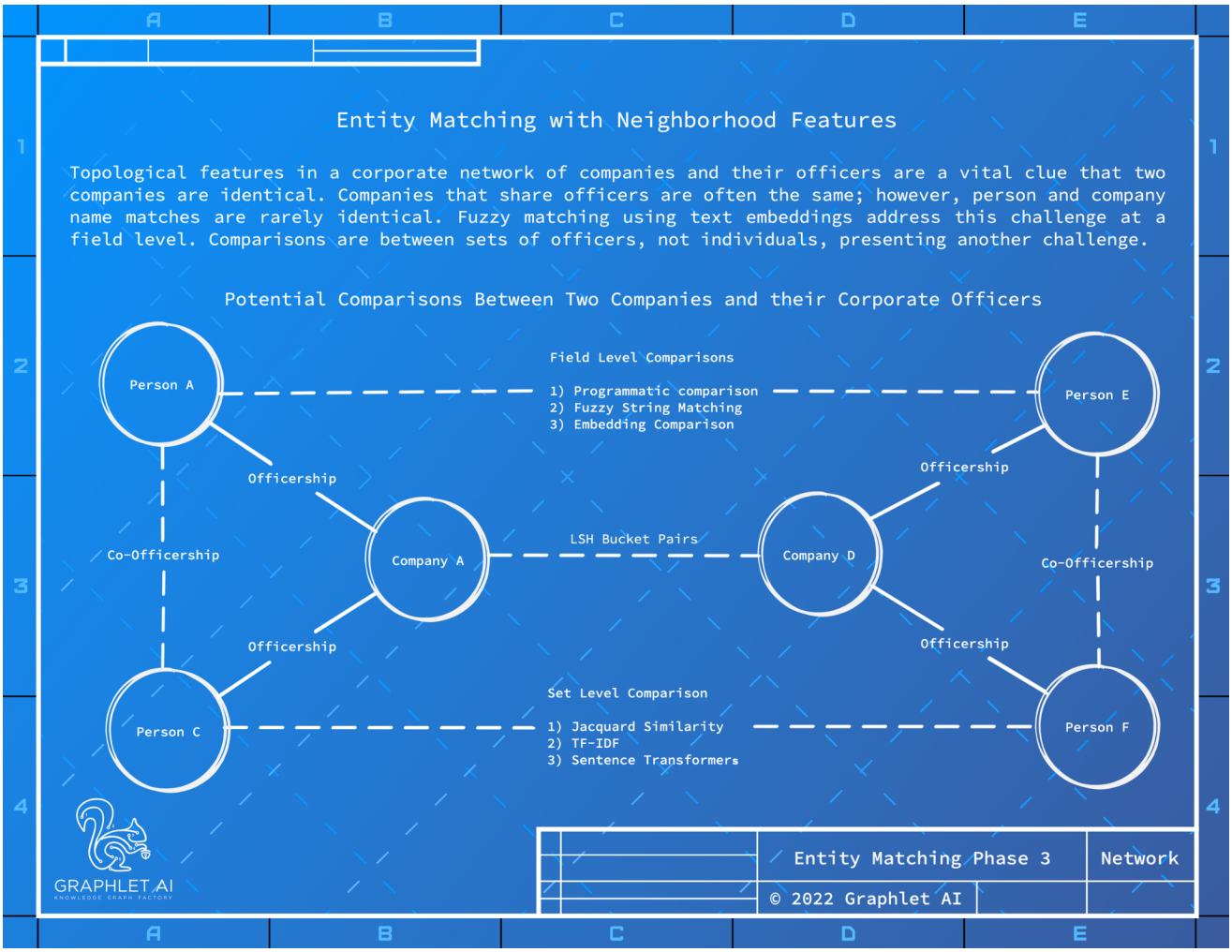
Entity Matching 2.0: Fine Tuned Classification

Labeled data for fine-tuning a classifier using the sentence transformer representations can be obtained via a labeling platform implementing active learning algorithms or by weak and distant supervision and programmatic data labeling (see: Snorkel).



Entity Matching 3.0: Hand Engineered Graph Features

The next phase of entity matching incorporates the topology surrounding a pair of nodes and incorporates them as features in the matching classifier. In the example below the fields of companies can be compared in the previous steps but in this phase the officer names and any fields describing them can be incorporated into the embedded representation via the scheme described above, for the example of entity matching between two companies below.



Because large language models have parsed a lot of XML, this scheme is sufficient to allow a sentence transformer or other text representation to infer the meaning of the semi-structured lists of officers and to consider them in context of other instances of descriptions of corporate officers.

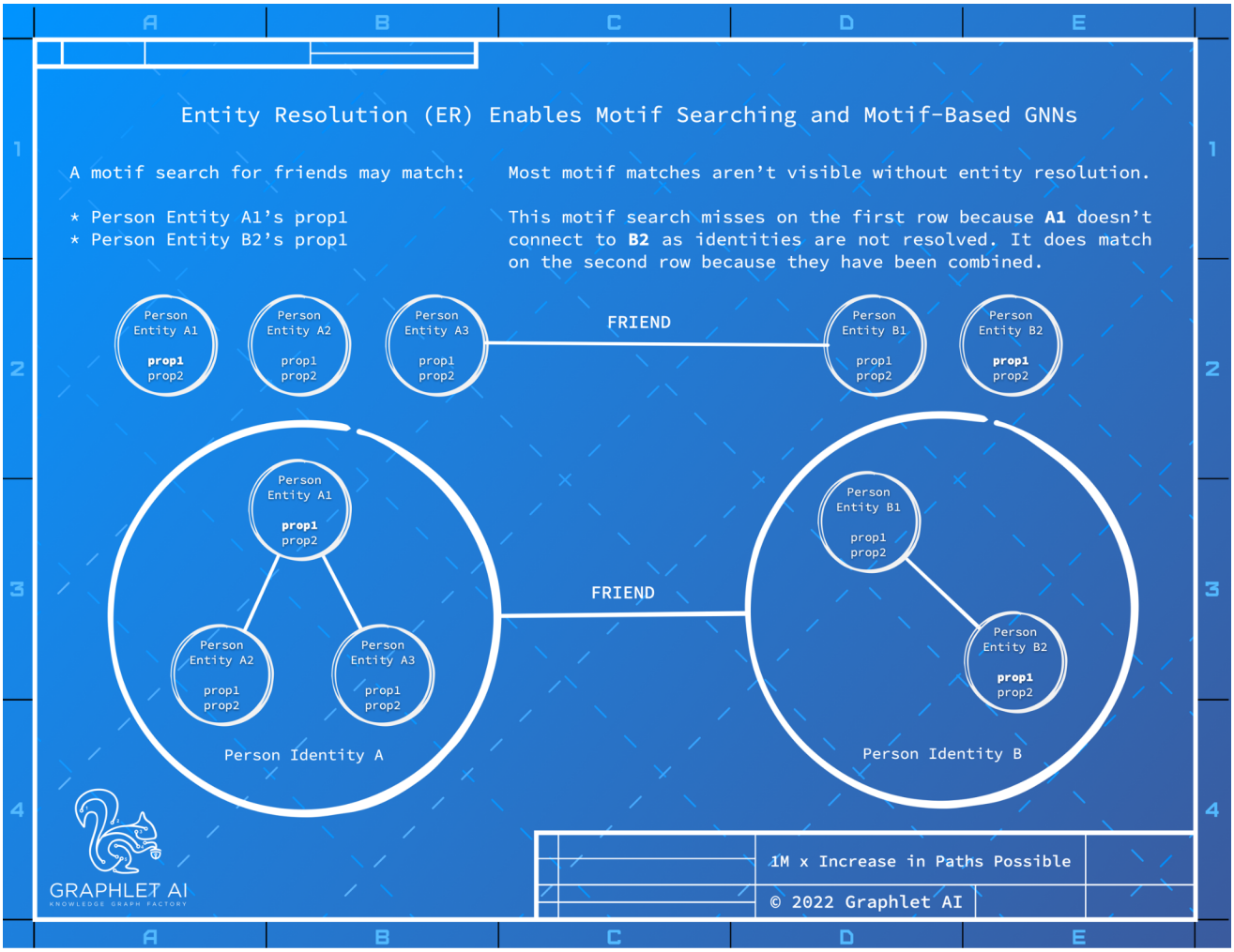
```
[CLS]
    [COL] type [VAL] company
    [COL] name [VAL] Apple, Inc.
    [COL] location [VAL] Sunnyvale, CA, USA
    [COL] officers.1 [VAL] Arthur D. Levinson
    [COL] officers.2 [VAL] Tim Cook
    ...
[SEP]
```

Entity Matching 4.0: Graph Neural Networks

In the highest level of sophistication for entity matching a GNN representation of nodes can be fine-tuned in a classifier to incorporate graph features beyond individual fields in a node's immediate neighbors.

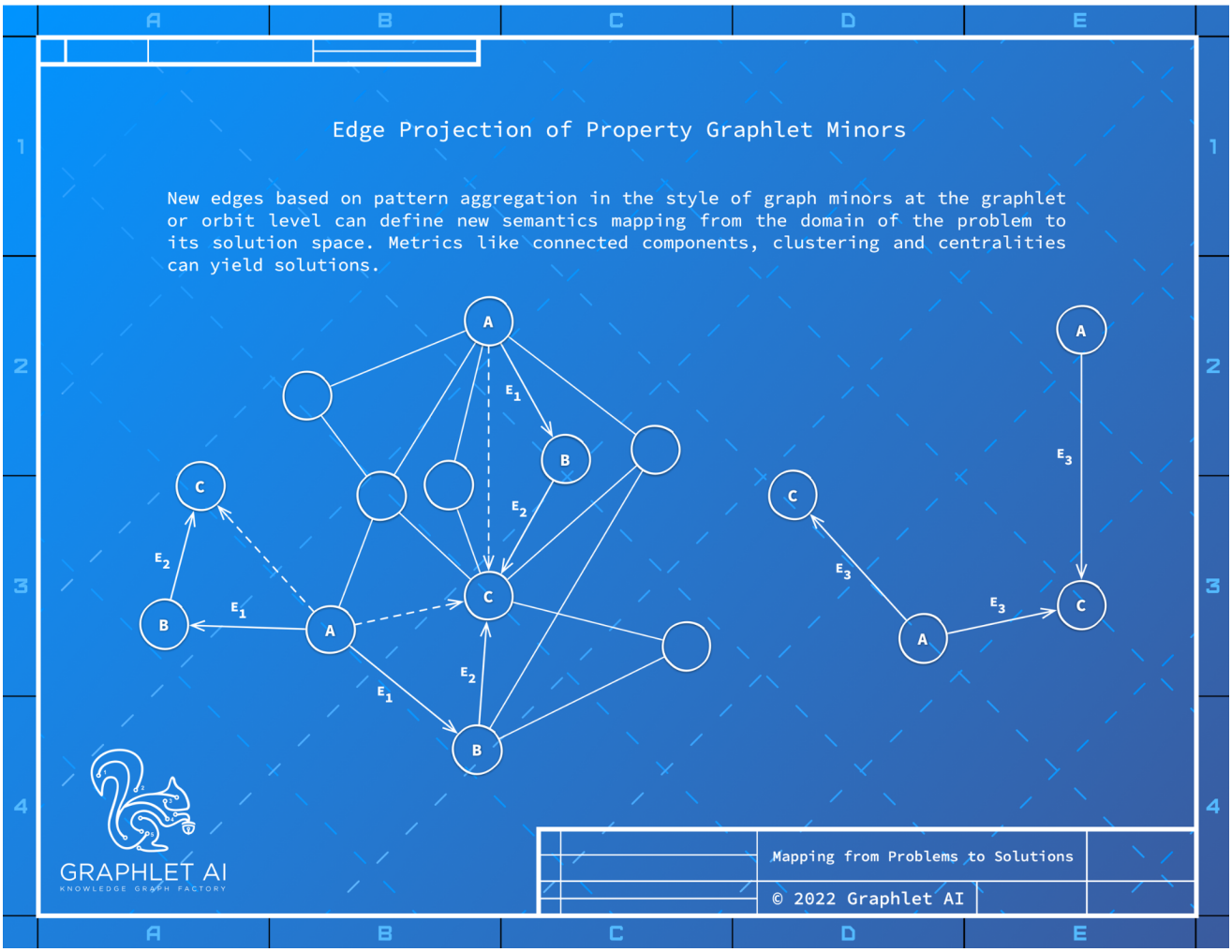
Heterogenous Graphlets and their Minors

Query languages for property graphs enable rich queries for network motifs incorporating both topology and node and edge properties. However, without first performing entity resolution, graph queries do not return good results because hopping across edges misfires due to duplicate nodes. This is especially the case on datasets and knowledge graphs that are themselves made up of disparate sources of nodes, edges and their properties because there is heavy bias in what results and how many are returned. *I have seen match counts for a motif search to be as much one million times more frequent after entity resolution is performed!*



Heterogenous graphlets are graphlets for property graphs that assign numeric structural roles to patterns in a network that allow the definition of a network defined not in the objects making up the problem domain (companies, people, officerships) but describing the concepts of the actual problem (competitive conflicts, financial secrecy or obscurity). Heterogenous graphlets are network motifs with numeric positions for structural roles.

Heterogeneous graphlets are then merged by removing edges into graphlet level or orbit level nodes in a process analogous to *graph minors* to create a concept level map of the problem space. This allows the problem to be modeled such that one of two approaches can be used to solve a given business problem: graph analytics or motif based graph representation learning, with the former being preferable as they are simpler than the latter.

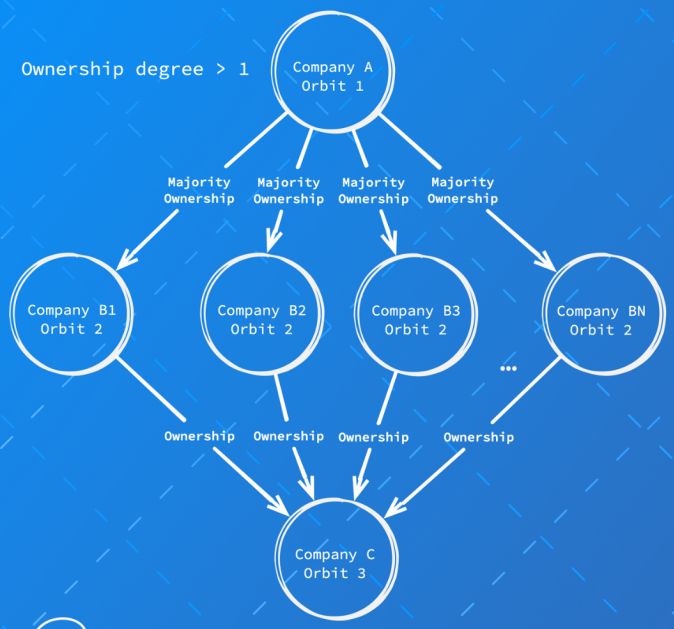


Graphlet and Motif Representation Learning

In addition to being useful for network analysis, heterogenous graphlets are used by systems like GL2Vec (code) as the basis for graph neural networks that perform representation learning by searching for the most significant graphlets as a way to bootstrap higher level representations and fine tune them for a given task. A simple approach to graphlet based learning is possible given that orbits representing structural roles in a network are easily represented as a bitwise position in a vector. These orbit vectors can be used as input for a graph neural network as they accept arbitrary input. A graph neural network can then learn deeper representations with the graphlet concepts available from the beginning of training.

In the image below you can see how a graphlet can be computed over a business graph using a Spark based library called GraphFrames that provides a powerful capability to search for network motifs which can then be labeled with orbits corresponding to structural roles to create a heterogenous graphlet that can then be used with the graphlet minors previously described or as a feature of a node or edge for a graph neural network.

Business Graph Motif for Financial Secrecy: *Multiple Path Indirect Ownership*



Sum of ownership percentages > 50%

Banks evaluate the risk of doing business with their customers by analyzing their assets and associations. A network science approach uses a business graph formed by companies and their officers from business registries and other public records.

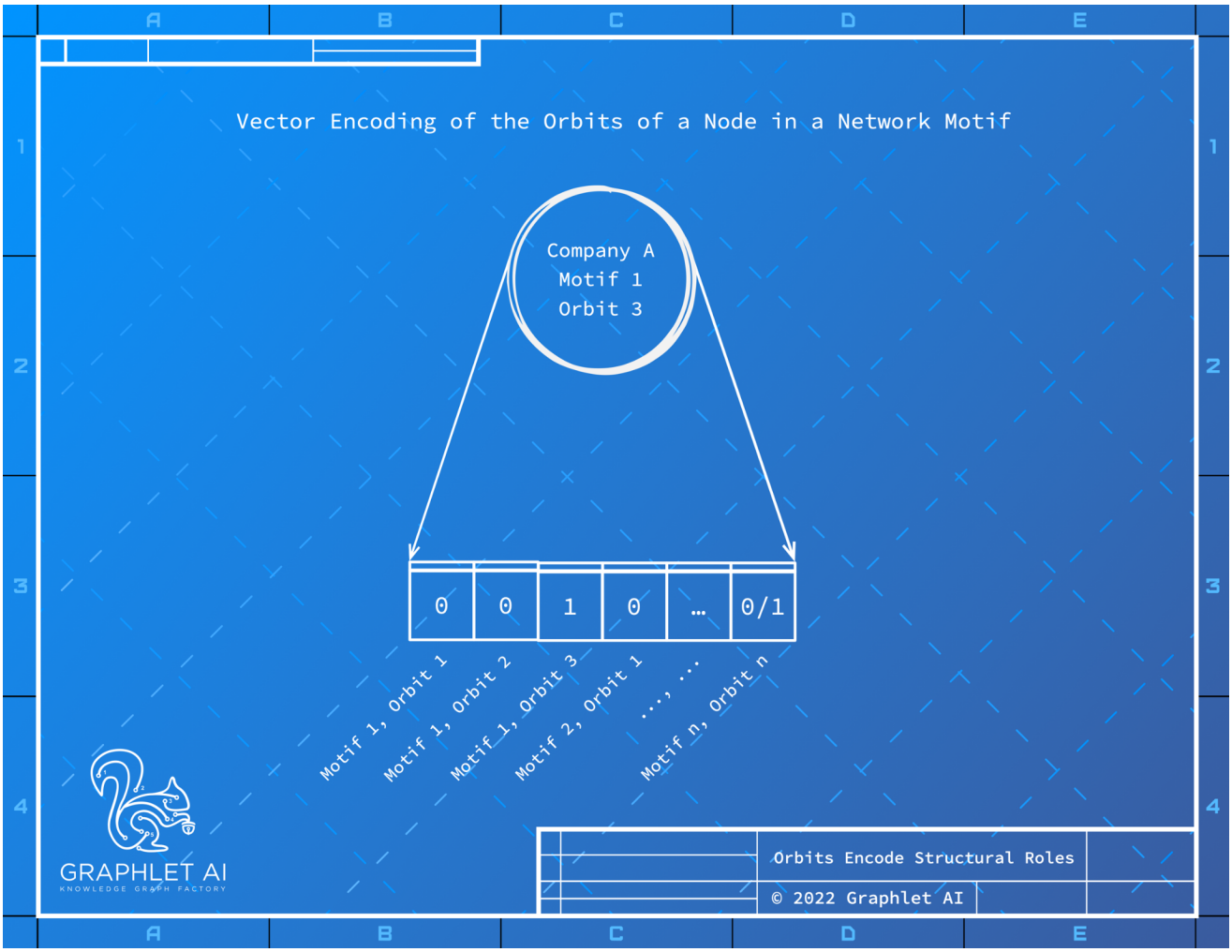
Multiple-path indirect ownership is a network motif for risk as it indicates an attempt at secrecy by obscuring a controlling interest in a company to avoid meeting financial disclosure requirements for banking.

(Company A) can obfuscate its controlling interest in (Company C) using an intermediate layer of shell (Companies B). (Company A) owns a majority share in multiple (Companies B), each of which owns a minority share in (Company C). In combination, these minority shares sum to a majority share of (Company C) for (Company A), which is in control of (Company C) as a result.

* Source: Creating clarity around ownership structures, Bureau Van Dijk

GRAPHLET AI
KNOWLEDGE GRAPH FACTORY

Risk Scoring via Network Motifs

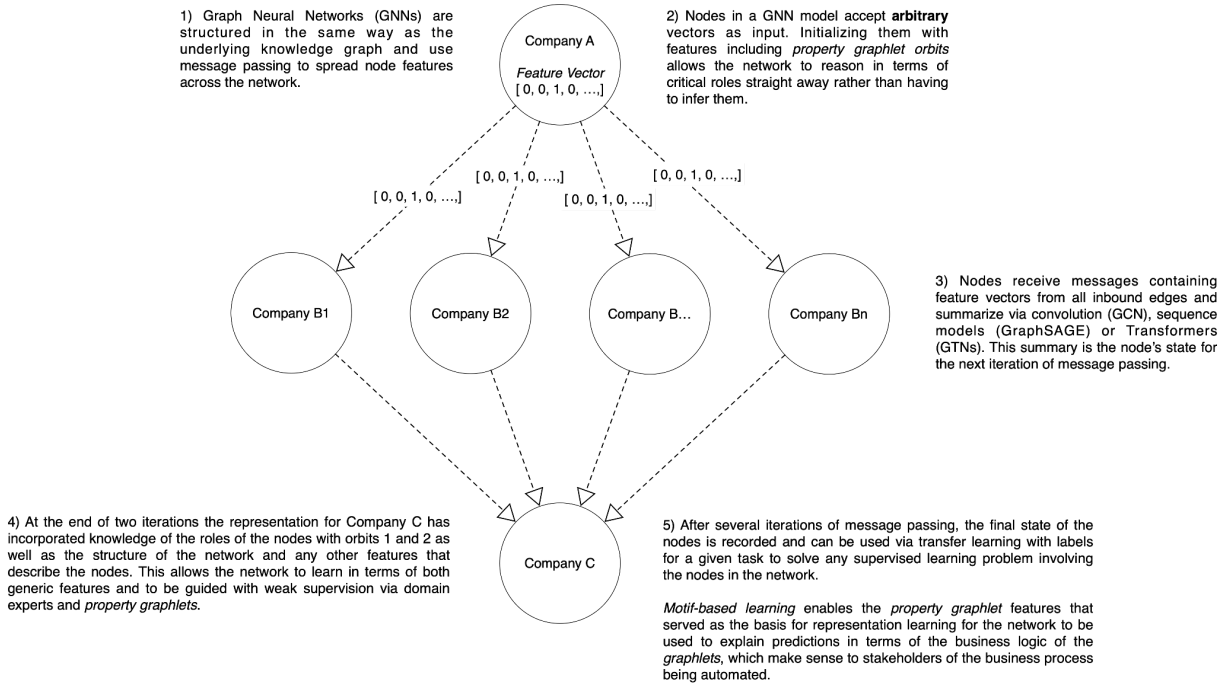


While most graphlet and motif based learning perform searches for the most significant patterns in a network, there is a combinatorial problem with searching for graphlets or motifs with more than 5 nodes, and the complexity of graphlet searches over property graphs are not yet well defined[1] but are even more complex once you factor in the use of properties.

[1] *A literature review by the author, Russell Jurney, turned up empty on the complexity for heterogenous graphlet search.*

Orbit Vectors as Features for Representation Learning

Graph Neural Network (GNN) - Orbit Message Passing



Complexity of graphlet search necessitates the incorporation of additional heterogenous graphlets defined through graph queries in GraphFrames created in collaboration with domain experts into the graphlet representation learning. Structural roles that relate to the solution to a downstream task will aid performance and can be used in explainability. Systems like MotifExplainer use network motifs to explain predictions, which can be visualized in the domain of the problem to explain weakly causal explanations to users of an application. This work to incorporate pre-defined graphlets with graphlet searches as part of a graph neural network architecture remains to be researched but is the key to understanding networks in an application where the network itself as well as the models predictions are of interest to the user.

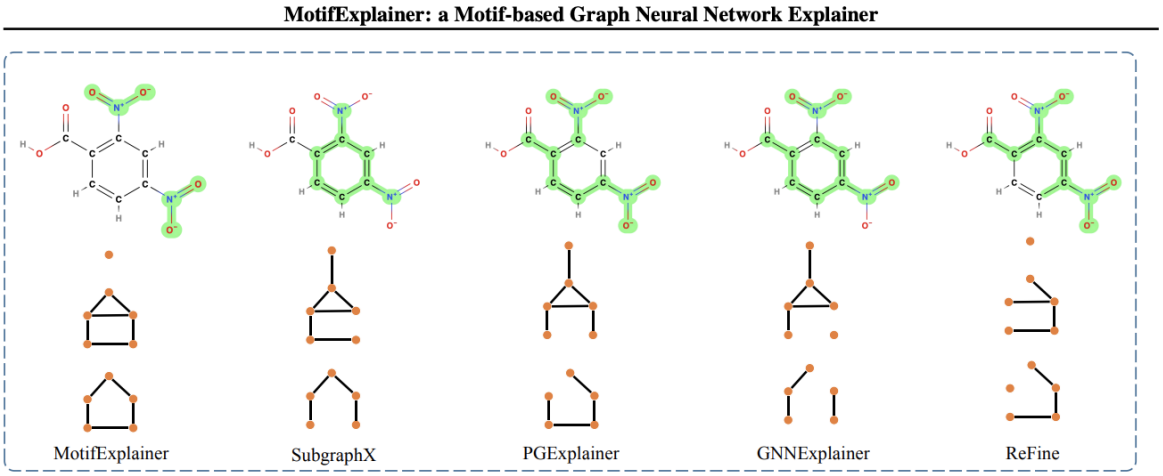


Figure 2. Visualization of explanation results from different explanation models on three datasets. The generated explanations are highlighted by green and bold edges. Three rows are results on the MUTAG dataset, the BA-Shape dataset, and the BA-2Motif dataset, respectively. We only show the motif-related edges for two synthetic datasets to save space.

MotifExplainer on the left provides explanations in terms of network motif visualizations

KNOWLEDGE GRAPH SERVICES

This section outlines how knowledge graphs, once constructed, find use inside an enterprise. Most enterprises have more than one reason to build a knowledge graph. *Node representations* from graph neural networks trained on knowledge graphs can find a range of internal use cases within a single organization by powering multiple models that automate tasks. A knowledge graph for this use case is a feature store. On the other hand many use cases for knowledge graphs within an enterprise involve serving a portion of the graph via an API based on a search query along with any inferences produced by a model.

Task Automation via Transfer Learning

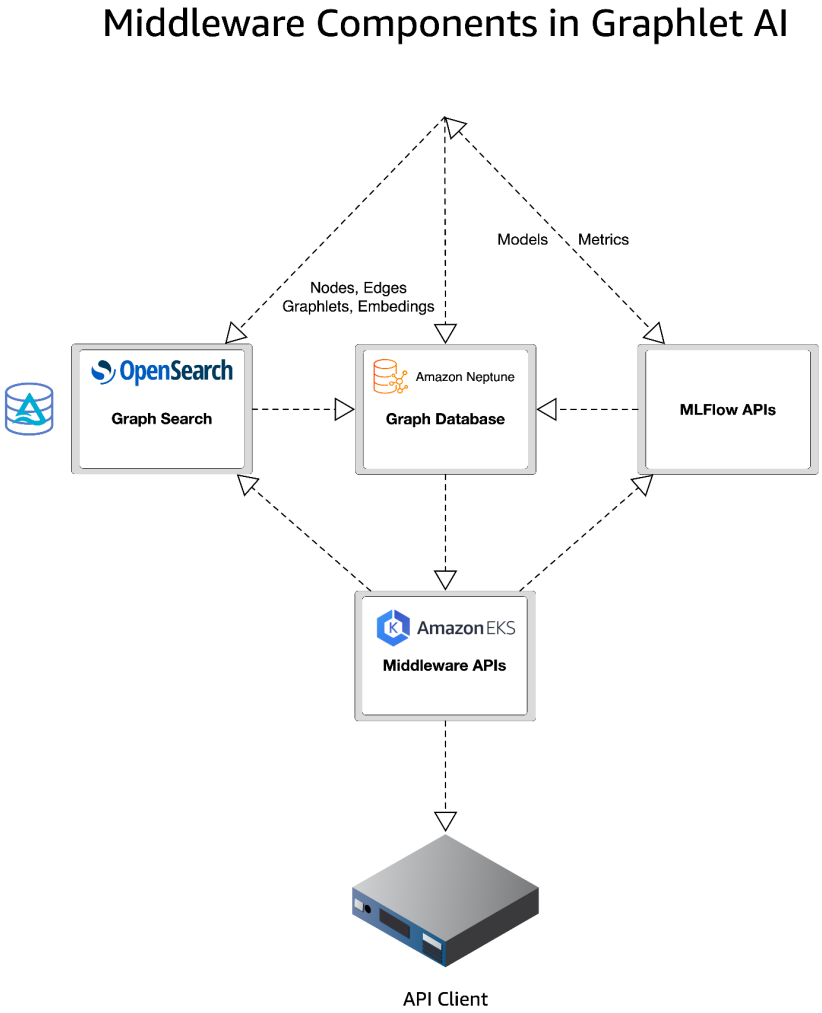
The novelty of graph machine learning in the form of graph neural networks is that the representations it produces are able to incorporate the context of a node's surroundings in a network in an embedding for each node in the network. These node embeddings are a general purpose asset to drive machine learning applications within a company. Once the problem domain of the business that the knowledge graph models is understood in the right representation, task automation using these representations is a much simpler matter. This is how knowledge graphs can power a company in its transformation to Software 2.0, where its core assets are the models that automate its business processes rather than the source code that powers its services.

Graph Search, Model Management and API Access

Many graph machine learning applications require real-time APIs to automate tasks using predictive models so the ultimate output of a property graph factory is to instantiate a cloud search engine and graph database and load the nodes and edges of the graph. Most graph databases tightly integrate with a search engine to provide real-time graph queries. Most graph queries in a graph database start with a search for one or more matching nodes and proceed to walk across the graph, defining a motif, which can be labeled with orbits to create a property graphlet.

Batch and Realtime Inference

Inference or prediction can often be pre-computed using batch computing systems like Spark in fitting with the ingest / build / refine / publish pattern used in most big data applications. Inferences create new edges that define semantics which enable simple queries or graph analytics that satisfy application requirements. Models powering these inferences can be applied to the graph in batch in advance of being indexed and queried. A model management system like Spark and MLFlow, which operates on top of DataBricks via its Managed MLFlow offering, can be employed to provide task automation using real-time APIs for machine learning. Amazon Neptune supports the openCypher query language and integrates with the Amazon OpenSearch Server to initialize or filter graph walks as part of queries based on the full lucene query language.



Once the ontology of the application graph is defined, the datasets making up the graph are transformed to match this ontology. Representation learning describes the network. These representations power models which add edges to the graph to create the type of relations needed to solve a given problem. These models are applied to the network using batch inference with MLFlow.

Additional systems are needed to serve the edges to the graph that define our solution in the context of their local neighborhoods: a search engine, graph database, version ML APIs and a middleware API server. The user of a knowledge graph factory should be able to point-click create the above diagram to serve that knowledge graph via an API. The queries can then be efficiently defined by the user and the system can move into production efficiently.

Vector Search

The representations of the nodes and edges in the network produced by training a graph neural network (GNN) are vectors that can be used via Approximate K-Nearest Neighbor (ANN or A-KNN) or HSNW to

perform semantic searches on an enterprise knowledge graph. Vector search is also a powerful tool for internationalization (i18n) using tools like multilingual sentence transformers. Scaled up queries using vectors are simple to add to both Elastic 8 now that Lucene 9 supports ANN and OpenSearch supports vector search as well. While there are numerous new vector search engines, Elastic with vector search support is an effective way to search billion node/edge knowledge graphs as it is easily partitioned across multiple machines and has excellent support through the Lucene community. Elastic makes it easier to load data and handle a query workload while scaling in a linear manner. Either version of Elastic can now use TF-IDF style search for matching or ANN and also rank and re-rank using vector matching. A property graph factory should include an API for vectorizing search requests through a model management system like MLFlow to incorporate them into queries against Elastic as it takes a query vector to search with an index vector. Tools like Steamlit and BentoML ensure most ML platforms will soon offer APIs for inference and associated information retrieval.

Explainability via Network Visualization

People do not act on predictions that they don't trust. Without explainability or causal modeling, even accurate predictions have limited utility in many domains. When working with knowledge graphs, network visualization is an effective way of visualizing explanations of predictions in a knowledge graph factory. In fact even without an explainable model, serving a prediction along with the subgraph of its neighborhood provides rich context that can satisfy explainability requirements.

A visual explanation of a money laundering risk score based on business networks

Systems like Graphistry can handle large scale network visualization but most explanations are in the local network of the node or nodes of interest. These are retrieved based on the search criteria the user of the application gives and which the search component of the factory sends the API. For smaller scale, interactive network visualization Cambridge Intelligence offers a system called Keylines that has a React wrapper called ReGraph. ReGraph allows you to define rich interfaces in a simple manner using skills common among application developers such that expertise in network visualization is not required to build interactive network visualizations.

A property graph factory should have an API layer for search and information retrieval along with a web interface for visualizing networks that together enable application developers using the factory to produce web and mobile apps that present and explain inferences and predictions in the business domain of the end user.

PROBLEMS AND SOLUTIONS

NOTE: THIS SECTION IS IN PROGRESS

The process defined above is executed by many organizations in one variation or another. It is never easy. The scale of the data involved combined with frequent combinatorial scaling issues occurring with graphs.

Large knowledge graphs arise from heterogeneous sources of data, each of which must be normalized into a common representation through an ETL process in order to be modeled and queried as part of a solution. *A series of common challenges are encountered by teams constructing knowledge graphs and using them with machine learning to power task automation.* ***An effective knowledge graph factory is a system designed to mitigate these problems and make teams more productive.***

Features of Graphlet AI

NOTE: THIS SECTION IS IN PROGRESS. It is pasted from a [too] long email :)

Features that Graphlet AI will incorporate include:

- Python base classes for speeding up PySpark ETL of different datasets into a uniform ontology that produce DataFrame schemas making them easy to use with pandas_udfs in PySpark. It would have saved us a ton of time if we had classes with properties and we just extended the methods and they were run in one LOC for each dataset we transformed from the raw datasets to our application graph. For example at the moment on a project I have gigabytes of GitHub, GitLab and BitBucket repository metadata and I want to transform it into a Repository entity in an open source ecosystem ontology. Makes it easy to define base classes to reuse code to transform. I am looking at using something like pydantic-spark which makes this pretty easy... just a matter of a good example in docs. It would have saved us time on a team that went from 1 to 16 engineers.

- GPU accelerated fuzzy LSH joins in Spark. Spark's included implementation of LSH joins is CPU and was a major bottleneck for us using 384 dimension sentence transformers to do blocking for entity resolution on Databricks... reducing the size of the vector involved was the way to optimize it on Spark but we ended up using distributed FAISS instead. For a single stop KG platform you want it to "just work" on your cluster to get the job done. The Google Grale paper takes note of the widespread use of LSH on embeddings to implement a sort of "MapLSH" for graph ML which is as fundamental for scaling graph ML as MapReduce is for scaling general data processing. Whatever the domain of your graph, you are missing many of the edges for the relations you have in your problem domain and you don't have the edges you need that define solutions in that space... you need to pair nodes to use ML to build the edges that enable simple graph analytics to build and deploy automation solutions.

- A configurable entity resolution system for large knowledge graphs using deep entity matching for pre-trained language models (Github)

- Tools to go from PySpark (GraphFrames) or pandas node/edge DataFrames to a deployed API driven by Amazon Neptune and OpenSearch... could be Neo4j and Elastic. The point is once you have a graph, you probably need to serve the actual graph with the inferences you're making (sometimes not, but I have always needed to in the problems I've worked on) and this should be a

point-click solution. I've thought of making a product on top of Databricks that does this on AWS.

- Support for writing motifs in GraphFrames and shipping them as graphlet features to the aforementioned API so you can visualize the structural roles involved in a viz as a form of explanation. You can also use them in your representation learning.

- Summarization of nodes and edges with embeddings that then ship in the form of vectors in OpenSearch / Elastic so you can use them for search. It would be great if you could go from a GraphFrame and do what PyGraphistry is doing with DGL to get an embedding from a GNN in a few lines of code. The scale part is what is hard.