KGStorage

A Benchmark for Knowledge Graph Generation at Scale

Brendon Wong and Tracy Tuvera
Unize Research

September 26th, 2024

Table of Contents

Summary

Unize is excited to announce KGStorage, our initial benchmark for evaluating AI-generated knowledge graphs. We benchmarked LangChain's LLMGraphTransformer module, one of the leading LLM-based graph generation systems which was created by Neo4j, against Unize ST-0.5, our new system for generating knowledge graphs at scale.

On short, single-document inputs, we find that Unize ST-0.5 and LangChain perform comparably, with Unize ST-0.5 achieving slightly higher accuracy rates. For example, Unize ST-0.5's achieved a node accuracy score of 95.8% compared to LangChain's 89.2%. On longer, multi-document inputs, we find that Unize ST-0.5 performs significantly better, with 86.8% node accuracy compared to LangChain's 57.0%. LangChain generates fragmented graphs, whereas Unize ST-0.5 generates well-connected graphs. When used with their default information extraction goals, both systems have information capture rates that are generally within 10% of each other. Unize ST-0.5 is considerably more expensive than LangChain, with a cost more than 10x higher when run on longer inputs.

We hope to add additional datasets and benchmarked systems, as well as automated evaluation, in the coming months. Please reach out if you have any questions or would like to collaborate!

Background

Unize is a social impact startup helping individuals and collectives harness structured knowledge and thought to better achieve their goals. We're excited to announce our initial effort at creating a benchmark to evaluate knowledge graphs generated by AI systems ("**knowledge graph generation**" or "**KGG**" for short), which we're calling KGStorage.

Existing KGG benchmarks such as Text2kgbench and the WebNLG 2020 dataset focus on evaluating graph generation with inputs up to several sentences long. We developed Unize ST-0.5, the first release of Unize Storage, an AI system with the objective of generating coherent knowledge graphs from text at any scale. This includes supporting any number of input documents, any length of input documents, and any size of destination graphs. As such, we felt the need to create a benchmark to reflect the capabilities of emerging systems that can handle much larger inputs.

KGStorage grades generated graphs on accuracy, connectivity, and capture. Accuracy and connectivity are evaluated relative to the graph itself, but capture is how much information the generated graph captures from the text evaluated relative to an evaluation graph. Constructing evaluation graphs is an interesting area of work that pertains to standards for how people and AI create knowledge graphs ("**knowledge graph construction**" or "**KGC**" for short). Today, many knowledge graphs are generated relative to small, predefined schemas, but we believe that large-scale KGG systems will need to dynamically generate schemas to automate graph construction at scale. As such, we did a cursory overview of the KGC space and decided to employ two KGC methodologies for evaluation graphs that are not dependent on predefined schemas: named entity graphs that contain all proper nouns in the input, and all entity graphs that contain all real or conceptual entities in the input.

Currently, KGStorage requires manual evaluation of accuracy and capture, but AI systems should be able to employ our evaluation criteria for evaluation, so we hope to introduce automated evaluation in the future.

Dataset Inputs

Due to time constraints in our initial benchmarking work, we decided to evaluate graph performance with two datasets: (1) a short ~5,000 character article, and (2) the short article combined with a longer ~15,000 word article about a highly related topic.

This first dataset was chosen because in systems such as LangChain, content can be ingested in several chunks, or even fed into an LLM in a single request. This makes processing this dataset a good task to evaluate baseline performance. This task is representative of how users might generate standalone graphs from relatively short, singular documents.

The second dataset represents a first foray into multi-document performance. We anticipate multi-document performance being very important for use cases like processing internal documents to aggregate information across an organization, or processing external documents to consolidate information about a certain topic. This task should be significantly harder for KGG systems.

There is much room here to expand the difficulty of the datasets. In particular, assessing one or more dramatically longer documents (like a textbook), or simply using a larger set of input documents (and varying the degree of document similarity within and across datasets). Due to our constraint of manual dataset creation and evaluation, we were unable to go beyond two documents in our initial benchmark attempt, but the multi-input dataset already gives an indication of how KGG performance changes as scale increases.

Single-Input Dataset

The article we used for the single-input dataset comparison is a movie review of “Avengers: Endgame.” The article is 5,364 characters and 922 words long, which is approximately 1.5 single-spaced pages.[1]

The article contains a mix of commentary and factual information, as well as a mix of real and conceptual entities, which is a fairly diverse set of elements to evaluate extraction against. It also

[1] Using this conversion table: https://capitalizemytitle.com/page-count/

contains content that many readers may be familiar with, which makes it easier to qualitatively assess graphs generated from the article.

Multi-Input Dataset

The article we selected for the multi-input dataset to be processed and merged into the "Avengers: Endgame" knowledge graph is the movie review for "Avengers: Infinity War." The article is 14,474 characters and 2,395 words long, which is approximately 4.5 single-spaced pages long.

This article was chosen because it contains elements that fall under many relevant categories of similarity within itself and in comparison to the first article, including: (1) identical, (2) identical but differently named (e.g. Wanda Maximoff vs. Scarlet Witch), (3) non-identical but identically named (e.g. Avengers the movie series vs. Avengers the superhero team), (4) non-identical but similarly named (e.g. MCU vs. MCU fans), and (5) completely non-identical.

These areas of similarity make processing this article a fairly challenging task for KGG systems.

Construction of Dataset Evaluation Graphs

In our cursory review of the KGC space, we found many approaches to entity extraction,[2] and graph structure can vary based on end user needs. While we believe there may be avenues to create more universal knowledge graphs, that's a separate research topic altogether.

We decided to use a dual approach to evaluate information captured by KGG systems: (1) capturing all named entities in the dataset and (2) capturing all entities in the dataset. The first construction approach results in a streamlined graph that's easy to construct and evaluate, but may lack certain details found in the text. The second construction approach results in a highly detailed graph that represents how well a system is capturing many aspects of the text, but results in a very large graph that may contain unnecessary details, and is potentially more arbitrary to create.

Please see here to view the underlying data for the constructed Named Entity and All Entity graphs.

Named Entity Graphs

We had a team member manually create named entity graphs, which took 2–3 hours with the single-input dataset, and 4–5 hours with the multi-input dataset.

[2] For instance, as described in this paper, do you represent "cancer patient" as one or two entities?

In order to be included, an entity had to be a proper noun, or have a proper noun associated with it.[3] While this criteria was relatively straightforward, there were some complex cases, like the proper noun “Zack Snyder” in the noun phrase: “Zack Snyder-style, heavy-metal gloom.”

All Entity Graphs

We experimented with a hybrid graph construction approach for our all entity graphs. We used OpenAi’s GPT-4o model to transfer text to Cypher statements, and had a team member correct the generated statements. While this method was faster than manual construction, it may or may not result in more errors, and the resulting graphs could be more similar to AI-generated graphs than human-generated graphs.

We made the following corrections to AI-generated Cypher statements:

- Removed nodes that were not part of the text.
- Deduplicated nodes by merging nodes that looked similar.
- Merged separate nodes that should be represented as a single node.
- Removed the relationships of nodes we removed, and deduplicated the relationships of nodes we merged.
- Removed relationships that could not be traced back to the original text.
- Added relationships that were missing between nodes.

We noticed that using AI-generated graphs resulted in a large number of errors; thus, it took 3–5 hours in total to correct the single-input and multi-input graphs.

Systems Evaluated and Usage Instructions

In this section, we cover the different KGG systems that we evaluated, the exact steps we followed to use the system to generate graphs, and how we prepared the generated graphs for evaluation.

LangChain LLMGraphTransformer

We chose to evaluate LangChain’s LLMGraphTransformer module because, to our understanding, it is the most prominent KGG system. LangChain is one of the top LLM frameworks. Neo4j, the leading graph database company, contributed the LLMGraphTransformer module to LangChain. They have an in-house KGG team. We evaluated the August 2024 version of LLMGraphTransformer.

[3] Possessive nouns and other entities with associated proper nouns account for <10% the total number of nodes, so they don’t impact results very much.

LangChain is easily accessible via Neo4j’s llm-graph-builder app. To use it, you have to connect to an AuraDB instance (you need to have an AuraDB account) which can easily be done by clicking the “Connect to Neo4j” button found on the app.

It should be noted that in Neo4j’s llm-graph-builder app, you can either upload a document, an image, select a Wikipedia article or a website, among other things. We opted to upload local text files instead to make the unstructured text input consistent across the systems we compared. The file name does not matter as we are only going to evaluate the nodes and relationships generated from the actual content of the movie review articles.

First, we uploaded the text file *endgame.txt* which contains the actual content of the "Avengers: Endgame" movie review, and chose GPT-4o as the LLM model.

To view the graph, click the “Explore Graph with Bloom” button, which redirects users to the AuraDB instance that houses the graph. To ensure that we were only dealing with content-related nodes and relationships, we filtered the nodes and relationships shown with the following Cypher statements:

```
MATCH (n)-[r]-(m)
WHERE NOT n:Chunk AND NOT n:Document AND NOT n:Message AND NOT
n:Session
RETURN n, r
```

The Cypher statements above removed any nodes that were labeled as “Chunk,” “Document,” “Message,” or “Session” (all of these labels were used by LangChain for nodes related to the source file).

Before generating and merging the “Avengers: Infinity War” article into the “Avengers: Endgame” graph, we had to evaluate the performance of LangChain on the single-input dataset first. This is because the moment that users upload and process a new article, LLMGraphBuilder will merge the article into the existing graph. This would make it nearly impossible for us to evaluate the non-merged graph as we would have little to no idea which nodes and relationships were for the single-input dataset. The evaluation process is covered separately in the Evaluation Methodology section.

Finally, we uploaded the text file *infinity war.txt* which contains the actual content of the “Avengers: Infinity War” movie review, and once again chose GPT-4o as the LLM model.

Unize ST-0.5

Unize ST-0.5 is easily accessible via the Unize API Playground. To use it, you have to sign up for a Unize API account, which can easily be done via the site.

To generate the single-input graph, we headed over to the Playground page and clicked the “Generate Graph” button. Clicking the button would bring out a right sidebar wherein we can add all information about the data that we want to generate a graph for. In our case, we copied the content of the *endgame.txt* file and pasted it on the appropriate box found in the Unize AI API site and filled out the rest of the boxes *except* for the Goal.

Just as mentioned in the preceding section, we had to evaluate Unize ST-0.5 single-input graph before processing the multi-input graph as doing so would make it harder to review the single graph.

For the multi-input graph, we selected the execution_id of the single-input graph and clicked the “Add Document” button to bring out the same right pane wherein we can enter the details and data for “Avengers: Infinity War.” We then proceeded to copy the content of the *infinity war.txt* file and filled out the rest of the fields except for Goal.

Raw LLM Output (GPT-4o)

We evaluated directly calling an LLM—in this case, GPT-4o—to provide a useful reference point for KGG systems. LLMs can only access the part of the input data fed into their prompts, and performance rapidly degrades as more of the their prompt context windows are used, so LLMs are expected to have poor accuracy across any input that begins to exceed the low thousands of characters in length. For this reason, raw LLM output results are presented in the appendix.

Comparing LLM-based systems like LangChain against raw LLM output on large datasets is a great way to assess KGG at scale.

To generate the raw GPT-4o graph, we chunked the entire *endgame.txt* content into chunks of <= 1000 characters. Then via the OpenAI playground, we sent requests to GPT-4o to transform the chunks into knowledge graphs using this prompt:

> **System**: “You are a Neo4j Cypher Query expert. Your task is to transform the entire text to a knowledge graph using Cypher Queries.”
> **User**: [insert_chunk_here]

We set temperature to 1 and max_tokens to the maximum of 4095.

We then reviewed each generated Cypher statements and made the following changes:

- Deduplicate identical nodes, variables, and relationships.
- Remove any relationships that contain variables that reference non-existent nodes.
- Remove semicolons after every Cypher statement, as it affects how the graph is created.

The Cypher statements were revised so that when saved to an AuraDB instance, for visualization purposes, we would not get empty nodes and duplicated nodes/relationships caused by incorrect formatting.

Other Systems

Here is a list of KGG systems we are aware of that we have not benchmarked yet.

Systems	Assessment
R2R	Pending benchmark, low-medium priority. Interesting looking system, especially the enrichment part, but does not appear to perform necessary post-processing, so results may be similar to raw LLM output.
LlamaIndex	Pending benchmark, low priority. Contains multiple knowledge graph constructors, but all constructors appear to simply call an LLM and perform no additional processing, so results are expected to be the same as raw LLM output.
WhyHow	Cannot currently benchmark. In “closed beta” and not publicly available.
Lettria	Pending benchmark, very low priority. Initial tests produce incomprehensible results. Appears to use pre-LLM technologies.

Graph Evaluation Methodology

In this section we explain in detail how we evaluated the graphs generated by Unize, LangChain, and GPT-4o.

All generated and evaluation graphs are using Neo4j-formatted labeled property graphs, which is why we are evaluating “nodes,” “relationships,” “node labels,” “relationship types,” and “properties.” This methodology can also be used in evaluating other graph types with modifications like ignoring “properties” if the graph does not have them.

KGStorage has three major measures of KGG performance: accuracy, connectivity, and capture. Accuracy is the correctness of a generated graph, which is a measure that is relative to the graph itself, but checked against the raw text in the dataset. Connectivity is how well-connected the generated graph is, and can be automatically calculated. Capture is the percentage of elements in an evaluation graph that a generated graph also has within it. In theory, an accuracy-like metric could be automatically generated by marking all elements in a generated graph that are not part of the capture evaluation graph incorrect. However, both systems were not assigned an extraction goal that aligned with either of the construction methods for our evaluation graphs, so we omitted that metric for now.

Accuracy is roughly equivalent to precision and capture is roughly equivalent to recall in IR and ML terms. We use the terms accuracy and capture because we believe they are more intuitive for external audiences, and also because our measures deviate from how precision and recall are traditionally calculated. Our current benchmark involves evaluating a generative, non-deterministic task, so the set of generated graph elements is unbounded. If a KGG system generates a relationship, while that relationship may not be in our evaluation graph, it may still provide value for users. One issue this causes is that the numerators for precision and recall would be different, since the former would be correct elements, and the latter would be captured elements. We will reevaluate this in the future, especially if we make our KGC standards more deterministic.

With current-day KGG systems, just like precision and recall, accuracy and capture are generally inversely correlated since capturing more information could result in more errors, like duplicating an existing element. It's important to take both measures into account, since as an extreme example, a generated graph with only one relationship could have a 100% accuracy rate, but would also have a near-0% capture rate, and thus be essentially useless.

Connectivity is an interesting metric since it can represent the quality of generated graphs even without any human or AI evaluation. Both a lower rate of information capture (relationships) and lower accuracy rate (duplicated nodes) will lower connectivity scores. However, this metric can be gamed by generating large numbers of incorrect relationships, so accuracy is still important to keep in mind.

Evaluating The Accuracy Rate of Generated Graphs

Nodes

According to Neo4j's documentation, a node is used to represent an entity. Entities are separate and distinct objects within a particular domain (field of knowledge). Nodes can have a label (defined below) that denotes which type of entity a node is.

Correct nodes are the number of nodes that are accurately generated by the KGG system relative to the total count of nodes generated. To determine the number of nodes that are correct, we determine which nodes are *incorrect* and categorize them by error type. After identifying all of the incorrect nodes, we subtract the total count of incorrect nodes from the total number of generated nodes to get the count of correct nodes.

Evaluating the accuracy of generated nodes requires subjective reasoning to decide whether a node is correct, or whether it falls under our four node error classifications.

Node Error Classifications	
Error Type	**Description**
Wrongly Named	These are nodes that have been named incorrectly. As an example we frequently saw scenarios that include two names lumped together in one node: “John and Marta” as a Person node vs “John” and “Marta” as Person nodes.
	These also include nodes that do not appear in the original text at all.
Mislabeled Nodes	These are nodes that have been labeled incorrectly. When doing the check, one must remember the context and check the properties of the nodes. For example, a node called “Apple” with a property “taste” should not be labeled “Electronics”.
Duplicate Nodes	These are nodes that bear the exact same or closely similar names and labels. For example: the Movie node “Harry Potter: Deathly Hallows” and the Film node “Deathly Hallows”, despite having different names and node labels, are considered duplicates of each other. Thus, we will count one of them as an incorrect node. If a node is a duplicate of another node but also falls under mislabeled or wrongly named, only count them under either of those categories.
Extraneous Nodes	These are nodes that exist in the text but are not relevant enough to be on the graph as nodes or nodes that should have been properties or relationships instead.

Relationships

A relationship describes a connection between a source node and a target node. In Neo4j, all relationships are all unidirectional. Relationships must have a “type” (defined below) to identify the meaning/classification of the relationship.

Correct relationships are the number of relationships that are accurately generated by the KGG system relative to the total number of relationships generated. To determine the number of relationships that are correct, we determine which relationships are *incorrect* and categorize them according to error type. After identifying the incorrect relationships, we subtract the total count of incorrect relationships from the total number of generated relationships to get the count of correct relationships.

Evaluating the accuracy of generated relationships requires subjective reasoning to determine whether the relationship is correct, or whether it falls under our three error classifications.

Relationship Error Classifications	
Error Type	**Description**
Wrong Relationships	These are relationships that use relationship types that are far different from the actual relationship between the two nodes. For example, the relationship in the text is (john)-[:IS_DATING]->(mary) but the relationship extracted by the system is (john)-[:ATTENDED_A_DATE]->(mary).
	These also include relationships that reference the wrong start and/or end node. If the node is wrongly named (labeled as an incorrect node), we will consider the relationship incorrect as well.
Duplicate Relationships	These are relationships that connect the exact same or essentially similar nodes to each other using relationship types that are exactly the same or essentially similar. For example: (john)-[:IS_DATING]->(mary) and (john)-[:IS_GOING_OUT_WITH]->(mary) are duplicated relationships thus we will count one of them as an incorrect relationship.
Extraneous Relationships	Relationships that lack details. For example, the text mentions John taking Mary on a date. The relationship should have been (john)-[:TAKES_ON_A_DATE]->(mary) but the relationship found in the system is (john)-[:TAKES]->(mary).
	Inferred relationships that cannot be proven right based on the text. For example, the inferred relationship of (john)-[:LOVES]->(mary) from the relationship (john)-[:IS_DATING]->(mary).

Node Labels

Node labels classify nodes into groups. All nodes with a certain label belong to that group. For example, all nodes labeled as “Person” belong to one group; this makes it easier to perform tasks like counting the total number of people in a domain, or retrieving data only from person nodes. A node can have zero to many labels.

Correct node labels are the number of node labels that are accurately generated by the KGG system relative to the total number of node labels generated. Just like with nodes and relationships, we first determine what node labels are incorrect, count the total, and subtract the total incorrect from the total number of node labels generated to get the count of correct node labels.

Evaluating the accuracy of generated node labels requires subjective reasoning to determine whether the node label is correct, or whether it falls under our two node label error classifications.

Node Labels Error Classifications	
Error Type	**Description**
Wrong Node Labels	These are node labels that are wrong in nature. An example would be labels that use the node's name like "InfinityStone" to label a node referencing the entity Infinity Stone.
	These can also be node labels that are not part of the original text and node labels that can be substituted with more specific labels. An example would be the label "Entity", this node label is too general and other node labels are better suited to categorize the entities.
Duplicate Node Labels	These are node labels that are exactly the same or essentially similar with another label. For example node labels "Movie" and "Films".

Relationship Types

Relationship types help classify the meaning/kind of connection between two nodes. A relationship must have exactly one relationship type.

Correct relationship types are the number of relationship types that are accurately generated by the KGG system relative to the total number of relationship types generated. Just like with the other elements, we first identify the incorrect relationship types, get the total, and subtract that from the total number of relationship types to get the count of correct relationship types.

Evaluating the accuracy of generated relationship types requires subjective reasoning because not all the relationship types might be extracted word-for-word from the text. Thus, we classify relationship types as correct, incorrect, or duplicate. The latter two are our error classifications.

Relationship Types Error Classifications	
Error Type	**Description**
Incorrect Relationship Types	These are relationship types that are not at all part of the text whether explicit or implied.
Duplicate Relationship Types	These are relationship types that have the same exact meaning as another relationship type and are interchangeable.

Properties

Properties are key-value pairs that are used to store data specific to particular nodes or relationships. In this evaluation, we only examined node properties. A node should have at least one property that helps identify the real-world entity the node pertains to – *i.d., name, title,* etc.

Correct properties are the number of properties that are accurately generated by the KGG system relative to the total number of properties generated. For properties, we skip keys like "description" and any other key that holds unstructured values. We also skipped properties that hold values that differentiate a node from another – *i.d., name, title, etc.* Like with the other elements, we first identify the incorrect properties, get the total, and subtract that from the total number of properties to get the count of correct properties.

Evaluating the accuracy of generated properties requires subjective reasoning to determine whether the property is correct, or whether it falls under our two property error classifications.

Properties Error Classification	
Wrong Value	Properties that hold values that are different from what is found from the text.
Non-verifiable Value	Properties that cannot be found in the text. This includes LLM hallucinations.

Evaluating the Connectivity of Generated Graphs

Graph connectivity is a measure of how well-connected a knowledge graph is. To be more specific, a connected graph is a graph where all nodes are directly or indirectly connected to all other nodes. A non-connected graph contains nodes that are separated from the main body of the graph.

Graph connectivity is one way to assess the ratio between the nodes and relationships of a knowledge graph, with a greater number of relationships generally signifying a more useful graph. The minimum number of relationships necessary to have a connected graph is the total count of nodes minus one. Actually determining whether a graph is connected requires more complex calculations. Based on the minimum number of relationships needed for a connected graph, we have created a “Connectivity Score” with the following formula:

Connectivity Score = Number of Relationships / (Number of Nodes - 1)

A score below 1 means that the graph is definitely not connected, and some nodes are separated from the whole, whereas a score above 1 means that the graph has the minimum number of relationships needed for connectivity, although connectivity is not guaranteed.

There is another popular measure of how well-connected a knowledge graph is, graph density. It compares the number of relationships in a graph with the number of relationships needed for all nodes to be directly connected with each other (not just indirectly, like what connectivity emphasizes). A knowledge graph should not necessarily be fully complete, as low-quality or incorrect relationships are not desirable, but a higher score generally indicates a more useful graph.

For context, in one assessment of AI-generated graphs, density scores were 0.05 for a human-constructed knowledge graph using a 276 character dataset, and 0.00085 using a 2,105 character dataset.

To calculate graph density, we take all the nodes and relationships generated by the KGG system and compute using this formula:

$$\mathbf{d = m / n(n-1)}$$

Where *m* represents the number of relationships and *n* represents the number of nodes.

Evaluating The Capture Rate of Generated Graphs

Nodes

Captured nodes are the number of nodes captured by the KGG system that are also present in the evaluation graph.

When comparing both graphs, one must exercise subjective judgment to determine if a generated node is truly similar to an evaluation graph’s node – for example: “ETF” vs “Exchange-Traded Funds” or “Barack O.” vs “Barack Obama”.

While comparing the systems’ generated nodes to the nodes from the evaluation graphs, we also took note of missing nodes. Missing nodes are the opposite of captured nodes and can be classified into two categories:

Missing Nodes Classifications

Type	Description
Missing Nodes	Nodes from the evaluation graphs that do not have an exact or essentially similar match from the KGG system’s graphs.
Nodes that are Properties	Nodes from the dataset that are made to be properties of another node instead. An example of this was in the dataset, actors of the characters were their own separate nodes labeled as Actors and separate from the Character nodes they played. However, in one of the system’s graphs, the actors were added as properties of the Character nodes instead.

Relationships

Captured relationships are the number of relationships captured by the KGG system that are also present in the evaluation graph.

When comparing both graphs, one must also exercise subjective judgment to determine if a generated relationship truly is similar to a dataset relationship – for example:
(john)-[:IS_DATING]->(mary) vs (john)-[:IS_IN_A_RELATIONSHIP_WITH]->(mary).

During evaluation, also we took note of the generated relationships that do not have an exact or essentially identical counterpart in the evaluation graphs.

Node Labels

Captured node labels are the number of node labels captured by the KGG system that are also present in the evaluation graph.

Subjective reasoning is required to determine whether the generated node label is a match of the dataset label. For example, if the dataset label is “Movie” and the system instead has a label “Film”, that could be considered a successfully captured label because the meaning is the same.

During evaluation, we noted if the node label from the evaluation graph did not have an exact or essentially similar node label match from the KGG graphs.

Relationship Types

Captured relationship types are the number of relationship types captured by the KGG system that are also present in the evaluation graph.

Subjective reasoning is required to determine whether the generated relationship type is a match of the dataset relationship type. For example, the relationship type “FRIENDS” and the relationship type “FRIENDS_WITH” will be considered essentially similar.

Just like with the other elements, we took notes of all missing relationship types for reporting purposes.

Properties

Captured properties are the properties in the generated graph that are also represented in the evaluation graph. The percent capture is computed by dividing the captured properties in the generated graph with the total number of properties in the dataset graph.

Property keys that identify the node such as *name, id, title, etc.* were not included because these were used to check the nodes. We also did not include the property *description* and other properties with arbitrary values.

Subjective reasoning is required to determine whether the generated property is a match of the dataset property. This requires looking at both the property key and the value. For example: {birthDate: 11-02-1995} and {dob: “November 02, 1995”} are exact matches.

Results

In our first run of KGStorage, we evaluated Unize ST-0.5 and LangChain using their default information extraction goals. The capture rates with default goals turned out to be fairly similar on both the named entity and all entity graphs, so most of our attention can be directed to our graph quality measures, accuracy and connectivity. Unize ST-0.5 achieved consistently strong performance across datasets, whereas LangChain exhibited decent performance on our smaller dataset, while suffering in our larger dataset. We present all key metrics below, with additional metrics available here.

Accuracy Rate and Connectivity

Graph Quality - Single-Input

	Correct Nodes	Correct Node Labels	Correct Relationships	Correct Relationship Types	Correct Properties	Connectivity Score
Unize ST-0.5	**95.83%** (23 out of 24)	**83.33%** (5 out of 6)	**79.31%** (23 out of 29)	**81.82%** (9 out of 11)	**100%** (2 out of 2)	**1.2609** (24 n / 29 r)
LangChain	**89.29%** (25 out of 28)	**85.71%** (6 out of 7)	**76.92%** (20 out of 26)	**66.67%** (4 out of 6)	**N/A**	**0.9630** (28 n / 26 r)
GPT-4o	**43.42%** (33 out of 76)	**68.00%** (17 out of 25)	**58.16%** (57 out of 98)	**71.70%** (38 out of 53)	**85.71%** (12 out of 14)	**1.3067** (76 n / 98 r)

On our smaller 5,364-character dataset, Unize ST-0.5 displays high performance scores overall, with all generated elements being quite accurate. LangChain's performance is slightly weaker but still strong overall, with nodes and relationships within 5% of Unize ST-0.5. However, LangChain's relationship type performance is notably lower.

GPT-4o's performance is already very poor, likely because we used a chunk size of <1,000 characters, resulting in six chunks. The more chunks that are added, the greater the duplication, since LLMs are not aware of anything beyond the context, so they do not know if an element has already been generated or not. It's important to note that because GPT-4o was instructed to extract all information, it has 3x the nodes and 4x the relationships of Unize ST-0.5 and LangChain. There are more relationships than nodes, so this results in a decent connectivity score that rises to 1.78 when the output is human corrected.

Graph Quality - Multi-Input

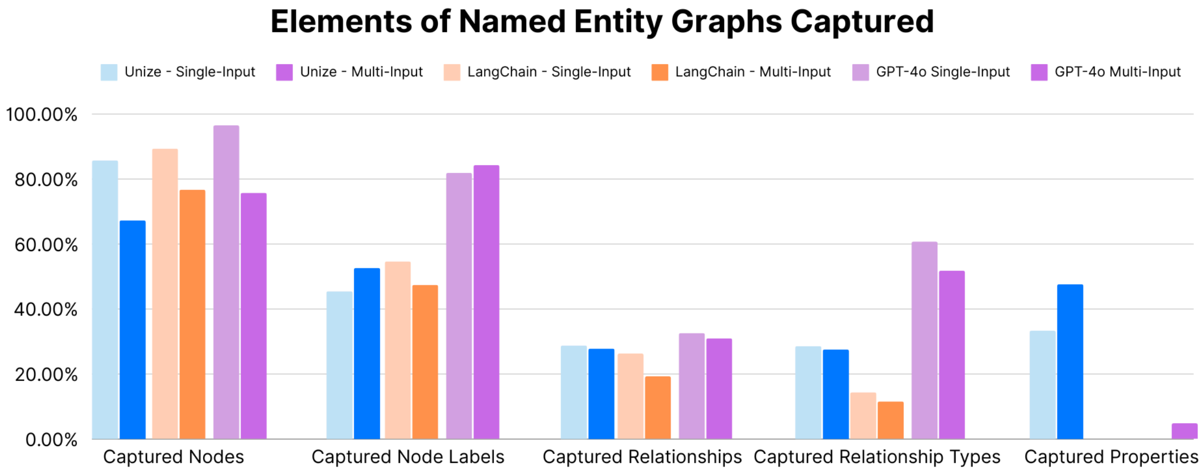
	Correct Nodes	Correct Node Labels	Correct Relationships	Correct Relationship Types	Correct Properties	Connectivity Score
Unize ST-0.5	**86.75%** (72 out of 83)	**85.71%** (12 out of 14)	**76.64%** (105 out of 137)	**84.31%** (43 out of 51)	**91.67%** (11 out of 12)	**1.6707** (83 n / 137 r)
LangChain	**57.01%** (61 out of 107)	**81.82%** (9 out of 11)	**79.75%** (63 out of 79)	**68.42%** (13 out of 19)	**N/A**	**0.7453** (107 n / 79 r)
GPT-4o	**37.25%** (130 out of 349)	**40.68%** (24 out of 59)	**44.21%** (210 out of 475)	**63.74%** (116 out of 182)	**69.44%** (25 out of 36)	**1.3649** (349 n / 475 r)

On our larger, dual-article 19,838-character dataset, Unize ST-0.5 maintains strong performance, with all metrics essentially unchanged except node accuracy, which drops 9%. Connectivity actually increases, in part due to new relationships forming between existing nodes as new information enters the graph.

Unfortunately, LangChain's node accuracy drops dramatically compared to our smaller dataset, from 89% to 57%. Connectivity also decreases. Other metrics remain largely unchanged. These results are likely because nodes can often be repeated across chunks, which introduces duplicates, whereas other things like relationships between two particular entities generally don't repeat very often (at least in our dataset), so duplicates are not created as much.

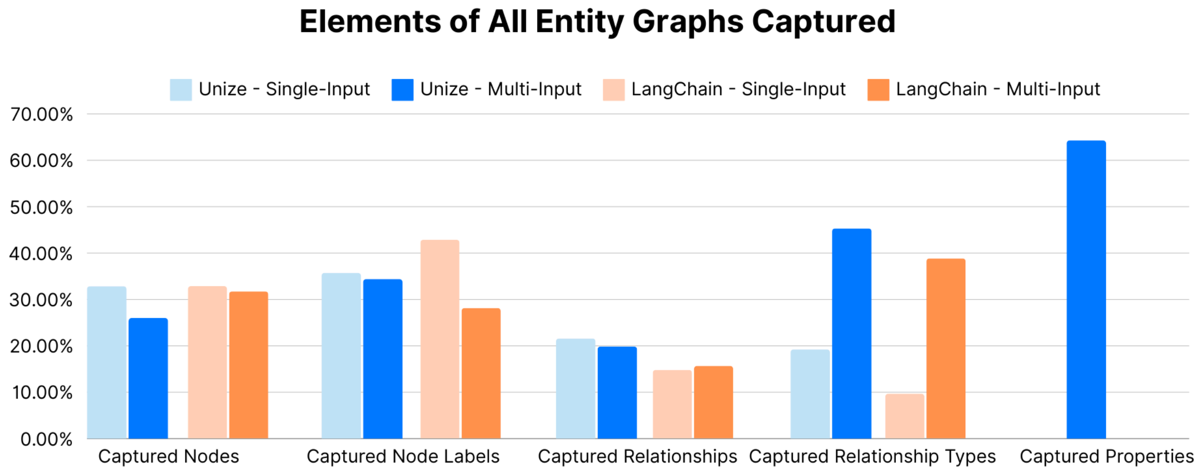
GPT-4o's performance continues to drop as more chunks are processed independently of each other, leading to more duplicates arising from identical elements across chunks.

Capture Rate



Unize ST-0.5 and LangChain exhibit similar capture rates relative to our named entity graphs, with LangChain capturing a slightly higher fraction of nodes, and Unize ST-0.5 capturing a slightly higher fraction of relationships and relationship types.

It's unfortunate that both systems don't have higher capture rates for named entities, at just 70%–85%, although that might be due to their default information extraction goals. The low figures for relationships just goes to show that it can be difficult to identify which information is worth capturing. We look forward to running these systems with clearer KGC guidelines in the future.



Results are fairly similar for all entities, with both systems capturing a low percentage of all entities mentioned, and relationship capture also remaining low. GPT-4o results are not presented, since we unfortunately corrected the GPT-4o generated graph without simultaneously recording error classifications. GPT-4o results could appear unusual in the All Entity graphs, since we used it to generate base graphs which were then human corrected.

Limitations and Opportunities

We want to flag limitations in our first run of KGStorage, as well as associated improvements we've identified for the future:

1. **Account for nondeterministic generation**. Unize ST-0.5 and LangChain are not deterministic systems, so their outputs usually vary each time they are run. While we ran both systems multiple times and believe our reported performance figures are representative of general performance, we only calculated the results based on a single system run. Ideally, we would run each system on the same dataset multiple times, and report an average of results. We were limited in our capability to do this based on the lack of automated evaluation.
2. **Introduce automated evaluation.** KGStorage currently does not have an automated evaluation system, which greatly limits the scale of benchmarking we can do. It's very time consuming to evaluate new systems as well as evaluate systems against new datasets. We hope to create an automated evaluation system in the future.
3. **Build a scalable dataset creation process.** At the moment, we have not created a process to reliably construct evaluation knowledge graphs in consistent ways. Such a process would likely require a workforce of graph constructors, as well as more reliable KGC standards. This means that we cannot assess large-scale knowledge graph generation or test systems on a wide variety of data. It may also be challenging for a workforce to create large-scale graphs in general, since merging elements into an existing graph requires checking against an ever increasing number of elements. In the

near future, we hope to manually create a larger dataset, and in the intermediate future, we hope to introduce standards and tooling to enable graph creation at scale.

4. **Introduce better knowledge graph creation standards.** As mentioned in scalable dataset creation, we're currently using two different KGC methods that result in pretty different looking graphs. Some users may prefer one over the other. Even with our documented standards, different people and AI systems end up generating very different looking graphs, especially with certain graph elements like relationships. Researching methods to create knowledge graphs that require less subjectivity and result in outputs that are more useful across a variety of use cases could not only help create scalable datasets, but also enhance evaluation as well as improve the performance of KGG systems themselves.
5. **Test alternative evaluation methods.** While we're pretty happy with our approach of comparing generated graphs against themselves and evaluation graphs, graphs are used for multiple purposes, including retrieving relevant context for LLMs with the recent advent of GraphRAG. It may be possible to measure graph usefulness by testing how well a generated graph can answer questions. This approach could largely or completely automate evaluation. We are actively working in this area—once we release our Unize Retrieval system and its associated KGRetrieval benchmark, we may want to release a benchmark that applies a retrieval method that works on different types of generated graphs, or a unified benchmark that evaluates storage and retrieval systems together.

Appendix

Cost Analysis

Generating a single-input knowledge graph from a 10k character article (we used a shortened version of the Infinity War article) using Unize incurred a bill of $0.94.

Running the same article using LangChain with GPT-4o required 3,422 input tokens and resulted in 2,845 output tokens being generated. This aligns with the general rule that 1 token is roughly equivalent to ~4 characters, once LangChain's prompt, repeated across chunks, is taken into consideration.

GPT-4o costs $5/1 million input tokens and $15/1 million output tokens. The cost came out to $0.06, making Unize ST-0.5 ~15.7x more expensive than running LangChain with GPT-4o. The cost breakdown is as follows:

Input Cost: ($5 / 1000000 tokens) * 3422 tokens = $0.01711
Output Cost: ($15 / 1000000 tokens) * 2,845 tokens = $0.042675
LangChain with GPT-4o Total Cost: $0.059785

Cost Relative to Unize: $0.94 / $0.059785 = 15.723 (~15.7x)

GPT-4-32k model costs $60/1 million input tokens and $180/1 million output tokens. Assuming GPT-4-32k generates the same number of output tokens as GPT-4o (we did not actually run

GPT-4-32k, but results shouldn’t be too dissimilar), the cost to process the same article would be $0.71. This would make Unize ST-0.5 1.3x more expensive than GPT-4-32k. The cost breakdown is as follows:

Input: ($60 / 1000000 tokens) * 3422 tokens = $0.20532
Output: ($1800 / 1000000 tokens) * 2,845 tokens = $0.5121
LangChain with GPT-4-32k Total Cost: $0.71742

Cost Relative to Unize: $0.94 / $0.71742 = 1.310 (~1.3x)