On the Efficacy of Using LLMs for Context Driven Entity Augmentation in Property Graphs*

Felipe Vasconcelos[1,**], Cristina Aguiar[1,**], Alexandre Chanson[2], Mirian Halfeld-Ferrari[3], Patrick Marcel[3,***] and Verónika Peralta[2]

[1]ICMC, University of São Paulo, São Carlos, Brazil
[2]LIFAT, Université de Tours, France
[3]Université d'Orléans, INSA CVL, LIFO, UR 4022, Orléans, France

Abstract

This paper investigates the use of large language models (LLMs) to complete missing attribute values in property graphs. Addressing this problem requires leveraging both the structural context of a node and the attribute information already present in the graph, and we explore the use of LLMs to infer or retrieve plausible attribute values from external sources. We propose an experimental protocol to evaluate LLM-based data augmentation in terms of accuracy, prompting effort, and environmental impact, and introduce a difficulty metric inspired by perplexity. Our preliminary results show that most of the models tested struggle with augmentation tasks and that context is useful for difficult tasks, with variations in the model and prompting strategy used.

Keywords

Data augmentation, property graphs, LLMs

1. Introduction

Graph databases now play a key role in modeling complex, highly interconnected data. Often built from heterogeneous sources, they evolve over time, leading to the progressive enrichment of nodes, relationships, and properties. Consequently, many real-world graph datasets contain missing information (nodes lack property values, relationships only partially defined, etc.) which hinders the effectiveness of querying, analysis, and reasoning with interconnected data. Data generation, discovery or augmentation are emerging as promising mechanisms for enriching graph data and addressing information incompleteness [1].

The goal of this paper is to propose an approach for completing data graphs with missing values for selected node attributes. We explore the use of large language models (LLMs) to infer or retrieve plausible attribute values from external sources. Their ability to exploit broad web-scale knowledge makes them a convincing alternative to manually crafted extraction workflows. However, relying on LLMs raises important questions regarding cost, reproducibility, and the provenance of generated information. In this work, we examine different models handling data augmentation tasks of various difficulty, and study how to balance the accuracy of the answers with the effort required to design prompts. We also discuss the environmental impact of the proposed solutions.

While existing research has mainly focused on inferring or predicting missing edges, the completion of attribute values remains underexplored. Addressing this problem requires to consider both the

DOLAP 2026: 28th International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data, co-located with EDBT/ICDT 2026, March 24, 2026, Tampere, Finland

*This work was partially supported by JUNON Program (DATA/LIFO) with financial support of Région Centre-Val de Loire, France, and partially supported by project TopOL with financial support of the French National Research Agency (ANR).

**The first and second authors were supported by the São Paulo Research Foundation (FAPESP), the Brazilian Federal Research Agency (CNPq), and the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, Brasil (CAPES), Finance Code 001.

***Corresponding authors

felipevsc@usp.br (F. Vasconcelos); cdac@icmc.usp.br (C. Aguiar); Alexandre.Chanson@univ-tours.fr (A. Chanson); mirian@univ-orleans.fr (M. Halfeld-Ferrari); Patrick.Marcel@univ-orleans.fr (P. Marcel); Veronika.Peralta@univ-tours.fr (V. Peralta)

0009-0007-7922-9746 (F. Vasconcelos); 0000-0002-7618-1405 (C. Aguiar); 0000-0001-9195-5950 (A. Chanson); 0000-0003-2601-3224 (M. Halfeld-Ferrari); 0000-0003-3171-1174 (P. Marcel); 0000-0002-9236-9088 (V. Peralta)

structural context of a node and the attribute information already available in the graph. In this paper, we assume a scenario in which, for each node, we construct prompts that include: (1) a set G of existing ("property", "value") pairs, and (2) a set H of property names for which values are sought. The selection of properties included in G and H takes into account the popularity of nodes and attributes within the graph, allowing us to analyse different task difficulties. Moreover, building on our previous work [2], G also incorporates contextual information derived from the graph.

Overall, this paper investigates to what extent LLMs can reliably enrich property graphs and whether contextual information can guide this process. Its main contributions are: (i) a protocol to assess the efficacy of using LLMs for data augmentation in property graphs, in terms of accuracy, prompting effort and sustainability (measured via an environmental impact score), (ii) a novel metric to quantify the difficulty of the data augmentation tasks, inspired by perplexity [3], (iii) tests showing that all models struggle even for relatively easy tasks, and that context helps improving accuracy for difficult tasks.

Paper organization. Section 2 describes our experimental protocol. Section 3 explains our prompting strategies, the models we selected and details how we computed environmental impact. Section 4 presents our results. Section 5 discusses related work and Section 6 concludes the paper.

2. Experimental protocol

This section presents our experimental protocol, detailed in Algorithm 1. The aim is to evaluate three metrics representing the prompting effort, result accuracy and process sustainability when performing entity augmentation tasks of varying difficulty in a property graph.

Algorithm 1 Protocol

Require: A set M of models, a set T of prompt templates, an odd integer r for the number of runs, a set of node N from property graph PG of a given type t

Ensure: Accuracy, effort, sustainability for combinations of models in M, instances of template in T

for each node n of N **do**
  compute $cent(n)$ the centrality of n and $context(n)$ the context of n
  for each property p in $context(n)$ **do**
    compute $pop(p)$ the popularity of p
  end for
end for
$TASKS = taskDefinition(N)$ ▷ See Algorithm 2
$Result = \emptyset$
for each model m in M **do**
  for each prompt template t in T **do**
    for each task $(n, H, G) \in TASKS$ **do**
      let e be the edited prompt t with n, H, G
      $R = \emptyset$
      for 1 to r **do**
        let res be the result of prompting model m with e
        $R = R \cup \{res\}$
      end for
      final results $R_{m,e} = aggregate(R, m, e)$
      $scores = ((n, H, G), Accuracy(R_{m,e}), effort(R_{m,e}), sustainability(R_{m,e}))$
      $Result = Result \cup \{scores\}$
    end for
  end for
end for
return Result

Algorithm 2 Tasks definition

Require: A set of nodes N with their respective contexts and centralities
Ensure: A set of tasks
$TASKS = \emptyset; D = \emptyset$
Let Q_0, Q_1, Q_2, Q_3, Q_4 be the quartiles of $cent()$ for N
for $Q_i \in \{Q_1, Q_2 \cup Q_3, Q_4\}$ **do**
    $D = D \cup$ 3 randomly drawn nodes with cent() in $[Q_{i-1}, Q_i]$
end for
for each node n of D **do**
    Let p_1, p_4 be the properties with $pop()$ resp. before 1st and after 3rd quartile for $context(n)$
    Let $prop(n)$ be the properties of n
    Let $H_1 =$ randomly draw 3 properties in $(p_1 \cap prop(n))$
    Let $H_4 =$ randomly draw 3 properties in $(p_4 \cap prop(n))$
    Let $G_1 =$ randomly draw 3 properties in $p_1 \setminus H_1$
    Let $G_4 =$ randomly draw 3 properties in $p_4 \setminus H_4$
    $TASKS = TASKS \cup \{(n, H_1, G_1)\} \cup \{(n, H_1, G_4)\} \cup \{(n, H_4, G_1)\} \cup \{(n, H_4, G_4)\}$
end for
return $TASKS$

2.1. Metrics

Algorithm 1 uses three metrics: accuracy, prompt formulation effort and sustainability. Accuracy consists of comparing the result given by the model with the ground truth, i.e., the values of properties hidden. Both results and ground truth are sets of pair p, v with p a property name (or property for short) and v a value. Measuring accuracy is done with the classical F1 score, aggregating precision and recall. Prompt Formulation Effort (PFE) measures human effort in crafting the different prompts passed to the model. As explained in Section 3.2 we have compact and base templates, and can instantiate these templates using 0 shot, few shots and Chain Of Thought (CoT). A simple proxy metric for this effort is the number of tokens in the prompt. Finally, sustainability relates to the choice of model and the parameters used. We measure this with a composite score described in details in Section 3.3.

2.2. Entity augmentation tasks and their difficulty

The task of the model is to predict the values of some node's properties, hidden from it. Formally, a task is a tuple (n, H, G) where n is a node, H is the set of properties for which we seek to predict values and G is the set of properties with their values, known by the model. Tasks are set by Algorithm 2. We posit that the difficulty of the task will vary with both the characteristics of the properties to predict, i.e., H, and the information passed to the model, i.e., G, that includes the specific node n, some of its properties and some properties in its context. Precisely, we assume the following:

- the more central n, the easier its properties in H will be to predict. For instance, if airports are ranked by node centrality (using e.g., page rank), then airports of high centrality likely correspond to big hubs whose properties should be easier to retrieve by the model,
- the more popular the property in H (in terms of presence in the graph, i.e., the probability that the property has a value when observing a node of a given type), the easier to predict,
- the higher the centrality of a property's element (node or arc) in a given node's context, the easier it is for the model to use it (e.g., if the airport city name of some highly central cities is in G, it will be easier for the model to retrieve the airport's hidden properties),
- the more popular the property in G, the more likely it is that it appears in the training examples of the model, and consequently the easier it is for the model to use it,
- the more properties in G given to the model (including properties from the node's context), the easier to predict.

element		property		$pop(p)\times$	
e	$cent(e)$	p	$pop(p)$	$cent^p(e)$	log
e1	0.9	a_1	1.0	0.9	-0.15
e1	0.9	a_2	1.0	0.9	-0.15
e2	0.5	b	1.0	0.5	-1.00
e3	0.2	c	0.9	0.18	-2.47
e2	0.5	d	0.7	0.35	-1.51
e3	0.2	e	0.5	0.10	-3.32
e2	0.5	f	0.3	0.15	-2.74
e1	0.9	g	0.1	0.09	-3.47

task node	task	specificity	difficulty
e_1	H1 = $\{a_1, a_2\}$	1.16	2.94
	G1 = $\{b, a_1\}$	1.77	
e_1	H1 = $\{a_1, a_2\}$	1.16	24.48
	G2 =$\{f, g\}$	22.31	
e_2	H2 = $\{f, d\}$	8.37	15.72
	G3 = $\{c, d\}$	7.34	
e_2	H2 = $\{f, d\}$	8.37	30.69
	G2 = $\{f, g\}$	22.31	

Table 1
Example of various tasks and their difficulty.

These considerations will drive our definition of task difficulty. To define this difficulty, we first need to define the context of a node.

Definition (Context) The context of a node n is a set of properties coming from n, its direct neighbor nodes, the relationships to its direct neighbor nodes, and the properties along paths where relationships have a $n:1$ cardinality, starting from n (like e.g., relationship IN between a city and a country since a city is IN only one country). It is computed with function $context()$.

Definition (Popularity of a property) Given a property p of an element (node or edge) e of type t, we call $pop(p)$ the popularity of p, defined as the probability of observing a (non null) value for property p when drawing an element in the set of elements of type t. This gives a probability distribution P for random variable p with two outcomes: $P(p \neq null)$ and $P(p = null)$. Note that we do not consider the probability distribution over the values of properties because the majority of values will be specific to the element and be present only once (e.g., latitude, name, length of runways, etc.).

Definition (Popularity of an element) The popularity of an element (node or edge) e is given by a centrality measure $cent(e)$, assumed normalized, for this element. Alternatively, centrality can be replaced with, when randomly drawing a node in the graph, the probability of the node being source or target of the relationship.

Definition (Specificity of a set of properties) Inspired by [3], we measure the specificity of a set of properties S based on the popularity of the properties in S. In [3], specificity is measured using perplexity, a standard measure of uncertainty. Intuitively, a lower specificity implies more generic prompts while a higher specificity corresponds to more detailed, specific prompts, i.e., prompts with less expected tokens (knowing the previous tokens used). In our context, since we use the popularity of properties, the more specific the set is, the more difficult it is to predict. This is why our specificity measure is: $specificity(S) = exp(-\frac{1}{|S|}\sum_{p\in S} log(pop(p) \times cent^p(e)))$, where $cent^p(e)$ is the centrality of the element that has property p.

Definition (Difficulty of a task) We define an a priori (in the sense that it is based only on the information to be placed in the prompt, before sending any prompt to any model) difficulty score for the task to give to the model, as follows. Let n be a node of type t in a graph PG, $cent(n)$ its centrality (assumed normalized), $context(n)$ its context, and H and G two disjoint subsets of $context(n)$ such that G and H are not $\emptyset$ and H has only properties of n. The difficulty of the task $T = (n, H, G)$ is the cumulated specificity of H and G: $dif(T) = specificity(H) + specificity(G)$.

Example Table 1 illustrates the difficulty of various tasks. Table 1 (left) lists various properties p coming from different elements e, with the centrality of e, the popularity of p and the result of $log(pop(p) \times cent^p(e))$. Table 1 (right) lists various tasks made from these properties, with for each the

specificity of sets H, G composing the task and its difficulty. It can be observed that, as desired, tasks with a central node and popular properties, like (e_1, H_1, G_1), have a low difficulty while tasks with a node of lower centrality and less popular properties, like $(e_2, H2, G2)$, have a much higher difficulty. Desirably, medium hard tasks like $(e_2, H2, G3)$, where H and G are balanced in terms of specificity, are scored less difficult than tasks like $(e_1, H1, G2)$ where H and G are unbalanced with G very specific.

2.3. Overview of the protocol

Algorithm 1 starts by computing nodes centralities, contexts and properties popularities (lines 1-7). Based on those, tasks are defined by drawing randomly (i) nodes of various centralities and for each (ii) properties of various popularities in sets H and G. (see Algorithm 2). Then for each model, tasks are used to instantiate various prompt templates (line 10-13) and models are prompted (line 16). Finally, as the model's answers to a specific prompt may vary, we will pass the prompt run many times (lines 15-18), and aggregate (line 19) the model results (see Section 4.2 for details).

3. Prompt engineering, model selection and sustainability

3.1. Model selection

We selected state-of-the-art proprietary large language models based on their performance in benchmarks such as LMArena[1] and Livebench [4]. We also selected smaller versions of these models to assess whether reduced-capacity models can achieve comparable performance with lower environmental cost. We chose proprietary models over open source models since the former are significantly more efficient and most commonly used on AI platforms [5].

We evaluate Gemini-2.5-Pro, Gemini-2.5-Flash, GPT-5, GPT-5-mini, Claude-4.5-Sonnet, and Claude-4.5-Haiku. We exclude Gemini-3-Pro, as it does not provide a smaller counterpart, and Claude Opus, following Anthropic's recommendation to use Sonnet for general-purpose tasks [6]. All models are evaluated through their official APIs using identical prompts and default configuration settings. We did not use any form of retrieval-augmented generation (RAG). Instead, as explained in Section 2, we rely on explicitly provided structured context, extracted from the graph.

3.2. Prompt Engineering Techniques

Prompt engineering refers to designing system instructions to guide the models for performing a specific task, without modifying its underlying parameters or weights [7]. Prompt engineering can improve results and perform similarly to fine-tuning without the cost of retraining the model [8]. Different techniques are available for performing prompt engineering, such as AutoPrompt, Tree of Thoughts, and Self-Consistency [7]. In this work, we focus on the most common ones: zero-shot, few-shot (1,3,5), and Chain-of-Thought (CoT). We also test a smaller, compact version of the base zero-shot prompt.

Figure 1 (left) summarizes the evaluated prompt techniques and their corresponding token lengths. For each prompt technique, we compute the number of tokens using the token-counting functions provided by the official APIs of the respective models and report the average token count across runs. The base prompt (zero-shot) has an average length of 219 tokens, whereas the compact variant contains 103 tokens. Prior work suggests that shorter prompts can lead to reduced computational overhead and energy consumption during inference, particularly for models that are sensitive to input length [9].

Our base prompt follows a zero-shot setting without CoT, in which the model is instructed to act as an assistant that enriches entities extracted from a database. Specifically, the model is asked to identify the most likely real-world entity corresponding to the input and to retrieve recent and trustworthy values for a predefined set of properties. The output is constrained to a CSV format to facilitate downstream processing and automated evaluation. Following prior work, we employ structured tags in the prompt, as this has been shown to improve response consistency and extraction quality. The user input consists

[1] https://lmarena.ai/leaderboard

Prompt technique	Tokens#
Compact	103
Zero-shot (Base)	219
1-Shot	258
Zero-shot + CoT	273
1-Shot + CoT	315
3-Shot	325
3-Shot + CoT	377
5-Shot	414
5-Shot + CoT	468

Models tested
Gemini-2.5-Pro
Gemini-2.5-Flash
GPT-5
GPT-5-mini
Claude-4.5-Sonnet
Claude-4.5-Haiku

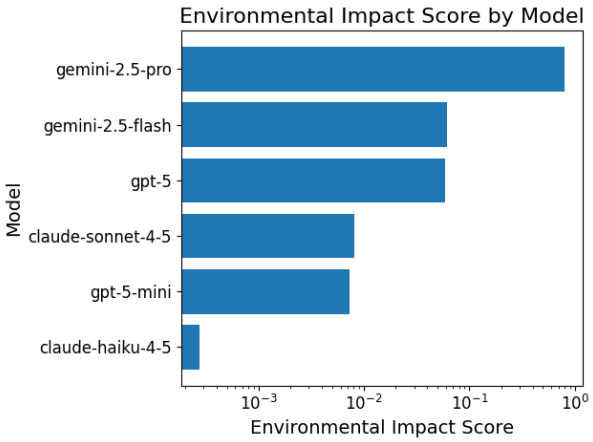

Figure 1: Prompting technique, ranked by formulation effort (mean number of tokens), models tested and Composite Environmental Impact Score (EIS) by Model

of the properties extracted from the target node, augmented with contextual information (Section 2). This mechanism resembles a RAG setup, but avoids the need for vector representations or similarity search, relying instead on explicitly provided structured context.

3.3. Environmental Costs of Models and Prompts

We analyze the environmental footprints of the selected models and prompting techniques using EcoLogits [10], a Python library to estimate the environmental impacts of LLM inference. While we use On-line LLMs in our tests, we rely on this library to account for differences in model size and hardware for each provider. Different solutions for calculating LLMs and machine-learning environmental costs exists, such as CodeCarbon[2] and MLEnergy [11]. We selected EcoLogits for various reasons. In particular, it focuses on inference-time computation and support for third-party provider APIs. It estimates the costs for each model and provider and attempts to normalize them to enable comparison. The library evaluates several sustainability indicators, besides energy consumption (EC). The Global Warming Potential (GWP) quantifies the contribution to climate change in CO_2 equivalents. The Water Consumption Footprint (WCF) accounts for water use associated with data center cooling and electricity generation. The authors of EcoLogits [10] adopt a Life Cycle Assessment (LCA) methodology to estimate environmental footprints based on model parameter counts and the most common hardware used for inference by each provider. To evaluate the sustainability enabling a unified comparison of the environmental footprint across models and prompting techniques we use, we derive a Composite Environmental Impact Score (EIS) as follows. We compute the EC, GWP, and WCF indicators for each model–prompt combination. Each configuration is executed five times, and the mean value of each indicator is stored. To ensure comparability between indicators with different scales, we apply a log-based min–max normalization. The normalized indicators are then aggregated using an unweighted average.

An overall environmental efficiency ranking based on this composite EIS is presented in Figure 1 (right), where lower values indicate lower relative environmental impact. The results show that the Gemini models exhibit the highest impact scores, while smaller models like Claude-Haiku-4.5 and GPT-5-mini achieve the lowest scores. Figure 2 presents the EIS of our different prompting techniques for the selected models. It can be seen that prompting strategies may induce substantial differences in energy consumption independently of prompt length or apparent complexity. These differences depend on how effectively additional context constrains model generation and reasoning. Interestingly, bigger prompts do not imply more energy, as models can spend more time reasoning for smaller prompts. For instance, for Claude models, few-shot prompting consistently yields lower environmental impact than

[2]https://codecarbon.io/

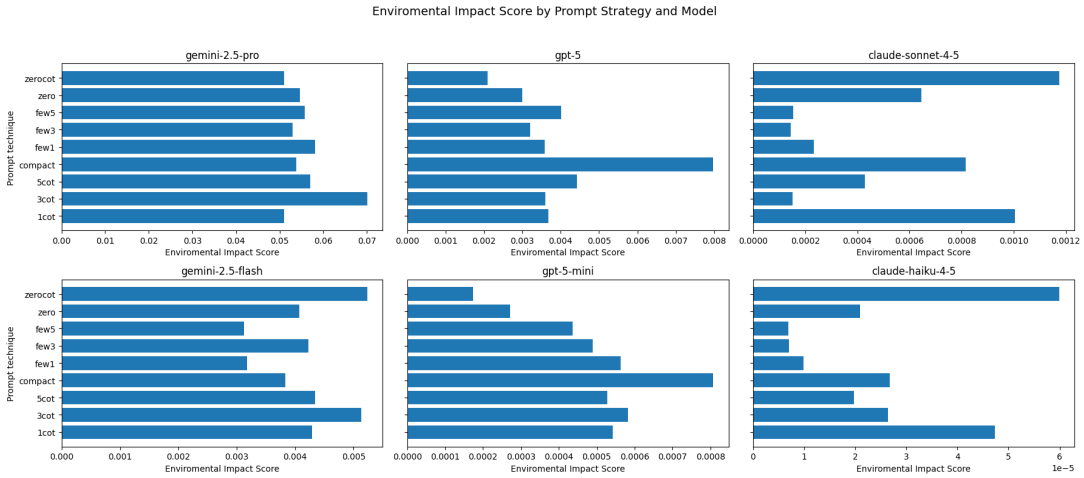

Figure 2: Composite EIS by Prompt Strategy and Model

Airport	Task	Specificity	Difficulty
PAKK	H = $\{'runways', 'iata', 'runwaysLength'\}$	4.37	16.72
	G = $\{'runwaysWidth', 'commercialFlights'\}$	12.4	
PAKK	H = $\{city', 'longitude', 'source'\}$	3.29	7.77
	G = $\{'runwaysLength', 'runways', 'runwaysWidth'\}$	4.48	
VRMT	H = $\{'runways', 'iata', 'commercial\_flights'\}$	22.16	30.65
	G = $\{city', 'tz\_database\_timezone', 'altitude'\}$	8.14	
VRMT	H = $\{'icao', 'timezone', 'city'\}$	8.49	16.96
	G = $\{dst', 'type'\}$	8.47	

Table 2
Tasks and their difficulty from the Airports database.

zero-shot prompting. Additionally, compact prompts often exhibit higher energy consumption than some longer prompt configurations, due to little information available for the model.

4. Experiments

This section details the tests we ran and the lessons learned, starting with the presentation of the dataset used and describing our implementation.

4.1. The Airport dataset

To evaluate our approach, we built a property graph database on air routes and trade, comprising three node types: `Airport` nodes with geographic coordinates and ICAO codes; `City` nodes with population (in millions) and pollution indices; and `Country` nodes characterized by numerical indicators such as GDP and birth rate. Edges represent different relationships: `FLOW` captures passenger exchanges between airports with property nPAX, ROUTE_TO represents flights between airports, and TRADE models trade flows between countries. We built a Neo4j database instance from multiple Kaggle and Eurostat datasets. An ETL pipeline was implemented to extract, clean, normalize, and integrate the data, ensuring consistency and integrity. The data were enriched with computed properties (e.g., distances between airports and between airports and city centers). Flight prices and durations were collected via web scraping using the FastFlights[3] library. This database exemplifies a dataset rich in numerical properties but poor in textual information. Since textual data are harder to obtain, this setting is well suited for data augmentation using external LLM knowledge. For instance, no dataset explicitly identifies airline hub airports, but this can be easily obtained from external knowledge sources or web searches.

[3]https://aweirddev.github.io/flights/

Table 2 reports the difficulty scores computed over the Airports dataset for Koyuk Alfred Adams Airport (PAKK), located in the United States, and Kaadedhdhoo Airport (VRMT), located in the Maldives. The results show that tasks involving properties classified as easy to find, such as an airport's city or altitude, yield lower difficulty scores. In contrast, even when similarly easy properties are used in the H and G sets, less popular airports, such as VRMT, exhibit higher difficulty scores, reflecting the increased challenge of retrieving reliable information for less prominent entities.

4.2. Implementation

We implemented the protocol described in Section 2 using Python. The code used is publicly available at GitHub[4]. The centrality function $Cent()$ was computed using PageRank, while the property popularity function $pop(p)$ was estimated from the Airports dataset. The context function $context()$ was computed following the approach described in [2]. Based on these quantities, tasks were generated as described in Algorithm 2. Model inference was performed using the official APIs of each provider, leveraging batch inference. Finally, we applied the following two post-processing steps to the collected outputs.

Since LLMs are inherently non-deterministic, we apply a voting function to obtain a single representative result per configuration. This function relies on data integration techniques tailored to the output data type [12]. For textual attributes, we select the most frequent value while for numerical attributes, we compute the mean. We also introduce a tolerance function to account for cases where model outputs are semantically correct but do not exactly match the ground truth. For example, a ground-truth value such as *"Guarulhos–São Paulo Airport"* may be returned as *"Guarulhos/São Paulo Airport"*. To handle such variations, we apply different similarity criteria depending on the data type. For textual values, we use a overlap coefficient with a threshold of 75% similarity. We employed the overlap coefficient because it measures containment of the ground-truth tokens rather than penalizing additional terms. For integer values, we rely on a relative-error–based similarity measure. For geospatial attributes (latitude and longitude), we compute the Haversine distance and consider predictions within a 10 km radius as correct. Any model output satisfying the corresponding threshold is counted as a correct answer.

4.3. Lessons learned

Model	F1 Score	Recall	Context (Y/N)	Effort	Mean EIS
Claude Sonnet 4.5	0.8	0.66	Y	414	0.0076
Claude Haiku 4.5	0.54	0.37	Y	103	0.0002
GPT-5	0.67	0.5	N	377	0.0581
GPT-5 Mini	0.57	0.4	N	325	0.0076
Gemini 2.5 Pro	0.65	0.48	Y	468	0.8042
Gemini 2.5 Flash	0.61	0.44	Y	468	0.0611

Table 3
Best accuracy of each model

Difficulty vs Accuracy Table 3 reports the best accuracy for each model with the corresponding effort and the EIS normalized with log minmax and bound to 0-1. It should be noted that precision is steadily equals to 1 for all models and situations (with or without context), since models are explicitly asked to produce results only when they are confident. This means that accuracy equals recall. It can be seen that all models struggle to achieve a F1 score greater than 0.8. We note that recall may be impacted by our prompting policy: LLM answer only when they are confident that the values are trustworthy, and we only count answers that strictly follow the requested CSV structure. Modifying this policy is part of our future work.

[4]https://github.com/felipeVsc/llmaugmentation

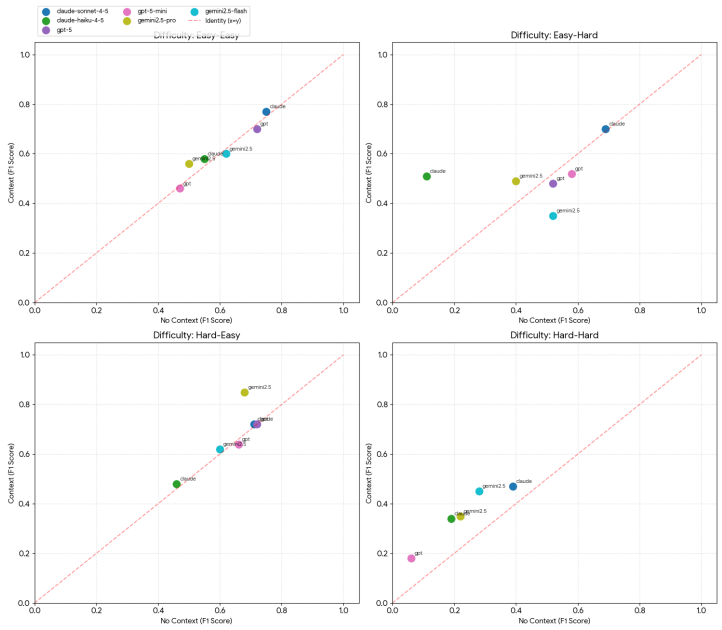

Figure 3: Accuracy vs Difficulty (of H and G)

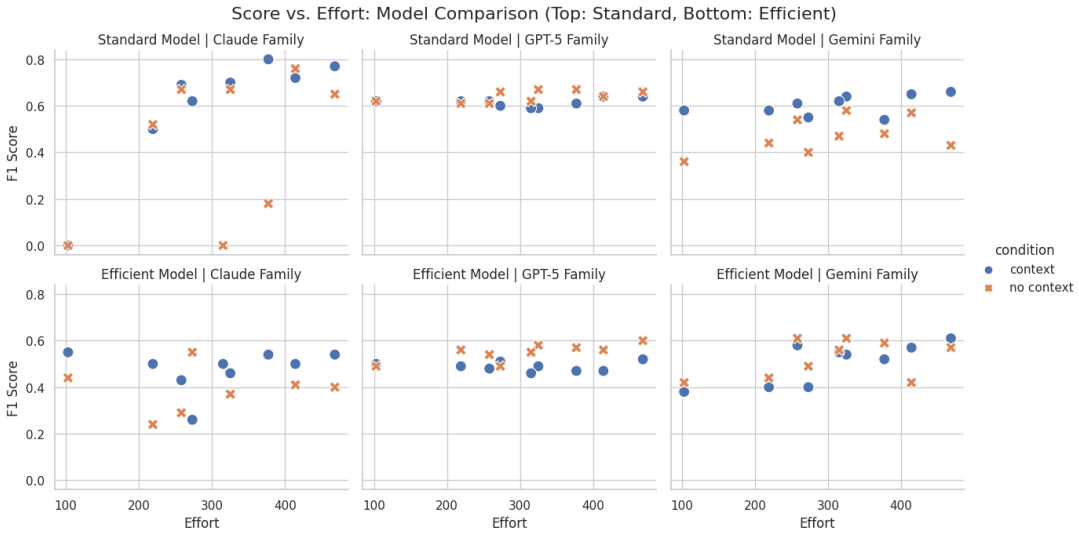

Figure 4: Accuracy vs Effort

Figure 3 plots accuracy vs difficulty of H and G (computed on the ground truth) for all the models tested, with and without context (same definitions as above). Difficulty of H and G is given as: Easy (property popularity in the fourth quartile) and Hard (property popularity in the first quartile). It can be seen that accuracy decreases with difficulty, as expected, and that context is useful when difficulty is high (hard-hard quadrant, where accuracy is systematically above the diagonal).

Effort vs Accuracy Figure 4 plots accuracy vs prompting effort for all the models tested, with and without context. Without context means that all properties in sets H and G come from node n while with context means that properties in G are randomly chosen from the node n and its context. It can be seen that achieving a better accuracy with more human effort is very model dependent. This is related to the number of extra tokens generated by models with different prompting strategies (not reported due to lack of space).

5. Related work

This section reviews complementary strategies to complete missing values, such as data discovery and data augmentation, and also highlights studies on LLM sustainability.

Data Discovery. *Data discovery*, also called *dataset search* or *table search*, concerns the discovery, exploration and retrieval of datasets related to an information need [13]. It was initially developed for *data integration*, aiming to enrich a dataset with additional instances (unionable datasets) or attributes (joinable datasets). Data discovery is more complex than keyword-based search, as it must account for factors such as data provenance, annotations, quality, and content and schema granularity to assess a dataset's fitness for use [13]. Many work focus on identifying a relevant subset of data sources within an organization (typically stored in a data lake) that are semantically similar to a query table [14]. They propose varied strategies for building dataset profiles, which are then used to build indexes or knowledge graphs for efficient query evaluation [15, 16]. Other work explores Web search by combining information retrieval and Semantic Web techniques, including *entity search* and *Web table search* [13, 17]. Many systems build knowledge graphs interconnecting datasets and columns while capturing their metadata and semantics, offering structured (e.g. SPARQL) or natural language queries [18, 19]. Recent approaches leverage language models for data discovery tasks [20], encoding datasets which are then retrieved by comparing vector representations [21, 22]. AutoDDG [23] leverages LLMs to automatically enrich summaries with semantic information and produce human-readable descriptions. Recent methods for table search are largely embedding-based, and the use of LLMs for these tasks remains underexplored [20]. A limitation of all these systems is the need to build and maintain huge repositories, while our proposal leverages context and already trained models.

Data Augmentation. Data augmentation (DA) expands and diversifies training datasets, reducing the need for new data collection and enriching models with synthetic examples. DA for Named Entity Recognition is surveyed in [24], emphasizing preservation of morphological, syntactic, and semantic properties. In contrast, [25] adopt a broader NLP view, framing LLM-based augmentation from data- and learning-centric perspectives. Both stress that synthetic data must satisfy task constraints rather than factual accuracy. Graph-based modeling is increasingly used for complex interactions, but sparse, noisy, and irregular data hinder learning. Graph-specific data augmentation (GDAug) improves generalization by modifying graph data, though graphs' non-Euclidean structure and interdependent topology and semantics make this challenging. GDAug techniques are surveyed in [1], proposing a taxonomy based on the hierarchy of graph elements. They cover heterogeneous, temporal, spatio-temporal graphs, and hypergraphs, and evaluate augmentation via downstream task performance and metrics assessing structural preservation and diversity. Although [1] ignores LLM-based augmentation, their evaluation criteria and methodology inspire us. LLMRec [26] uses LLMs to augment user–item graphs (inferring edges, enriching item attributes and profiling users) improving recommendation quality.

Sustainability of LLM. The widespread use of LLMs across domains raises growing concerns about their environmental and economic sustainability. The systematic review in [27] analyses LLM electricity consumption across two stages. While *training* is far more energy-intensive per run, it is infrequent; *inference*, executed continuously at scale, ultimately dominates cumulative electricity demand. The paper proposes a framework capturing distinct electricity-related challenges and emphasises the need for a holistic approach beyond the sole responsibility of developers. Inference-time energy usage is examined in [28]. The authors introduce EnergyMeter, an open-source tool that monitors CPU, GPU, memory, and storage energy with minimal overhead. Results show that software-level monitoring is feasible; latency alone does not predict energy use; model size increases latency and consumption sub-linearly due to parallelisation; and larger batch sizes reduce energy per token until GPU saturation. In [9], the authors study how linguistic complexity in prompts affects inference energy, showing that model architecture remains the dominant factor for efficiency, with prompt complexity playing

a secondary but meaningful role. As explained in Section 3.3, we use the Python library [10], that estimates energy use and hardware-related impacts through a bottom-up Life Cycle Assessment.

Measuring prompting effort and difficulty of LLM tasks The effort to prompt language models is often measured as an interaction cost, for instance by means of the time spent by human to craft the prompt or the number of refinements or clarification queries needed [29, 30, 31]. Recent works explicitly introduce a task difficulty metric for LLMs, mostly in two flavors, (i) regarding the model's output signals, or (ii) regarding external reference points. The first category can be thought of as defining a posteriori metrics to observe the model's results or internals while producing the result. For instance, [32] express difficulty as the point at which adding more context (sentences) stops significantly lowering the model's error. [33] quantify difficulty based on the information required to solve the task. On the other hand, metrics can be based on external reference points, in an a priori fashion since this is computed before sending any prompt to the model. For instance, [3] use perplexity of the prompt as a proxy for how specific or vague a prompt is. [34] define difficulty the amount of time a qualified human expert would take to complete the task. [35] quantify the difficulty by analyzing the internal numerical representation of the input prompt.

6. Conclusion

This paper proposes an experimental protocol to evaluate LLM-based data augmentation tasks of various difficulties in property graphs, in terms of accuracy, prompting effort, and environmental impact. Our preliminary results show that most of the models tested struggle with such tasks and that context is useful for difficult tasks, with variations in the model and prompting strategy used. Our current work includes the evaluation of our approach over various datasets (for instance from the Open Graph Benchmark[5]) and a comparison to baseline data augmentation approaches. On a longer term, we plan to investigate how techniques like RAG or agentic AI could help in achieving better accuracies.

Declaration on Generative AI

The scientific contribution being about the use of LLMs to perform data augmentation tasks, the use of Generative AI tools is described in Sections 3 and 4. The authors have not employed any Generative AI tools for text generation or image creation.

References

[1] J. Zhou, C. Xie, S. Gong, et al., Data augmentation on graphs: A technical survey, ACM Comput. Surv. 57 (2025) 274:1–274:34.

[2] C. Aguiar, J. Chabin, A. Chanson, M. Halfeld-Ferrari, N. Hiot, N. Labroche, P. Marcel, V. Peralta, F. Vasconcelos, Extracting node comparison insights for the interactive exploration of property graphs, 2025. arXiv:2512.15157.

[3] O. Kim, Detail matters: Measuring the impact of prompt specificity on reasoning in large language models, 2025. arXiv:2512.02246.

[4] C. White, S. Dooley, M. Roberts, et al., Livebench: A challenging, contamination-limited LLM benchmark, in: The Thirteenth International Conference on Learning Representations, 2025.

[5] F. Nagle, D. Yue, The latent role of open models in the ai economy, Available at SSRN 5767103 (2025).

[6] Anthropic, Claude models overview, https://platform.claude.com/docs/en/about-claude/models/overview, 2025.

[5] https://ogb.stanford.edu/

[7] P. Sahoo, A. K. Singh, S. Saha, V. Jain, S. Mondal, A. Chadha, A systematic survey of prompt engineering in large language models: Techniques and applications, arXiv preprint arXiv:2402.07927 (2024).
[8] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al., Language models are few-shot learners, Neurips 33 (2020) 1877–1901.
[9] V. D. Martino, M. A. Zadenoori, X. Franch, et al., Green prompt engineering: Investigating the energy impact of prompt design in software engineering, 2025. arXiv:2509.22320.
[10] S. Rincé, A. Banse, Ecologits: Evaluating the environmental impacts of generative ai, Journal of Open Source Software 10 (2025) 7471.
[11] J.-W. Chung, J. J. Ma, R. Wu, et al., The ML.ENERGY Benchmark: Toward automated inference energy measurement and optimization, in: NeurIPS Datasets and Benchmarks, 2025.
[12] F. Naumann, J. Bleiholder, Conflict handling strategies in an integrated information system, in: Proceedings of the WWW Workshop in Information integration on the Web (IIWEB), 2006.
[13] A. Chapman, E. Simperl, L. Koesten, et al., Dataset search: a survey, VLDB J. 29 (2020) 251–272.
[14] G. Fan, J. Wang, Y. Li, et al., Table discovery in data lakes: State-of-the-art and future directions, in: SIGMOD/PODS, 2023.
[15] R. Castro Fernandez, Z. Abedjan, F. Koko, et al., Aurum: A data discovery system, in: ICDE, 2018.
[16] S. Castelo, R. Rampin, A. S. R. Santos, et al., Auctus: A dataset search engine for data discovery and augmentation, Proc. VLDB Endow. 14 (2021) 2791–2794.
[17] N. W. Paton, J. Chen, Z. Wu, Dataset discovery and exploration: A survey, ACM Comput. Surv. 56 (2024) 102:1–102:37.
[18] A. Helal, M. Helali, K. Ammar, et al., A demonstration of kglac: A data discovery and enrichment platform for data science, Proc. VLDB Endow. 14 (2021) 2675–2678.
[19] Q. Wang, R. C. Fernandez, Solo: Data discovery using natural language questions via A self-supervised approach, Proc. ACM Manag. Data 1 (2023) 262:1–262:27.
[20] J. Freire, G. Fan, B. Feuer, et al., Large language models for data discovery and integration: Challenges and opportunities, IEEE Data Eng. Bull. 49 (2025) 3–31.
[21] Y. Dong, C. Xiao, T. Nozawa, et al., Deepjoin: Joinable table discovery with pre-trained language models, Proc. VLDB Endow. 16 (2023) 2458–2470.
[22] A. Khatiwada, H. Kokel, I. Abdelaziz, et al., Tabsketchfm: Sketch-based tabular representation learning for data discovery over data lakes, in: ICDE, 2025.
[23] H. Zhang, Y. Liu, W. Hung, et al., Autoddg: Automated dataset description generation using large language models, CoRR abs/2502.01050 (2025).
[24] Y. Huang, Y. Gao, C. Ren, A survey of data augmentation in named entity recognition, Neurocomputing 651 (2025) 130856.
[25] B. Ding, C. Qin, R. Zhao, et al., Data augmentation using large language models: Data perspectives, learning paradigms and challenges, 2024. arXiv:2403.02990.
[26] W. Wei, X. Ren, J. Tang, et al., Llmrec: Large language models with graph augmentation for recommendation, 2024. arXiv:2311.00423.
[27] Z. Ji, M. Jiang, A systematic review of electricity demand for large language models: evaluations, challenges, and solutions, Renewable and Sustainable Energy Reviews 225 (2026) None.
[28] M. F. Argerich, M. Patiño-Martínez, Measuring and improving the energy efficiency of large language models inference, IEEE Access 12 (2024) 80194–80207.
[29] M. J. Q. Zhang, E. Choi, Clarify when necessary: Resolving ambiguity through interaction with lms, in: Findings of the Association for Computational Linguistics: NAACL 2025, Association for Computational Linguistics, 2025, pp. 5526–5543.
[30] Z. He, S. Naphade, T. K. Huang, Prompting in the dark: Assessing human performance in prompt engineering for data labeling when gold labels are absent, in: CHI, ACM, 2025, pp. 1195:1–1195:33.
[31] K. Trinh, S. Seidenberger, R. Wijewickrama, et al., A picture is worth a thousand prompts? efficacy of iterative human-driven prompt refinement in image regeneration tasks, in: IJCAI, ijcai.org, 2025, pp. 10189–10197.
[32] A. Dutulescu, S. Ruseti, M. Dascalu, et al., How hard can this question be? an exploratory analysis

of features assessing question difficulty using llms, in: EDM, 2024.
[33] M. Gabburo, N. P. Jedema, S. Garg, et al., Measuring retrieval complexity in question answering systems, in: Findings of the ACL, Association for Computational Linguistics, 2024, pp. 14636–14650.
[34] T. Kwa, B. West, J. Becker, et al., Measuring AI ability to complete long tasks, CoRR abs/2503.14499 (2025). arXiv:2503.14499.
[35] Y. Zhu, D. Liu, Z. Lin, et al., The LLM already knows: Estimating llm-perceived question difficulty via hidden representations, CoRR abs/2509.12886 (2025). arXiv:2509.12886.