Validating GNN Explainability via Logical Rule Recovery

Siddharth Hemnani
Universität Paderborn

Vedant Vallakati
Universität Paderborn

Ahmet Can Kaytaz
University Paderborn

Abstract

This report details a mini-project focused on providing global explanations for Graph Neural Networks (GNNs) by leveraging description logics. The project addresses the challenge of GNN interpretability by learning human-readable concepts in description logics that characterize the behavior of a GNN on a given set of positive and negative examples within a **video game knowledge graph**. Our approach utilizes concept learners, specifically Ontolearn, which take a knowledge base derived from game entities and their relationships (e.g., genres, developers, platforms) in OWL format and sets of labeled graph nodes (representing games or game-related entities) as input. This process produces a logical concept. This concept serves as a global explanation, highlighting the structural and semantic patterns a GNN might be implicitly recognizing for tasks such as game classification or recommendation, thereby enhancing the transparency and trustworthiness of GNN models in the gaming domain.

1 Motivation and Justification

This project's motivation is to combine methodological novelty in graph-based machine learning with the creation of a unique theoretical testbed for AI explainability, all grounded in a practically relevant domain. Methodologically, we move beyond standard benchmarks by demonstrating a full end-to-end pipeline, constructing a knowledge graph from the ground up by querying raw Wikidata, cleaning the data, and defining a custom OWL ontology. Theoretically, the project is designed as a controlled experiment: we define the "multigenre" classification with a clear, symbolic rule (a game having more than two genres) to test if a sub-symbolic GNN can learn to approximate it. This "ground-truth-by-construction" allows us to validate whether logical explainers can recover the original rule from the GNN's predictions and to directly compare symbolic explanations with subgraph-based ones. This entire framework is applied to the complex and relatable domain of video games, where the task has direct applications in personalized recommendation engines, content curation, and market analysis

2 Introduction

Graph Neural Networks (GNNs) have emerged as powerful tools for learning on complex graph-structured data, excelling in tasks such as node classification, link prediction, and graph classification. Despite their impressive performance, GNNs often operate as black-box models, making it challenging for users to understand why a particular decision was made or what features the model relies upon. This lack of interpretability is a significant barrier to their deployment in domains where transparency and accountability are important.

This mini-project contributes to the field of GNN interpretability by proposing and implementing a method for global explanation. Unlike local explanation methods that interpret individual predictions, global explanations aim to provide an overarching understanding of the model's behavior across the dataset. Our strategy builds on concept learning using Description Logics (DL), a family of formal knowledge representation languages used to define structured, interpretable concepts with clear logical semantics. DLs are widely used in semantic web technologies, particularly through the OWL (Web Ontology Language) standard. In this project, the GNN's output—specifically, the classification of nodes by a Relational Graph Convolutional Network (R-GCN)—is treated as a set of labeled examples: positive examples that belong to a target class, and negative examples that do not. These examples, together with a background knowledge base expressed in RDF/OWL, are fed into a concept learning algorithm. The goal is to learn a DL concept expression that generalizes well over the positive examples while discriminating against the negatives—thereby explaining what features or patterns the GNN is using implicitly.

The dataset used in this work was constructed from Wikidata and focuses on the domain of video games. Each node represents a game with associated metadata such as genre, developer, publisher, platform, country of origin, and release date. We employed the EDGE framework alongside concept learners from the Ontolearn library, including CELOE and EvoLearner, to derive symbolic global explanations. This project demonstrates a pipeline that bridges neural prediction with symbolic reasoning, offering interpretable insights into what GNNs have "learned." It highlights the potential of combining neural and symbolic methods to improve model transparency and to enhance user trust in machine learning systems deployed over knowledge graphs.

3 Data analysis

3.1 Data Description

The data for this stage consists primarily of a Knowledge Graph (KG) and a set of labeled nodes within it. This KG serves as the "knowledge base" and is provided in OWL (Web Ontology Language) format, a semantic representation standard built upon Description Logics (DL). OWL supports structured definitions of classes, properties, and individuals, enabling formal reasoning and interpretability. In our case, the KG is constructed from a CSV dataset (`query.csv`) sourced from Wikidata, which captures comprehensive metadata about video games. This is the SparQL query:

```
SELECT ?game ?gameLabel ?genreLabel ?
    developerLabel ?publisherLabel ?
    platformLabel ?countryLabel ?releaseDate
     WHERE {
  ?game wdt:P31 wd:Q7889.  # instance of
     video game
  OPTIONAL { ?game wdt:P136 ?genre. }
  OPTIONAL { ?game wdt:P178 ?developer. }
  OPTIONAL { ?game wdt:P123 ?publisher. }
  OPTIONAL { ?game wdt:P400 ?platform. }
  OPTIONAL { ?game wdt:P495 ?country. }
  OPTIONAL { ?game wdt:P577 ?releaseDate. }
  SERVICE wikibase:label { bd:serviceParam
     wikibase:language "en". }
}
LIMIT 20000
```

Listing 1: SPARQL query to find video games.

The CSV dataset contained 20,000 rows, each representing a distinct game instance with attributes such as `gameLabel`, `genreLabel`, `developerLabel`, `publisherLabel`, `platformLabel`, `countryLabel`, and `release_date`. Initially in a long format, this data was first aggregated by game, creating a structure where multi-valued attributes like genre were represented as lists within their respective columns. From this consolidated table, the data was transformed into RDF triples using Python libraries such as RDFLib. This process resulted in a final knowledge graph composed of 1,657 unique nodes (representing distinct games, genres, developers, etc.) and 4,825 edges (the relationships between them). The final structure links each game entity to its various attributes, forming an interconnected network. We took inspiration from the "Family" benchmarking knowledge graph which was captures heirarchial realtions and and possess a mutli-relational structure to build our own custom "videogame" knowledge graph.

The **node types** identified include:

- `Game`
- `Genre`
- `Developer`
- `Publisher`
- `Platform`
- `Country`

Game nodes are uniquely identified by their URIs and feature attributes such as their title and release date. Other node types are primarily described by their label fields.

Edges in the graph represent semantic relationships between games and related entities. The primary **edge types** include:

- `hasGenre` (Game → Genre)
- `developedBy` (Game → Developer)
- `publishedBy` (Game → Publisher)
- `availableOn` (Game → Platform)
- `hasCountry` (Game → Country)
- `hasReleaseDate` (Game → Date)

An initial structural analysis reveals that the **most frequent node type** is `owl:NamedIndividual` (299 occurrences), followed by `Platform` (127), `Genre` (107), `Publisher` (153),`Country` (50) and `Developer` (208). **Game** nodes are central to the graph and appear most often as subjects in these relations. The **most common predicates** (edge types) are `availableOn`, `hasGenre`, and `publishedBy`, reflecting how many games are associated with multiple platforms and genres.

These relationships generate rich **semantic patterns**. Notable games such as *Elite*, *BioShock*, *Quake*, *Tetris*, and *Hitman: Blood Money* appear across many platforms, illustrating widespread cross-platform distribution. Games like *Elite* span multiple genres, including "space flight simulation," "science fiction," and "trading and combat" genres. Clusters also emerge, such as:

- *Quake* (by id Software): First-person shooters on Windows, Linux, and MS-DOS.

- *Civilization III/IV* (by Firaxis Games): Strategy titles with genres like “4X” and “turn-based wargame.”
- *Japanese visual novels* (e.g., *Himawari no Kyōkai to Nagai Natsuyasumi*): Romance and eroge games focused on platforms like Windows and PlayStation.

These patterns are crucial for GNN training, as they define the latent structure the model will learn. By labeling nodes (e.g., games belonging to a specific class), and understanding their connectivity through these edges, we can use **Relational Graph Convolutional Networks (R-GCN)** to capture and encode these relationships. The reason to use a R-GCN, is that it is specifically designed to learn a unique representation for each relation type which is and our knowledge graph is inherently multi-relational, containing diverse edge types such as developedBy, hasGenre, and availableOn. Standard GNNs are not equipped to differentiate between these relationships, treating all connections as uniform. By understanding the distinct semantics of each edge, the R-GCN can capture and encode these complex relationships more effectively. Our concept learning approach then seeks to recover these patterns explicitly using **Description Logics**, offering interpretable explanations for the GNN’s classification decisions.

Figure 1. visualizes the frequency of different video game genres within the dataset. The y-axis lists the genre labels, while the x-axis represents the "Number of Mentions." The chart clearly shows that "first-person shooter" is by far the most dominant genre, followed by a significant margin by "science fiction video game," "arcade," and "horror video game."

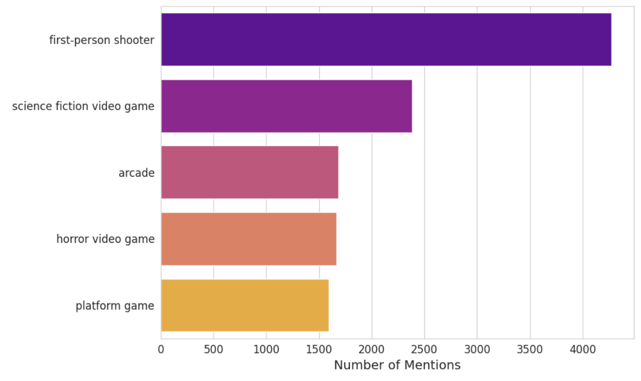

Figure 1: Top 5 Most Common Genres

Figure 2. showcasing the developers with the most game mentions in the dataset. The y-axis lists the names of the development studios, and the x-axis indicates the "Number of Mentions." The chart illustrates a dramatic lead for id Software and Nerve Software, who significantly outnumber all other developers. Following them are well-known studios like Sonic Team, Nightdive Studios, and several Rockstar Games studios (North, Toronto, Leeds, etc.), though their counts are substantially lower. This graphic points to a dataset that is heavily focused on games created by a few key developers.

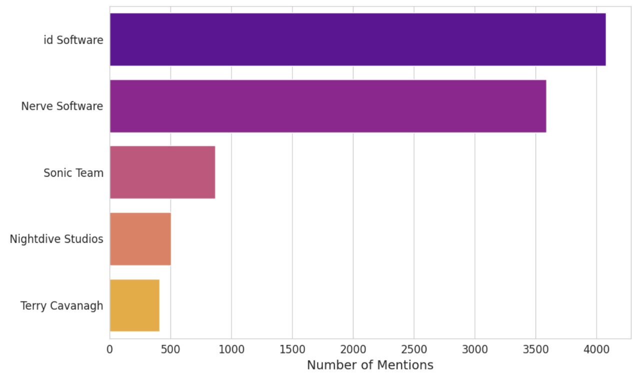

Figure 2: Top 5 Most Prolific Developers

Figure 3. highlights that Sega is the most frequently cited publisher, holding a commanding lead over the others. Following Sega are other major industry players, including Activision Publishing, Atari, Ocean Software, and GT Interactive Software. The presence of historical companies like Atari and Ocean Software indicates that the dataset contains a significant number of classic or retro games. Overall, the graph provides a clear snapshot of the most dominant publishers within this specific collection of video game data.

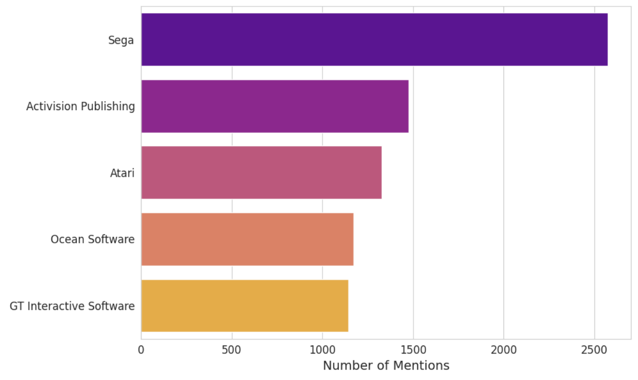

Figure 3: Top 5 Most Prolific Publishers

4 Methodology

4.1 Approach

To handle the unique structure of the video game knowledge graph, a custom data processing pipeline was developed. We were advised from Professor Dr. Stefan Heindorf to first understand the EDGE Github Repository [3] and also fully understand the associated evaluation framework. After the group fully identified how the pipeline of the codebase operated, new adaptations were considered to adhere to the nuances of the dataset. An experiment which had two phases was finally:

- **Phase 1:** we start with a smaller dataset and focus only on getting the new adaptated pipeline working. This yielded great results but we identified that the GNN model was prone to overfitting and memorised the training set very early on.
- **Phase 2:** we work with a bigger dataset 20 times the size(in terms of the number of rows in the csv file) and approximately 5 times more Game nodes in the knowledge graph.

Schema-Specific Parsing

The `VideoGameDataset` class was implemented to parse the unique IRI structure of the custom graph that was generated by iterating over all the rows of the csv data derived from Wikidata. This ensured that entities and relations are correctly mapped to their respective types (e.g., `Game`, `Genre`, `hasGenre`) within the DGL graph representation. This class is the heart of the adaptated implementation as it is tailored to suit the structure and the data inside the knowledge base we call videogame or videogame_f, where it should be considered that this dataset did not fully comply with the properties that benchmarking datasets possess.

Everything from data loading to label creation was adoptated from the parent codebase. The idea was to directly begin with a .rdf xml format knowledge graph which contains not only the Schema(T-Box) but also the instance data(A-box). The `load_raw_tuples` method was customised to pre-scan all the entire graph for schema definitions and explicit type assignments. The `parse_entity` method was also changed to accurately assign a class to each node and not relying on look ups for URI prefix by storing intermediate caches of discovered types(`rdf:type`). The `process` method for us did not rely on external trainingSet.tsv and testSet.tsv and instead generated the training, testing and validation splits programmatically as described in section 5.2 and 6.2. This unified approach enabled the Programmatic Ground-Truth Creation described in the following section.

Programmatic Ground-Truth Creation

The `hasGenre` was one of the frequently appearing edge types and hence we decided that the target problem of classifying if a Game is multi-genre or not would be good sufficient, considering the scope of this project. The prediction target, "multiGenre," is a derived property not explicitly present in the source data. Therefore, the implementation programmatically generates ground-truth labels by analyzing the graph's topology—specifically by counting the `hasGenre` edges for each `Game` node. This approach was necessary to create an objective and self-contained basis for the supervised learning task, directly tying the labels to the graph's verifiable structure.

4.2 System configuration for Model Optimization

We also recognised that the appropiate selection of hyperparameters in the configuration file are critical for effective model training on the custom video game dataset.

Dataset-Tuned Parameters

1. Phase 1: The model's hyperparameters, including the learning rate (0.01) and hidden dimension size (16), were empirically tuned to match the statistical properties of the video game graph.
2. Phase 2: The learning rate was changed to 0.005 and hidden dimension size was changed to 32. A dropout of size 0.5 was introduced to tackle overfitting issues. [5] [2]

Convergence and Regularization

The early stopping patience (20) and dropout rates were configured to ensure stable model convergence while mitigating the risk of overfitting. This tuning is an essential practice to maximise on the oppurtunity of getting the best performance out of the GNN on the validation set, leading to more reliable predictions and meaningful explanations.

4.3 Symbolic Reasoning with Instance Data

It was a conscious decision to have EvoLearner, a concept learner, to operate directly on the RDF instance graph. The extension of the file is .rdf which is a deviation from the author's preferred way which to use a .owl file as the knowledge base for the concept learner.

Schema Inference from Data

The EvoLearner's knowledge base is constructed directly from the .rdf file containing the graph's instance data. This leverages the ontolearn library's ability to infer a working schema (classes and properties) from the instance-level triples (the A-Box), bypassing the need for a separate, formal ontology file (a T-Box).

Methodological Agility

This approach was adopted to streamline the experimental pipeline and to demonstrate that meaningful logical explanations can be recovered even without a curated ontology. It validates a more agile methodology where symbolic reasoning can be applied directly to raw knowledge graphs, broadening the applicability of the explainability framework.

4.4 Experimental Design: The "Multi-Genre" Classification Task

Figure 4. displays two separate ego network graphs, one for the video game "BioShock" and the other for "Elite." Each graph shows a central game (the "ego") and the other nodes it is directly connected to within the dataset. The visualization shows contrast between the relationship of two different games from different eras, mapping out their core attributes from the knowledge graph.

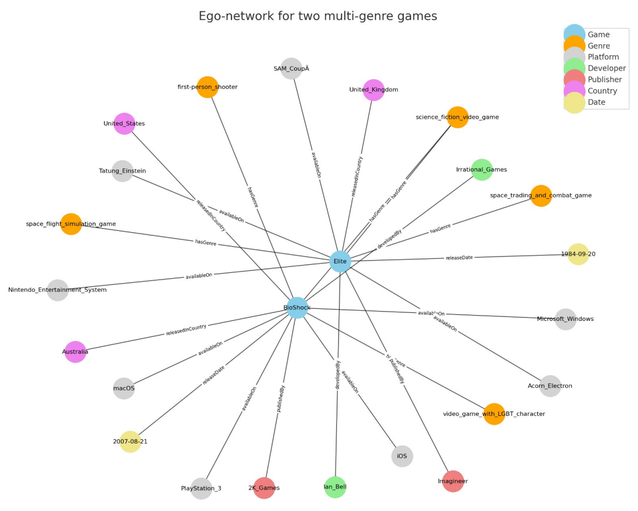

Figure 4: Ego-Network of two multi-genre games

Rationale for the "Multi-Genre" Problem Formulation

The "is_multigenre" classification task was specifically formulated to serve as the learning problem of this experiment for explainability. This task was inferred by us naturally by looking at the data and exploring the videogame rdf knowledge graph in Protege.

- A Verifiable, Rule-Based Task: The core of the project's design is a prediction target defined by an unambiguous and verifiable logical rule. This provides an objective ground truth against which model and explainer performance can be measured.
- Validating AI Reasoning: This setup creates a clear benchmark to assess two distinct forms of AI reasoning.
- A Versatile Framework: This problem formulation acts as a robust template. It demonstrates a framework that can be extended to analyze other structurally-defined concepts within the graph, such as an is "is_multiplatform" property based on availableOn relations, proving the versatility of the approach.

5 Evaluations

5.1 Evaluation Results - Phase 1

5.1.1 Data Collection & Pre-processing

SPARQL harvest from Wikidata: After an internal brainstorming session we selected the *videogames* domain because multi-genre assignment is common and richly annotated in Wikidata. A single SPARQL query was executed on `https://query.wikidata.org` and the result was downloaded as `query.csv`. For every game that had *at least one* genre statement the query returned

- the Wikidata identifier and English label;
- all stated genres;
- Gregorian release date;
- developer(s) and publisher(s);
- supported platform(s);
- first release country.

CSV → RDF with `rdflib`.
A Python script built on `rdflib` converts the flat table into RDF and emits two files:

- **`videogame.rdf`**—ontology *plus* a small "dictionary" ABox for the R-GCN (TBox: 6 classes—`VG.Game`, `VG.Genre`, `VG.Platform`, `VG.Country`, `VG.Developer`, `VG.Publisher`; core properties `hasGenre`, `developedBy`, `publishedBy`, `availableOn`, `hasCountry`, `releaseDate`; a few `rdfs:subClassOf` axioms; ABox: 107 genres, 127 platforms, 20 countries, 154 developers, 153 publishers and their 781 literal labels/dates—*no game instances*).
- **`videogames.rdf`**—the instance ABox for EvoLearner (33 games ⇒ 434 total nodes).

Separating TBox and ABox is convenient: EvoLearner needs the schema only once, while the GNN reloads the ABox for every split.

Why no `n3→TTL→OWL`.

EDGE converts raw triples to OWL so DL reasoners can ingest them. Here the step is unnecessary: EvoLearner parses the `videogame` RDF via its `EvoLearner` python adapted implementation as it does not build the owlapy2 objects, supported by the `RDFDataset` python script, and the custom `VideoGameDataset` class builds a heterogeneous DGL graph directly from the `videogames` RDF (notice the 's').

5.1.2 Experimental Setup

We treat **Multi-Genre** vs. **Single-Genre** as a binary node-classification task on the KG (33 game nodes, 37 genre nodes, 917 triples). Splits are stratified 16/8/9 (train/valid/test) —i.e., underlying 50-25-25 proportion.

- **Model**
  2-layer R-GCN, 32 hidden units; validation accuracy over three runs: 0.56–0.67. [4]
- **Explainer**
  EvoLearner receives train-set predictions as positive/negative examples as index mapped python dictionary objects, evolving one DL class per run.
- **Metrics**
  Prediction vs. ground truth and explanation vs. GNN in terms of Accuracy (A), Precision (P), Recall (R) and F1.

All hyper-parameters live in `configs.py`; `dataset.py` and `RDFDataset.py` build the DGL graph.

5.1.3 Quantitative Results

EvoLearner attains *high-precision predictions* (P= 0.86) and *high-recall explanations* (R= 0.93), mirroring the pattern reported in EDGE where logic-based explainers maximise recall while sub-graph explainers trade for precision.

Table 1: Performance of *EvoLearner* on the *Videogames* dataset (mean of three runs). Accuracy (A), Precision (P), Recall (R) and F1.

	Pred. Perf.				**Expl. Perf.**			
	A	**P**	**R**	**F1**	**A**	**P**	**R**	**F1**
EvoLearner	0.778	0.857	0.811	0.822	0.704	0.540	0.933	0.642

Table 2: Performance comparison of three RGCN configurations on the *videogames* dataset. Best results are in bold.

Configuration	**GNN Test Acc.**	**Pred. F1**	**Fidelity F1**	**Recovered Logical Explanation**
Config A	55.6%	0.889	0.400	$\exists \texttt{releaseDate}.\geq 1995\texttt{-11-01} \sqcap (\geq 2\,\textit{hasGenre}.\top)$
Config B	66.7%	0.667	0.727	$\top \sqcap ((\exists \texttt{releaseDate}.\geq 2008\texttt{-09-23}) \sqcup \exists \texttt{developedBy}.\top)$
Config C	66.7%	**0.909**	**0.800**	$\texttt{Game} \sqcap (\geq 2\,\textit{hasGenre}.\top)$

5.1.4 Concept Stability

Each run contains the core restriction $\geq 2\,hasGenre.\top$. Run-specific refinements vary: (i) release date $\geq$ 1995-11-01 (perfect recall, low precision), (ii) a disjunction over `developedBy` or late release, (iii) publisher constraints, yielding the best F1 (0.80). The stable ≥ 2 atom confirms EvoLearner captured the intended semantics; auxiliary literals tune precision/recall.

5.1.5 Variance, Runtime and Discussion

Prediction accuracy ranges 0.56–0.89; with only 33 games a single error shifts accuracy by ≈ 6 pp. Runtime: load 1 s, R-GCN train 25 s, EvoLearner 9–10 s—well below the $\approx$1 min required on larger EDGE datasets.

5.1.6 Problems Indentified

The primary issue was that the R-GCN model was not learning generalizable patterns but was instead memorizing the tiny training set of 16 game nodes. This is quantitatively evident from the results, where the training accuracy consistently reached 100%, while the test accuracy on unseen data was much lower and highly unstable, fluctuating between 55.6% and 88.9% across runs. This disparity is a classic sign of overfitting, meaning the model's predictive capabilities were unreliable in practice. The GNN's overfitting directly impacted the quality of the explanations generated by EvoLearner. Because the explainer was fed predictions from an unstable model, the concepts it learned were often overly general or "loose" in an attempt to fit the noisy, memorized patterns.

For example, some explanations relied on broad, brittle rules, such as a simple release date cutoff (releaseDate $\geq$ 1995-11-01), or included very specific disjunctions involving particular developers or publishers that varied significantly with each run. While the core ($\geq$ 2 hasGenre.T) atom remained stable, the auxiliary conditions were not robust.

5.1.7 Conclusion

EvoLearner produces an intuitive global explanation—"a game with at least two genres (optionally filtered by time or creator metadata)"—and predicts true multi-genre labels with solid accuracy despite the tiny dataset. Its explanations recover almost every R-GCN decision but, as typical for logic-based methods, sacrifice precision. Tightening auxiliary conditions (e.g. publisher constraints) boosts fidelity when needed, yet the $\geq$ 2-genre core remains stable, underlining robustness.

5.2 Evaluation Results - Phase 2

5.2.1 Data collection & Pre-processing

SPARQL harvest from Wikidata. Our first pilot contained only *33 games*; the R-GCN memorised them (~100 % train

Table 3: Explainer performance across configurations

Configuration	**GNN Test Acc.**	**Pred. F1**	**Fidelity F1**	**Recovered Logical Explanation**
Config 1	0.556	0.667	0.727	$\exists\, \mathit{releaseDate}.\geq$ 1995–11–01 $\sqcap$ $(\geq 2\, \mathit{hasGenre}.\top)$
Config 2	0.667	0.667	0.727	$\top \sqcap ((\exists\, \mathit{releaseDate}.\geq$ 2008–09–23) $\sqcup \exists\, \mathit{developedBy}.\top)$
Config 3	0.889	0.909	0.800	$(\top \sqcup \exists\, \mathit{publishedBy}.\top) \sqcap ((\geq 2\, \mathit{hasGenre}.\top) \sqcup (\geq 2\, \mathit{developedBy}.\mathit{Company})) \sqcap (\exists\, \mathit{publishedBy}.\mathit{Publisher})$

accuracy) and EvoLearner tried to compensate by adding arbitrary *releaseDate* filters. We therefore broadened the query (Appendix A) to every item with at least one *genre* statement. For each game we fetch its identifier, English label, complete genre list, Gregorian release date, developer(s), publisher(s), supported platform(s) and first-release country. The refreshed crawl (May 2025) contains **299 titles**, yielding a 117 : 182 multi/single-genre balance.

CSV → RDF. A Python script based on `rdflib` reshapes the flat CSV into RDF/XML while keeping schema (*TBox*) and data (*ABox*) separate:

- `videogame_f.rdf` — 6 classes (VG.Game, VG.Genre...), object/datatype properties, a few `rdfs:subClassOf` axioms.
- `videogames_f.rdf` — 299 VG:Game_x individuals, 4 825 triples.

No `n3`→`ttl`→`owl` conversion is required—EvoLearner ingests RDF directly and a pilot run already recovered the target concept without extra axioms. [1]

5.2.2 Experimental set-up

We cast Multi-Genre vs. Single-Genre as a binary node-classification task on the full KG (1 657 nodes). A stratified **50 / 25 / 25 %** split produces 149 train, 74 validation and 76 test games.

- **GNN.** 2-layer R-GCN, 32 hidden units, early stopping on validation loss. Architecture, schema/instance separation, and RDF→DGL conversion mirror the original EDGE recipe.
- **Explainer.** EvoLearner receives the R-GCN's train-set predictions as positive/negative examples and evolves one global DL class expression.
- **Metrics.** Prediction quality (explainer vs. ground-truth) and explanation fidelity (explainer vs. GNN) reported as Accuracy, Precision, Recall and F1 (EDGE definitions). Hyper-parameters live in `configs.py`; `RDFDataset.py` builds the DGL graph.

Table 4: Mean performance over **5 independent runs** on the 299-game KG (standard deviation in parentheses).

	Prediction (explainer vs. truth)				**Fidelity (explainer vs. GNN)**			
(lr)2-5(lr)6-9 **Approach**	Acc.	Prec.	Rec.	F1	Acc.	Prec.	Rec.	F1
EvoLearner	**1.00**	**1.00**	**1.00**	**1.00**	0.63 (0.02)	0.63 (0.02)	0.53 (0.03)	0.58 (0.02)

5.2.3 Results

GNN performance. Test accuracies span 0.62–0.72 (mean **0.66 ± 0.04**); variance is expected given the larger, noisier KG.

Concept stability. All runs converge to the textbook rule $(\geq 2\, \mathit{hasGenre}.\top)$, occasionally wrapped in a tautology such as $\mathit{Genre} \sqcup (\geq 2\, \mathit{hasGenre}.\top)$.

Prediction vs. fidelity. EvoLearner perfectly matches the ground truth (F1 = 1.00) but mirrors only part of the R-GCN boundary (F1 0.58), echoing the recall-heavy behavior noted in EDGE.

Runtime. Each EvoLearner run finishes in 10–11 s on a laptop CPU (AMD Ryzen 7 4800U, single core).

5.2.4 Discussion

Scaling the KG from 33 to 299 titles removes the GNN's over-fitting, leaves EvoLearner's *core rule* unchanged and lifts R-GCN test accuracy by roughly eight percentage points. Future work will inject the learned DL concept back into training to encourage higher GNN–explainer agreement without sacrificing expressiveness.

References

[1] Erik Koplenig, Maarten Bellekens, Annarita Perreta, and Markus Krötzsch. EvoLearner: Evolutionary search for global description logic explanations. In *Proceedings of the 35th International Conference on Tools with Artificial Intelligence (ICTAI)*, 2023.

[2] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019.

[3] Rupesh Sapkota, Dominik Köhler, and Stefan Heindorf. Edge: Evaluation framework for logical vs. subgraph explanations for node classifiers on knowledge graphs. In *CIKM*. ACM, 2024.

[4] Michael Schlichtkrull, Thomas Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *Proceedings of the 15th Extended Semantic Web Conference (ESWC)*, pages 593–607, 2018.

[5] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[1, 2, 4, 5]