VizML x KG4Vis

Final Report of the Capstone Project

João Miguel Peixoto Lamas
Pedro Afonso Nunes Fernandes
Tiago de Pinho Bastos de Oliveira Pinheiro
Tiago Grilo Ribeiro Rocha

Bachelor in Informatics and Computer Engineering

Supervised by: Alexandre Miguel Barbosa Valle de Carvalho

June 27, 2025

Contents

1 Glossary

- **KG4Vis** - Knowledge Graph for Visualization Recommendation.
- **VizML** - Visualization using Machine Learning.
- **CUDA** - Compute Unified Device Architecture.
- **TSV** - Tab-Separated Values.
- **CSV** - Comma-Separated Values.
- **GPU** - Graphics Processing Unit.
- **CPU** - Central Processing Unit.
- **RL** - Reinforcement Learning.

2 Concepts

Activation function: A function used in neural networks to introduce nonlinearity into the model. Common examples include ReLU, Sigmoid, and Softmax.

Adam optimizer: An adaptive learning rate optimization algorithm that combines the advantages of AdaGrad and RMSProp. Widely used for training deep learning models.

AutoGrad: PyTorch's automatic differentiation engine, which computes gradients for optimization during backpropagation.

Batch size: The number of training samples processed in one forward/backward pass during model training. Larger batches offer more stable gradient estimates, but consume more memory.

DataLoader: A PyTorch utility that provides an efficient way to load data in mini-batches and shuffle them during training.

Distributed computing: A computing model where multiple computers (or nodes) work together over a network to solve a large problem.

Embedding: A representation of complex data as a vector of numbers in a continuous space, so that similar things have similar vectors.

Feedforward neural network: A type of neural network in which information moves in only one direction, from input to output, without cycles or loops.

Graph partitioning: The process of dividing a graph into smaller parts that have roughly the same number of nodes, with the objective of minimizing the number of edges between parts.

Knowledge graph: A structured way to represent information using a graph of real-world entities and their relationships, where the nodes represent the entities and the edges represent the relationships.

Learning rate scheduling: A technique for dynamically adjusting the learning rate during training, often to improve convergence and performance.

Loss function: A function that measures how far off the network's predictions are from the actual labels. Common examples include cross-entropy and mean squared error.

Mean Rank (MR): A ranking-based evaluation metric that represents the average rank position of the correct visualization type in a list of recommendations. It measures how well the model ranks correct options—performance is better when the MR is lower.

Mini-batch training: A training approach where a subset (batch) of the dataset is used to update the model weights rather than using the entire dataset.

Negative sampling: A training technique used in knowledge graph embeddings to generate incorrect (negative) examples, helping the model distinguish valid from invalid relationships.

Neural network: A machine learning model that recognizes patterns and makes predictions by adjusting connections between layers of artificial ”neurons.”

Neural network topology: The way that neurons are connected in a neural network, defining the number of layers and neurons, how they are connected, and what types of layers are used.

PyTorch: An open-source machine learning framework based on Python and Torch that is widely used for developing deep learning applications.

ReLU (Rectified Linear Unit): A popular activation function defined as $f(x) = \max(0, x)$, used to introduce non-linearity into neural networks while avoiding the vanishing gradient problem.

Simple classifiers: Less complex machine learning models that assign labels to data using basic decision rules or statistical techniques.

Softmax: An activation function that converts a vector of values into a probability distribution. Often used in the output layer of classification tasks.

Torch variables: Multi-dimensional arrays used in PyTorch that support operations like matrix multiplication, addition, reshaping, etc. They can reside in CPU or GPU memory for efficient computation.

TransE embedding: A knowledge graph embedding model that represents entities and relationships as vectors in a continuous space. It allows KG4Vis to infer new relationships and make explainable recommendations.

3 Introduction

This report presents the development and analysis of a capstone project focused on VizML and KG4Vis, two different approaches for visualization recommendation. The project was developed in an academic setting, with the goal of exploring and comparing these two methodologies to recommend data visualizations.

The main motivation was to analyze and compare these methods and their efficiency in recommending visual encodings.

3.1 Objectives and expected results

The primary objectives of this project were to analyze and compare the VizML and KG4Vis approaches for visualization recommendation, to understand the strengths and limitations of each method, particularly in terms of accuracy, explainability, scalability, and computational costs, and to successfully run both models using the provided dataset and evaluate their performance.

The expected results include a detailed comparison of the two approaches, insights into their computational requirements, and a working implementation of both models.

3.2 Report structure

This report is structured as follows:

- **Methodology and Development Process**: Describes the iterative approach, stakeholder roles, and timeline of activities.
- **Solution Development**:
  - *Approach*: Compares VizML (neural network-based) and KG4Vis (knowledge graph-based) methodologies.
  - *Technical Implementation*: Details neural network training (VizML) and embedding learning (KG4Vis).
  - *Visualization Recommendation Process*: Contrasts direct prediction (VizML) and explanation-based recommendations (KG4Vis).
  - *Pre-Execution Comparative Analysis*: Evaluates technology stacks and core differences.
  - *Dataset*: Describes the dataset structure and preprocessing steps.
  - *Training*: Covers feature extraction and model training for both approaches.
  - *Tests*: Summarizes the challenges with VizML and KG4Vis testing configurations.
  - *Comparative Analysis*: Discusses performance, scalability, accuracy, and precision metrics.

- **Conclusions**: Summarizes the results achieved, lessons learned, and future work.
- **Appendix**: Includes meeting logs and command lists for reproducibility.
- **References**: List of cited works and resources.

4 Methodology and Development Process

4.1 Methodology used

The project followed an iterative development approach, with weekly meetings to discuss progress, analyze findings, evaluate the results, and plan next steps. Thanks to this process, as the project developed, we were able to get feedback and make the required changes.

The key resources used during the project included GitHub for report collaboration, PyTorch for neural network training, and TransE embeddings for the KG4Vis model. The team also relied on the original papers for VizML and KG4Vis to guide the implementation and analysis.

4.2 Stakeholders, roles and responsibilities

There were several participants in this project, and each had different roles and responsibilities contributing to the project's success:

- **Project Coordinator and Tutor** - Professor at Faculdade de Engenharia da Universidade do Porto Alexandre Miguel Barbosa Valle de Carvalho, responsible for guiding the project, providing feedback, and evaluating the final results.
- **Team Members and Distribution of Work**

  - João Miguel Peixoto Lamas (up202208948): Initial research, information gathering, model testing, and analysis of results (25%).

  - Pedro Afonso Nunes Fernandes (up202207987): Initial research, information gathering, and subsequent refinement of the collected information to build the report (25%).

  - Tiago de Pinho Bastos de Oliveira Pinheiro (up202207890): Initial research, information gathering, and subsequent refinement of the collected information to build the report (25%).

  - Tiago Grilo Ribeiro Rocha (up202206232): Initial research, information gathering, model testing, and analysis of results (25%).

4.3 Activities developed

Table 1: Activities developed during the project timeline.

Timeline	**Activities**
13/02/25	Initial discussion of the VizML paper and verification of the dataset.
20/02/25	Analysis of the VizML and KG4Vis papers, making explanatory diagrams for each one, dataset characterization, and software installation.
27/02/25	Exploration of neural network training and embedding learning, start running the dataset.
06/03/25	Deep dive into scalability challenges, computational costs, and model training preparation.
13/03/25	Researched graph storage methods, explored training tech, calculated computational costs, corrected prior errors, detailed VizML-KG4Vis pipeline, added references, prepared dataset scripts, and initiated report development.
20/03/25	Dove deeper into the code, analyzed neural network topology and deep learning model, explored Torch variables and data loaders, implemented simple classifiers, documented dataset file types and execution, and reviewed the report structure.
03/04/25	Filled initial report sections, refined index, restructured to prioritize "Concepts," added comparative analysis of papers/code (performance/scalability post-execution), included metrics (accuracy, precision), and pre/post-execution comparisons. Concluded with work distribution.
08/04/25	Restructured report to discuss approach before dataset, reordered sections, added centered image captions. Explored applying functions with AI, documented issues with functions, noting attempted fixes.
24/04/25	After multiple attempts to adjust parameters and test VizML, concluded it wasn't yielding meaningful results and shifted focus to running the original KG4Vis code as implemented by the authors.
07/05/25	Filled missing report topics and tested KG4Vis code for different dimensions, changing some testing parameters (inferior batch size, hidden dimension).
15/05/25	Finalized testing and documentation, prepared for project submission.

5 Solution development

5.1 Approach

According to [1] and [2]:

- VizML is a machine learning approach that uses deep neural networks to recommend visualizations. It processes dataset features and predicts the most suitable chart type through a fully connected feedforward neural network. The system is trained on dataset-visualization pairs focusing on accuracy and scalability.
- KG4Vis is a knowledge graph-based approach that models relationships between dataset features and visualization choices using TransE embeddings. It emphasizes explainability and flexibility, allowing for dynamic expansion of the knowledge graph without retraining. The method builds on the VizML dataset but structures the data as a graph for inference-based recommendations.

All this is illustrated in Fig. 1.

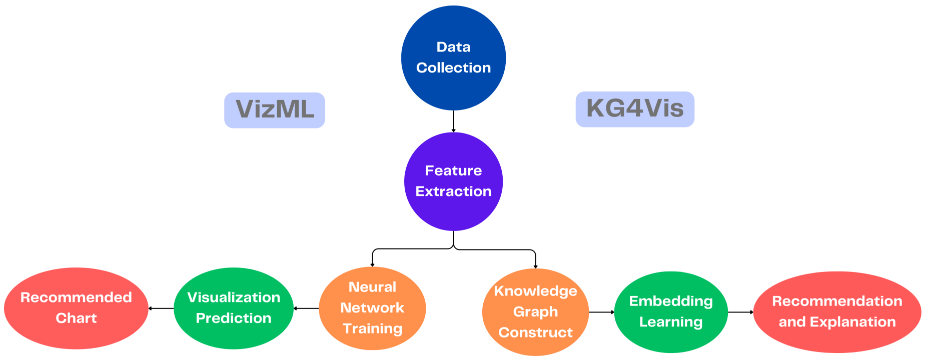

Figure 1: Pipeline of VizML and KG4Vis.

- Both models rely on a large dataset of dataset-visualization pairs, and both typically get their data from Plotly (KG4Vis gets it from VizML which gets it from Plotly).

We also found that, when extracting, the goal is to quantify the characteristics of the dataset to help select the right representation, with VizML extracting numerical features, such as statistical properties (mean, variance, etc.) and structural attributes (number of columns, missing values, etc.), and KG4Vis encoding the features as nodes in a knowledge graph.

5.1.1 VizML

According to [1]:

- VizML uses a supervised deep learning model, which means that it learns from labeled dataset-visualization pairs. It processes 841 extracted features per dataset and gives them to a fully connected neural network. The goal is for the network to map dataset features to the best-fitting visualization type.
- As mentioned above, VizML makes use of PyTorch to turn dataset features into sensors, which then allows the network to learn by minimizing prediction error, by adjusting weights through backpropagation.
- This process is done in small batches to improve efficiency, but even so it is computationally expensive, often requiring larger RAM and VRAM capacity and GPU acceleration, for example, to handle large datasets.

We also found that once the model has finished training, it can predict the most suitable visualization type in real time using a dataset's feature vector in a single forward pass through the network, resulting in fast inference. This, however, reduces its explainability, as it quickly gives an answer but does not justify or elaborate it.

5.1.2 KG4Vis

According to [2]:

- KG4Vis builds knowledge graphs with the extracted features, with the nodes representing the dataset attributes, visualization types and encoding choices, and the edges defining meaningful relations, for example, categorical data being best represented by a bar chart ("Categorical data ——— Bar Char"). This makes it dynamically expandable, meaning it can receive and integrate new dataset features and/or visualization types without retraining and starting again from the beginning. It also helps with interpretability, since all relations are explicitly defined.
- KG4Vis learns vector representations (embeddings) for each node and relation in the knowledge graph. It uses TransE embeddings, which dictate that if A and B are related by R, then in vector space A + R ˜˜ B. Pytorch is used in this case to help train these embeddings, as mentioned before, by helping to compute and update the knowledge graphs, so the model can predict possible missing links from it. The embedding space preserves the relationships between data and visualization types, which helps explainability.

We also found that KG4Vis, instead of directly predicting, instead retrieves the most relevant visualization based on its learned knowledge graph structure post-training. Due to the nature of the graph, it can then trace back the recommendation, allowing it to explain why that particular visualization is suitable, offering better explainability.

5.2 Technical Implementation

5.2.1 Neural Network Training (VizML)

According to [1]:

- VizML employs a fully connected feedforward neural network (NN) as its primary model, implemented using the PyTorch framework. The network architecture consists of three hidden layers, each containing 1,000 neurons with Rectified Linear Unit (ReLU) activation functions. ReLU was selected because of its computational efficiency, ability to mitigate the vanishing gradient problem, and tendency to induce sparse activations, which can improve model generalization.

- The training process utilizes the Adam optimizer with a mini-batch size of 200 and an initial learning rate of 5×10^{-4}. To optimize convergence, a dynamic learning rate scheduling mechanism is employed: the learning rate is reduced by a factor of 10 whenever the validation accuracy fails to improve beyond a threshold of 10^{-3} over a span of 10 epochs. Training terminates either after the third learning rate reduction or upon reaching 100 epochs, whichever occurs first. Empirical evaluations indicated that additional regularization techniques, such as weight decay, dropout, and batch normalization, did not yield significant performance improvements, suggesting that the baseline architecture already achieves sufficient generalization without further constraints.

Neural Network Topology The network follows a sequential feedforward structure, where each layer is fully connected to the subsequent one, and the information propagates unidirectionally from input to output without cyclic loops. The architecture comprises:

- Input Layer: The dimensionality of this layer is determined by the size of feature set, serving as the initial representation of the input data.

- Hidden Layers: Three intermediate layers, each with 1,000 neurons and ReLU activations, facilitate hierarchical feature learning.

- Output Layer: The size of this layer depends on the classification task. For binary classification, a single neuron with a sigmoid activation function outputs a probability-like score, whereas multiclass classification employs multiple neurons with a softmax activation to produce a probability distribution across candidate visualizations.

Training Pipeline The training procedure involves the following steps, illustrated in Fig. 2:

1. Data Preparation: Input features, initially stored as NumPy matrices, are converted into PyTorch tensors and loaded into a DataLoader for efficient batch processing.

2. Forward Pass: Input data is propagated through the network to generate predictions.

3. Loss Computation: Predictions are compared against ground-truth labels using a task-specific loss function (e.g., binary cross-entropy for binary classification).

4. Backpropagation: Gradients are computed via PyTorch's autograd system, enabling efficient optimization.

5. Parameter Update: The Adam optimizer adjusts the network's weights based on the computed gradients.

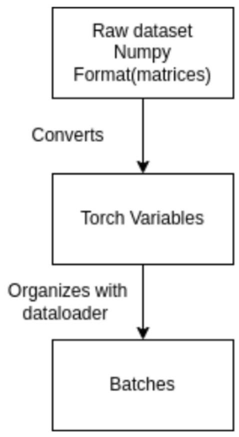

Figure 2: Training pipeline.

- Once trained, the model can autonomously predict suitable visualizations for unseen datasets by leveraging the learned feature representations. The use of PyTorch ensures computational efficiency, with support for both CPU and GPU acceleration, while the modular architecture permits straightforward adaptation to varying input dimensions and classification tasks.

We ultimately concluded that, in summary, VizML's neural network training framework combines a high-capacity architecture with dynamic optimization strategies to achieve robust performance in visualization recommendation.

The absence of significant gains from additional regularization implies that the model's default configuration effectively balances complexity and generalizability.

5.2.2 Embedding Learning (KG4Vis)

According to [2]:

- KG4Vis is a knowledge graph-based approach designed to automate visualization recommendations while maintaining explainability. At its core, it employs a knowledge graph (KG) to model relationships between various entities, such as dataset features, columns, and visualization design choices. To learn meaningful representations of these entities and their relationships, KG4Vis utilizes TransE embeddings, a knowledge graph embedding technique that maps entities and relations into a continuous vector space. This embedding method enables the system to infer new relationships and generate explainable recommendations by capturing structured patterns efficiently.

- The learning process in KG4Vis involves several key steps. During training, the model optimizes its embeddings using gradient descent, adjusting vector representations to better fit the observed relationships in the knowledge graph. A key component of this optimization is the loss function, which ensures that correct relationships (i.e. valid triplets of head entity, relation, and tail entity) receive higher scores than incorrect ones. To improve learning robustness, KG4Vis employs negative sampling, a technique that generates false triplets, helping the model distinguish between valid and invalid relations. This approach improves generalization and scalability, making the system adaptable to various visualization recommendation tasks.

- KG4Vis is implemented using PyTorch, which facilitates efficient computation and updating of knowledge graph embeddings. However, one limitation of this method is its computational cost, particularly when dealing with large-scale knowledge graphs. As the number of entities increases, the training process can become resource-intensive, and the quality of learned embeddings may degrade if the graph contains an excessive number of nodes.

The knowledge graph in KG4Vis consists of several types of nodes, each representing different aspects of the data and visualization process. These include:

- Data columns, which represent individual columns within datasets;

- Discretized continuous data features, where continuous attributes are divided into intervals for analysis;

- Categorical data features, representing discrete attributes;

- Visualization design choices, denoting different visualization types and their configurations.

The relationships between these nodes are captured through edges, which define how entities interact. For instance, edges connect data columns to their respective features (e.g., data type, name), while other edges link data features to appropriate visualization choices, ensuring that certain data characteristics align with optimal visual encodings. Additionally, some edges directly associate data columns with visualization choices, specifying how columns should be used in different chart types.

We then found that, unlike some other recommendation systems, KG4Vis does not incorporate reinforcement learning (RL) to dynamically adjust its knowledge graph based on feedback. Instead, it relies solely on embedded learning to infer recommendations, which remain static once learned. Although this approach ensures efficiency and explainability, it does not adapt over time through trial-and-error mechanisms, which could potentially refine recommendations based on user interactions.

5.3 Visualization Recommendation Process

We found that the visualization recommendation process differs significantly between the two models compared in this study. Both aim to suggest appropriate visualizations; however, they employ fundamentally different approaches to arrive at their recommendations.

5.3.1 VizML's Direct Prediction

According to [1]:

- VizML employs a data-driven approach for visualization recommendation, operating through direct prediction mechanisms. The model analyzes the characteristics of the input data and directly predicts the visualization design choices without explicitly modeling the reasoning process.

This approach leverages deep neural networks trained on large datasets of visualization examples. The direct prediction process follows these steps:

- Feature extraction from input data, including statistical properties, data types, and distribution characteristics
- Processing these features through neural network training
- Outputting probability scores for different visualization types and encoding options
- Selecting the visualization design with the highest probability score

We found that VizML's strengths lie in its ability to identify complex patterns between data characteristics and visualization choices based on statistical correlations observed in the training data. However, this approach significantly decreases possible insight into why specific recommendations are made beyond numeric probability scores.

5.3.2 KG4Vis' Explanation-Based Recommendations

According to [2]:

- KG4Vis adopts a knowledge-based approach centered on explanation and reasoning. The model constructs recommendations through a structured knowledge graph that represents relationships between data attributes, visualization designs, and design principles.

KG4Vis generates recommendations through:

- Analyzing input data and mapping its characteristics to entities in the knowledge graph
- Traversing the knowledge graph using embedding-based algorithms (TransE) to identify related visualization concepts
- Applying rule-based reasoning to evaluate potential visualization options
- Generating explanations that connect input data characteristics to visualization suggestions through explicit reasoning chains

We found that:

- This approach aims to increase interpretability and transparency in the recommendation process. Rather than simply suggesting a visualization type, KG4Vis provides justifications based on established design principles and domain knowledge encoded in its knowledge graph.
- The embedding learning component allows the model to capture semantic relationships between visualization concepts, allowing more thought-out recommendations that consider multiple factors simultaneously.

The following Fig. 3 represents examples of KG4Vis' graphs.

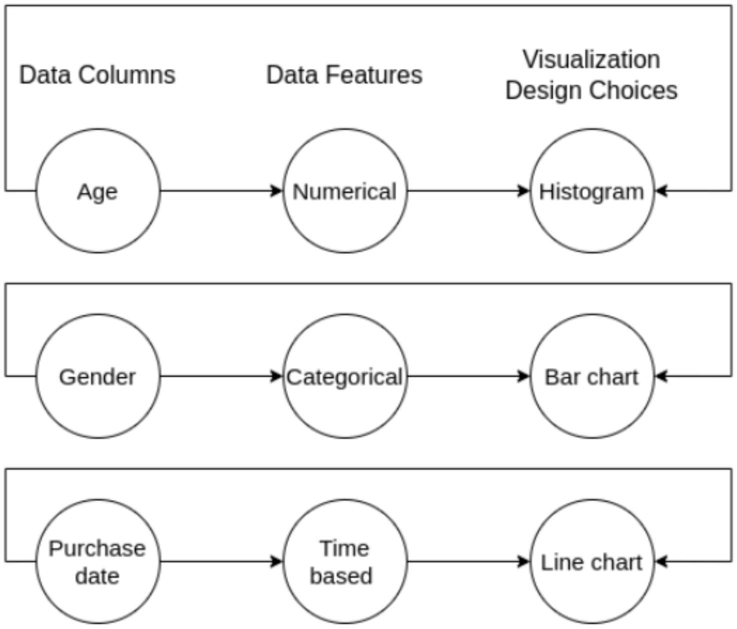

Figure 3: Examples of Graphs on KG4Vis model.

5.3.3 Models' Summary

Table 2: Models' Comparison.

	VizML	**KG4Vis**
Core Approach	Machine learning-based approach using neural networks trained on large corpus of dataset-visualization pairs from Plotly Community Feed	Knowledge graph-based approach using TransE embedding technique to model relationships between data features, columns, and visualization design choices
Explainability	Limited explainability - neural networks work as "black box" models, though feature importances can be extracted from baseline models	High explainability - generates semantically meaningful rules that can be traced back to understand why specific visualizations are recommended
Extensibility	Requires retraining models with new data; limited by the original training corpus structure and visualization types	Easily extensible by adding new entities, relations, and triplets to the knowledge graph without complete retraining
Technology Requirements	- Large-scale dataset (1M+ dataset-visualization pairs) - Neural network training infrastructure - PyTorch/deep learning frameworks - Significant computational requirements for training	- Knowledge graph construction tools - TransE embedding learning - Moderate computational requirements - MDLP discretization for continuous features

5.4 Technology Stack

Both VizML and KG4Vis require specific Python environments and associated libraries to support their implementation. VizML is built on Python 3.7.3, relying heavily on data-science and machine-learning packages for model training, feature extraction, and visualization. Core packages include NumPy, Pandas, Matplotlib, Scikit-learn, and PyTorch, as well as additional utilities for handling time-series, text processing, and imbalanced datasets. The full list of packages and their versions is shown in Table 3.

Table 3: Library Versions.

Name	**Version**
cycler	0.10.0
editdistance	0.5.3
kiwisolver	1.1.0
matplotlib	3.0.3
numpy	1.16.3
pandas	0.24.2
pyparsing	2.4.0
python-dateutil	2.8.0
pytz	2019.1
scikit-learn	0.20.3
scipy	1.2.1
six	1.12.0
seaborn	
IPython	
pytorch	
imbalanced-learn	

In contrast, KG4Vis is implemented using Python 3.7.9, leveraging a more compact library stack centered on knowledge-graph representation and embedding techniques. Its requirements focus on NumPy, Pandas, Scikit-learn, PyTorch, and utility packages like tqdm for efficient progress tracking. These dependencies, shown in Table 4, reflect its emphasis on relational learning and graph-based reasoning rather than end-to-end neural network training.

Table 4: Library Versions.

Name	**Version**
scikit-learn	0.21.0
numpy	1.16.3
editdistance	0.5.3
pandas	0.24.2
pytorch	1.7.0
tqdm	4.66.4

5.5 Dataset

According to [1] and [2]:

- The dataset was retrieved from the KG4Vis GitHub repository. It is the same dataset used for VizML and was originally sourced from various datasets available on Plotly, according to [1] and [2].

It consists of rows corresponding to unique visualizations or charts. Each row contains two main columns:

- `fid`: A unique identifier for each visualization (e.g. xiemei:82, Wini:4).
- `chart_data`: A JSON-like structure containing the configuration and metadata for the visualization.

The dataset includes a variety of visualization types, such as scatter plots, line charts, bar charts, histograms, heatmaps, and fitted curves. Each chart_data entry contains detailed properties for the visualization, including the chart type (e.g. scatter, bar, heatmap), mode (e.g., markers, lines, lines+markers), name (a label or title for the data series), marker properties (e.g., color, size, symbol), and line properties (e.g., color, width, dash style).

For the purpose of model development and evaluation, the dataset was split into training and testing subsets.

5.6 Training

This section details the training process for both VizML and KG4Vis models, covering dataset preparation, feature extraction, and model training. The subsections are structured as follows:

- **5.6.1 Dataset Build:** Describes the preprocessing steps to convert, clean, and prepare the raw dataset for analysis.
- **5.6.2 VizML Feature Extraction:** Explains how numerical features are extracted and normalized for neural network input.
- **5.6.3 KG4Vis Feature Extraction:** Outlines the transformation of raw data into graph-structured representations for knowledge graph construction.
- **5.6.4 VizML Model Training:** Covers the neural network architecture, optimization, and training pipeline for VizML.
- **5.6.5 KG4Vis Model Training:** Details the embedding learning process and knowledge graph optimization for KG4Vis.

5.6.1 Dataset Build

The following steps describe the preprocessing steps used to convert the raw dataset described in the previous section into a dataset that can be used for VizML training. These steps are not necessary for the KG4Vis project, since it uses the raw dataset for feature extraction and model training.

Prior to conducting any analysis, the dataset must undergo a series of preprocessing steps to ensure compatibility and data integrity. The initial phase involves converting the dataset from Comma-Separated Values (CSV) format to Tab-Separated Values (TSV) format. This conversion is performed to mitigate potential parsing conflicts arising from embedded commas within the data fields. The transformation is executed using the following AWK command:

```
awk 'BEGIN { FS=","; OFS="\t" } {$1=$1; print}' raw_data_all.csv
> plot_data.tsv
```

Following the format conversion, the dataset undergoes a data cleaning procedure to eliminate invalid and redundant entries. This step is crucial to maintain the consistency and reliability of subsequent analyses. The cleaning process consists of two sequential operations, executed within the data_cleaning directory:

```
python remove_charts_without_all_data.py
```

This script filters out visualization records that lack complete data attributes, ensuring that only fully specified entries are retained.

```
python remove_duplicate_charts.py
```

This step identifies and removes duplicate visualization entries, preventing redundancy and potential bias in downstream analyses.

Once the dataset has been cleaned, the next phase involves feature extraction, which transforms raw data into structured, machine-interpretable representations. This process is executed by running the following command within the feature_extraction directory:

```
python extract.py
```

The feature extraction module systematically parses the dataset, identifying and encoding relevant attributes that will later inform visualization recommendations. This structured preprocessing pipeline ensures that the input data adheres to the required format and quality standards before further computational analysis and is illustrated in the following Fig. 4.

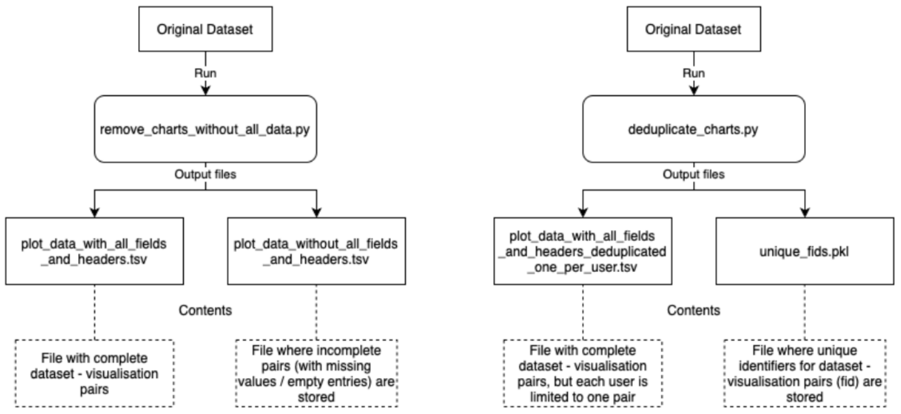
Original Dataset
Run
remove_charts_without_all_data.py
Output files
plot_data_with_all_fields _and_headers.tsv
plot_data_without_all_fields _and_headers.tsv
Contents
File with complete dataset - visualisation pairs
File where incomplete pairs (with missing values / empty entries) are stored
Original Dataset
Run
deduplicate_charts.py
Output files
plot_data_with_all_fields _and_headers_deduplicated _one_per_user.tsv
unique_fids.pkl
Contents
File with complete dataset - visualisation pairs, but each user is limited to one pair
File where unique identifiers for dataset - visualisation pairs (fid) are stored

Figure 4: Data Cleaning Pipeline for VizML.

5.6.2 VizML Feature Extraction

According to [1], VizML performs feature extraction by analyzing dataset columns and quantifying their statistical and structural properties. Specifically, it computes 841 features per dataset, which include metrics such as mean, standard deviation, skewness, kurtosis, and missing value ratios. These features are derived from individual columns and across column pairs, enabling the model to capture relationships within the dataset. The extracted features are then normalized and converted into numerical vectors, forming a feature vector that serves as input to the neural network. This process is entirely automatic and allows the model to generalize across a wide range of dataset types.

5.6.3 KG4Vis Feature Extraction

According to [2], KG4Vis leverages the same raw dataset-visualization pairs used in VizML but represents the extracted information in a different format. Instead of computing numerical feature vectors, KG4Vis encodes dataset characteristics as nodes in a knowledge graph. For example, a column's data type (”Categorical data”) is represented as a node, and its relationship to a recommended chart (”Bar chart”) is encoded as an edge. This transformation from raw data to graph-structured representation allows the system to explicitly model semantic relationships between features and visualization types. The graph structure enhances both interpretability and extensibility, as new features and chart types can be integrated without retraining the system.

5.6.4 VizML Model Training

According to [1], VizML uses a supervised deep learning approach, training a fully connected feedforward neural network using the extracted 841-dimensional feature vectors and their corresponding chart labels. The network is trained using PyTorch, where input vectors are processed in batches, passed through hidden layers, and updated through backpropagation to minimize categorical cross-entropy loss. This training process is computationally intensive, often requiring GPU acceleration and large memory capacity (RAM/VRAM) to handle large datasets. Once trained, the model can predict the most suitable chart type for new datasets via a single forward pass, achieving high-speed inference at the cost of reduced explainability.

5.6.5 KG4Vis Model Training

According to [2], KG4Vis does not rely on traditional classification training but instead trains vector embeddings for each node and relation in the knowledge graph. Using TransE embeddings, KG4Vis learns to represent semantic relationships in vector space such that if A is related to B through R, the embedding satisfies the approximation $A + R \approx B$. This training is conducted using PyTorch, which manages both the computation of the embedding vectors and the optimization of the model through stochastic gradient descent. The trained embeddings allow KG4Vis to perform embedding-based inference, meaning it can recommend the most contextually appropriate visualization and explain the reasoning behind the choice. Additionally, this setup allows the model to infer missing links in the graph, dynamically expanding its knowledge without retraining from scratch.

5.7 Tests

When testing, we evaluate the generalizability of each model.

For VizML, during its testing, we found several problems with the model's implementation. Due to missing code and system-level implementation issues beyond the scope of our expertise, we were unable to execute certain components of the model training pipeline to full capacity.

To avoid these technical limitations, we sought the help of generative AI to fill the gaps in the code and fix most errors; however, no progress was able to be made and we continued to have the same recurring difficulties.

Seeing as we had found another model and had been interacting with it so far, we decided to stop attempting to run VizML and dedicated our full focus to KG4Vis.

For the KG4Vis project, we trained the knowledge graph embedding model using TransE with various parameter configurations to optimize computational efficiency while trying to maintain model quality.

Initially, the evaluation was carried out using a configuration constrained by hardware limitations, specifically, the available system memory. Due to memory-related crashes at higher batch sizes, we iteratively halved this parameter, settling on a batch size of 32 as the largest viable value that allowed training to be completed successfully.

Subsequently, a second configuration was tested, which allowed training with the full batch size of 1024. However, this configuration did not support CUDA acceleration due to the absence of a NVIDIA GPU, relying on CPU computation.

Both configurations used hidden dimensions ranging from 500 to 1000, increasing in steps of 100. This range was selected to study how the representational capacity of the model embeddings affects performance. Larger hidden dimensions can capture more complex patterns, potentially improving accuracy at the cost of an increase in computational cost and training time.

Configuration summary:

- **Configuration 1:** Batch size = 32, Hidden dimensions = 500–1000 (step of 100), ran on Lenovo's Legion Pro 5, on a Windows 11 system, equipped with an AMD Ryzen 9 7945HX processor (2.50 GHz, 32 cores), 32GB of installed RAM and an NVIDIA GeForce RTX 4070 Laptop GPU with 8GB dedicated VRAM. However, training was performed within a Windows Subsystem for Linux (WSL) environment, which utilized Python 3.7.17 with PyTorch 1.7.0 and CUDA 12.9 support through NVIDIA driver version 576.40 but only had access to 15GB of the total system memory. Due to memory management constraints within the WSL, batch size had to be limited to 32 (crashes at higher sizes).
- **Configuration 2:** Batch size = 1024, same hidden dimensions, running on Apple M2 Max (32 GB RAM), no CUDA acceleration available, computation performed on CPU with low power mode on.

Model performance was assessed using standard knowledge graph metrics, including Hit@k and Mean Reciprocal Rank (MRR). These metrics evaluate how effectively the embedding model ranks relevant visualizations when presented with user input or data characteristics. Training convergence was monitored through loss values.

5.8 Comparative Analysis and Discussion of KG4Vis

5.8.1 Scalability

To evaluate the model's scalability, we measured the total training compared for each of the hidden dimension values, as shown in Fig./Table 5 and 6.

Configuration 1

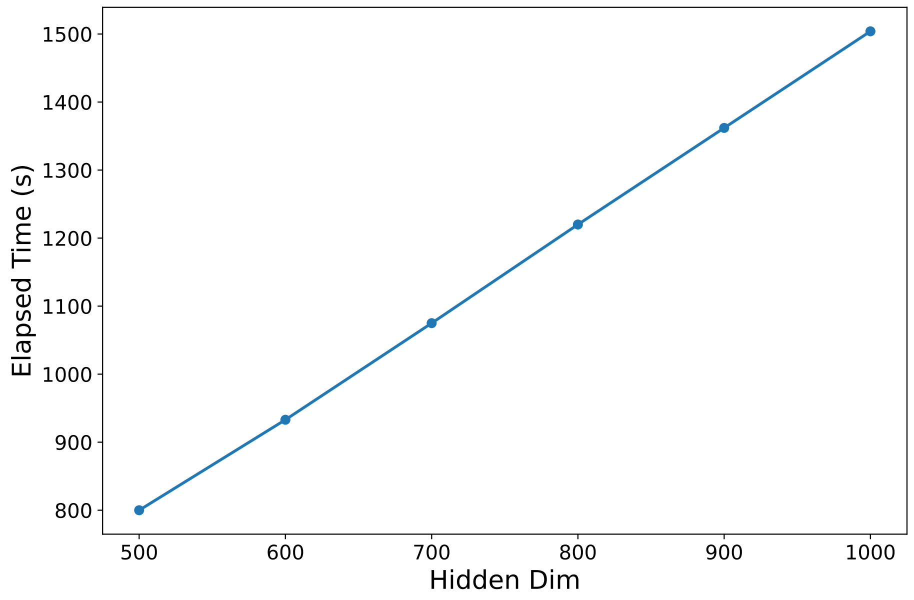

Figure 5: Total training time for different values of Hidden Dimension (Configuration 1).

Table 5: Numerical values of total training time for different values of Hidden Dimension (Configuration 1).

Hidden Dimension	Training Time (s)
500	800
600	933
700	1075
800	1220
900	1362
1000	1504

We noticed that for this configuration, as shown in Table 5, the relation between the hidden dimension and the training time closely follows a linear relation of $m \approx 1,54$. This indicates that training time increases proportionally with model complexity, suggesting efficient and consistent computational scaling as dimensionality grows.

Configuration 2

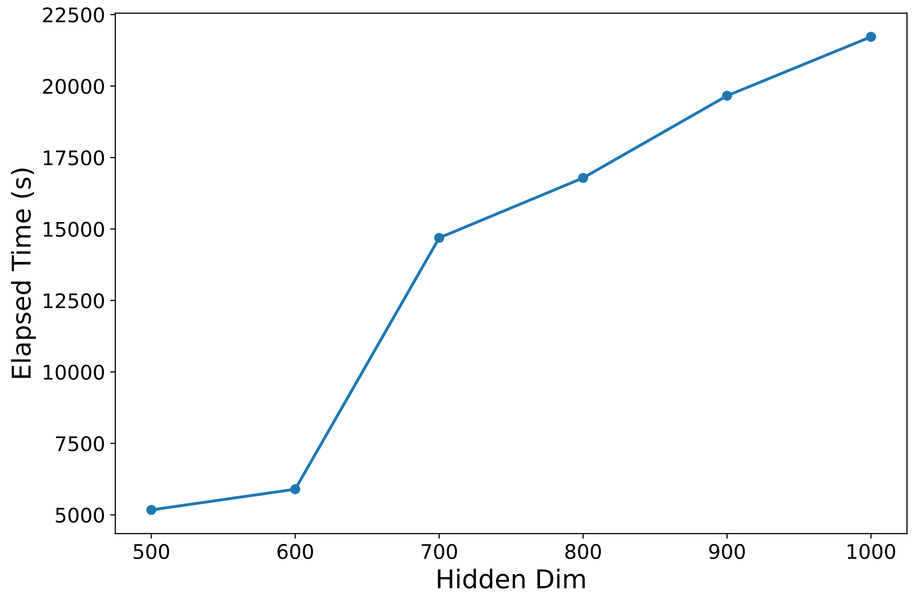

Figure 6: Total training time for different values of Hidden Dimension (Configuration 2).

Table 6: Numerical values of total training time for different values of Hidden Dimension (Configuration 2).

Hidden Dimension	Training Time (s)
500	5173
600	5897
700	14689
800	16787
900	19662
1000	21723

Configuration 2 has a much longer training time overall compared to configuration 1, as we can observe from the results in Table 6, mainly due to the missing CUDA acceleration. The graph shows a drastic increase in training time between 600 and 700 hidden dimensions.

- For hidden dimensions between 500 and 600, the relation is that of $m \approx 10,09$;
- For hidden dimensions between 700 and 1000, the relation is that of $m \approx 21,38$.

This difference suggests that different internal computational behaviors may be triggered at different hidden dimension thresholds, such as memory management or other types of hardware-dependent optimizations.

5.8.2 Performance

To evaluate the model's performance, we analyzed the overall evolution of the metrics over the different hidden dimensions, as shown in Fig./Table 7 and 8.

Configuration 1

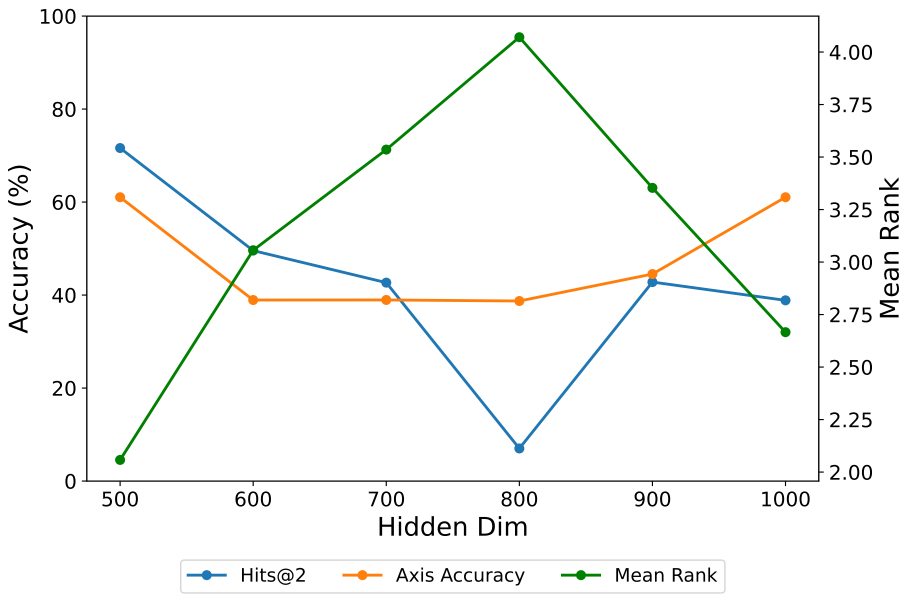

Figure 7: Metrics evolution for different values of Hidden Dimension (Configuration 1).

Table 7: Metrics for different values of Hidden Dimension (Configuration 1).

Hidden Dimension	Hits@2 (%)	Axis Accuracy (%)	Mean Rank
500	71.6	61.0	2.06
600	49.6	38.9	3.06
700	42.7	38.9	3.54
800	7.0	38.7	4.07
900	42.8	44.5	3.35
1000	38.9	61.0	2.67

For a batch size of 32, we obtained the best results for a hidden dimension of 500, as is seen in Table 7. Contrary to what is expected, reducing the size of each batch from 1024 (default value) using a lower hidden dimension proved to be beneficial for the model. We also noticed that for a hidden dimension of 800, the results were significantly worse compared to the other values used. Since only the hidden dim changed across different executions, this could suggest that using a batch size of 32 and a hidden dimension of 800 caused the model to overfit or enter a saturated regime, leading to ineffective learning.

Configuration 2

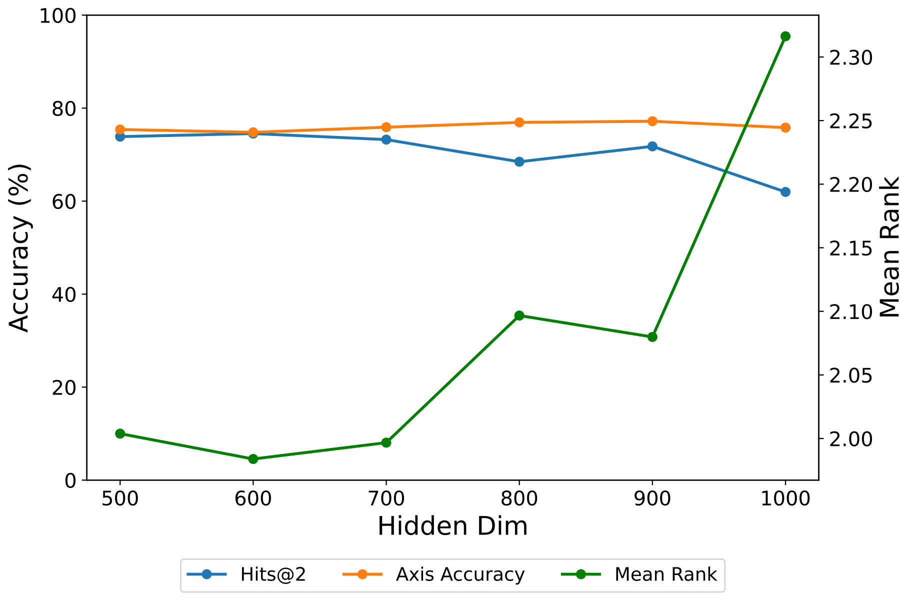

Figure 8: Metrics evolution for different values of Hidden Dimension (Configuration 2).

Table 8: Metrics for different values of Hidden Dimension (Configuration 2).

Hidden Dimension	Hits@2 (%)	Axis Accuracy (%)	Mean Rank
500	73.9	75.4	2.00
600	74.5	74.8	1.98
700	73.2	75.9	2.00
800	68.5	76.9	2.10
900	71.8	77.2	2.08
1000	62.0	75.8	2.32

Similarly to the previous configuration, the best results were obtained at a lower hidden dimension, specifically 600, seen in Table 8. This suggests that at higher batch sizes, smaller hidden dimensions may lead to better generalization by reducing model complexity while still maintaining sufficient representational capacity. Higher hidden dimensions (e.g. 800-1000) likely introduced excess capacity, which may have led to overfitting. Additionally, lower hidden dimensions train faster and may benefit from improved gradient stability and a reduced risk of overfitting, resulting in a better generalization, observed by highest Hits@2 and the lowest Mean Rank.

A performance drop is also observed in a hidden dimension of 800, although less pronounced than in the previous configuration. This consistency across different batch sizes suggests that, for this particular model architecture, a hidden dimension of 800 may correspond to a suboptimal capacity regime, potentially leading to inefficient gradient flow or representational instability.

5.8.3 Accuracy

To evaluate the model's accuracy, we analyzed the evolution of the Hits@2 and Axis Accuracy metrics across the different hidden dimensions, as shown in Fig./Table 9 and 10.

Configuration 1

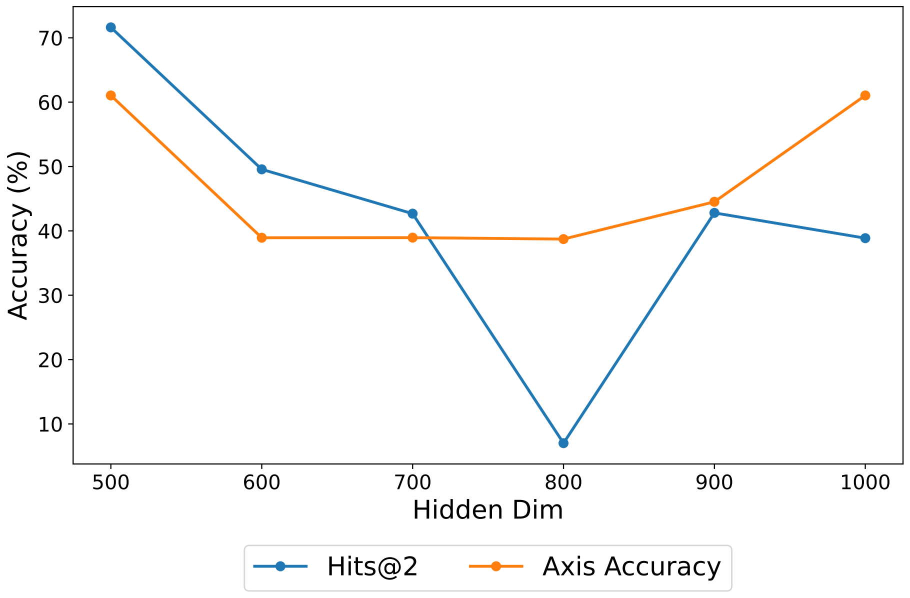

Figure 9: Hits@2 and Axis Accuracy percentages for different values of Hidden Dimension (higher is better) (Configuration 1).

Table 9: Hits@2 and Axis Accuracy for different values of Hidden Dimension (Configuration 1).

Hidden Dimension	Hits@2 (%)	Axis Accuracy (%)
500	71.6	61.0
600	49.6	38.9
700	42.7	38.9
800	7.0	38.7
900	42.8	44.5
1000	38.9	61.0

The evolution of Hits@2 and Axis Accuracy aligns with the performance trends discussed above. As shown in Table 9, the highest accuracy metrics were observed at a hidden dimension of 500, while a clear degradation was observed at 800. These results reinforce the conclusion drawn regarding the capacity and generalization of the model.

Configuration 2

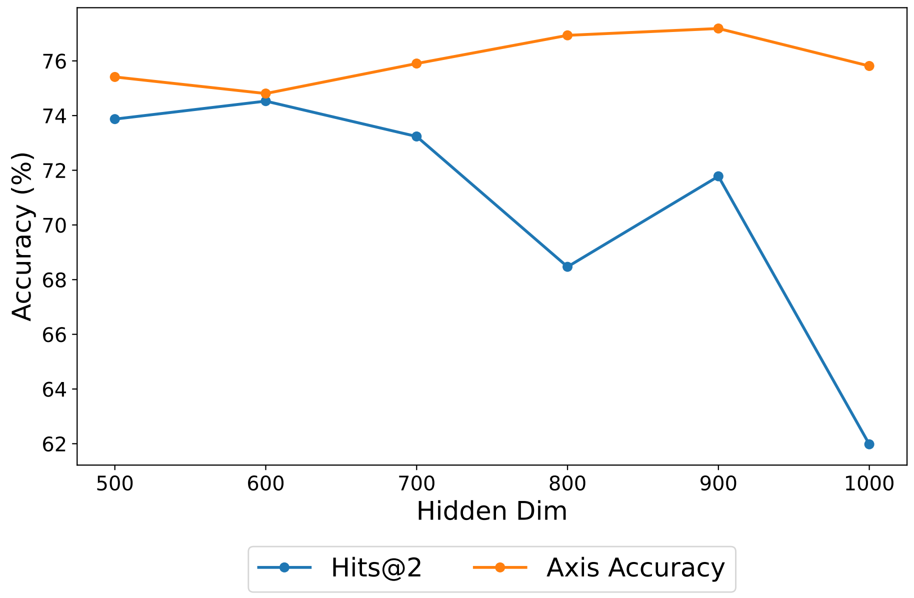

Figure 10: Hits@2 and Axis Accuracy percentages for different values of Hidden Dimension (higher is better) (Configuration 2).

Hits@2 and Axis Accuracy remained relatively high in most dimensions, with the best values at a hidden dimension of 600, observed in Table 10. These results support the earlier observation that smaller hidden dimensions tend to generalize better at larger batch sizes.

Table 10: Hits@2 and Axis Accuracy for different values of Hidden Dimension (Configuration 2).

Hidden Dimension	Hits@2 (%)	Axis Accuracy (%)
500	73.9	75.4
600	74.5	74.8
700	73.2	75.9
800	68.5	76.9
900	71.8	77.2
1000	62.0	75.8

5.8.4 Precision

To evaluate the model's precision, we analyzed the evolution of the Mean Rank metric across the different hidden dimensions, as shown in Fig./Table 11 and 12.

Configuration 1

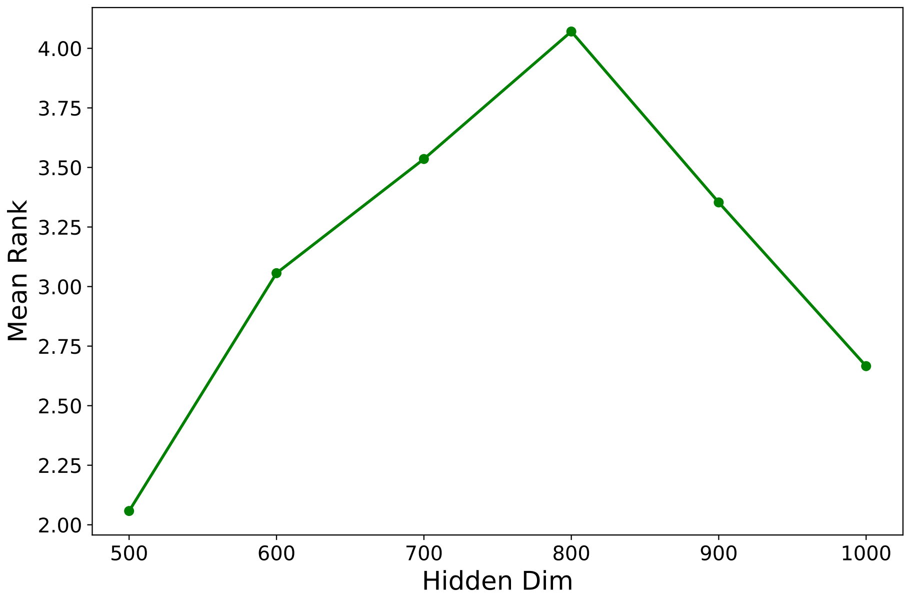

Figure 11: Mean Rank for value different values of Hidden Dimension (lower is better) (Configuration 1).

Table 11: Mean Rank for different values of Hidden Dimension (Configuration 1).

Hidden Dimension	Mean Rank
500	2.06
600	3.06
700	3.54
800	4.07
900	3.35
1000	2.67

Mean Rank values also followed the trend discussed previously, where the best value was observed at a hidden dimension of 500, shown in Table 11. The sharp increase in Mean Rank at 800 confirms the degraded performance, highlighting reduced precision in ranking predictions.

Configuration 2

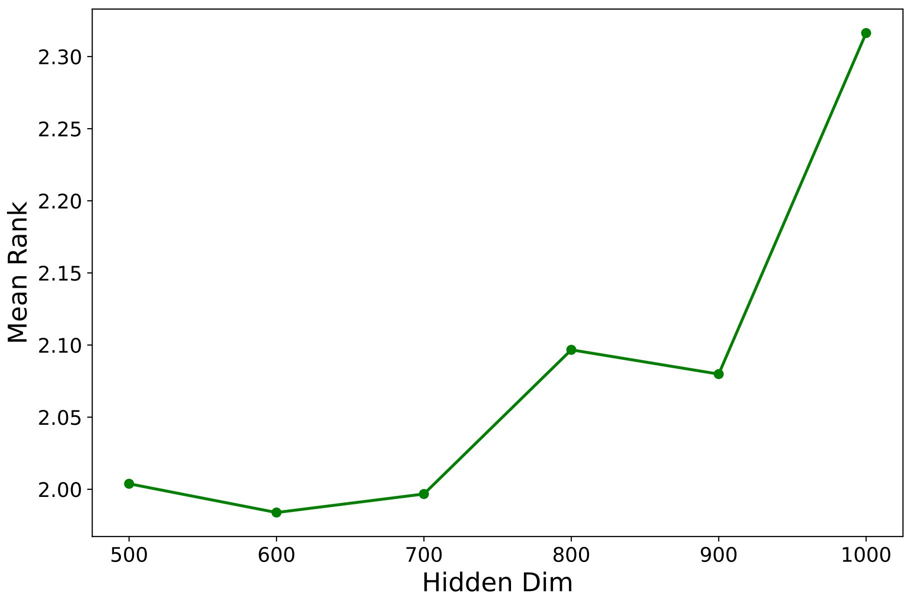

Figure 12: Mean Rank for value different values of Hidden Dimension (lower is better) (Configuration 2).

Table 12: Mean Rank for different values of Hidden Dimension (Configuration 2).

Hidden Dimension	Mean Rank
500	2.00
600	1.98
700	2.00
800	2.10
900	2.08
1000	2.32

As observed earlier and in Table 12, the Mean Rank was better at a hidden dimension of 600. The gradual increase beyond this point supports the argument that larger hidden dimensions may hurt precision due to increased model complexity and potential overfitting.

6 Conclusions

6.1 Achieved results

Comparative Analysis Implementation: We conducted a comprehensive comparison between the VizML and KG4Vis approaches, analyzing their core methodologies, explainability mechanisms, extensibility capabilities, and technology requirements. This analysis provided clear insights into the fundamental differences between neural network-based and knowledge graph-based visualization recommendation systems.

Successful KG4Vis Implementation and Testing: We successfully implemented and tested the KG4Vis model across multiple configurations, evaluating performance with different hidden dimensions (500-1000) and batch sizes (32 and 1024). Our tests revealed optimal performance at lower hidden dimensions (500-600), contrary to typical expectations, demonstrating the importance of empirical validation in model optimization.

Performance Metrics Analysis: We established comprehensive evaluation metrics including Hits@2, Axis Accuracy, and Mean Rank to assess model performance. Configuration 2 achieved the best overall results with Hits@2 of 74.5%, Axis Accuracy of 74.8% and Mean Rank of 1.98 at hidden dimension 600, demonstrating effective visualization recommendation capabilities.

Scalability Insights: We documented the computational scaling behavior of KG4Vis, finding linear relationships between hidden dimensions and training time ($m \approx 1.54$ for Configuration 1, and varying relationships for Configuration 2), providing valuable insights for resource planning and model deployment.

Technical Documentation: We created a complete technical implementation guide covering dataset pre-processing, feature extraction, model training pipelines, and evaluation procedures, contributing to the reproducibility of knowledge graph-based visualization recommendation research.

Limitations and Challenges Identification: We identified and documented technical challenges encountered with VizML implementation, providing valuable lessons for future research and highlighting the importance of robust implementation frameworks in machine learning research.

These results directly address our stated objectives of analyzing both approaches, understanding their computational requirements, and successfully implementing at least one working model for performance evaluation.

6.2 Lessons learned

Model Complexity vs. Performance Trade-Offs: Contrary to conventional wisdom that larger models perform better, our experiments revealed that lower hidden dimensions (500-600) consistently outperformed higher dimensions (800-1000). This taught us the importance of empirical validation over theoretical assumptions and highlighted that the optimal model capacity depends heavily on the characteristics of the dataset and computational constraints.

Hardware Limitations Impact on Research: Working within WSL environment constraints (limited to 15GB RAM) and varying hardware configurations (CUDA vs. CPU-only) demonstrated how computational resources directly influence research scope and methodology. We learned to adapt experimental designs to hardware limitations while maintaining scientific rigor.

Implementation Challenges in Academic Research: Our difficulties with VizML implementation highlighted the critical importance of code reproducibility and documentation in academic research. Missing components and system-level issues taught us that even well-documented papers may have implementation gaps that require significant troubleshooting skills.

Explainability vs. Performance Balance: Through comparing VizML's black-box approach with KG4Vis's interpretable knowledge graphs, we learned that the choice between explainability and raw performance depends on the application context. This reinforced the importance of considering the end-user requirements when selecting machine learning approaches.

Iterative Development and Adaptation: When faced with technical roadblocks with VizML, our decision to focus entirely on KG4Vis taught us the value of strategic pivoting in research projects. Sometimes redirecting efforts toward achievable goals yields more valuable results than persisting with problematic implementations.

Collaborative Research Skills: Working in a four-person team across different phases (research, implementation, analysis, documentation) enhanced our ability to distribute work effectively, coordinate parallel efforts, and integrate diverse contributions into a cohesive final product.

These lessons will prove invaluable in future research endeavors, particularly in understanding the practical challenges of implementing and evaluating machine learning systems in academic and professional settings.

6.3 Future work

Future work could explore several directions to build on this project. One promising avenue would be developing a **hybrid system** that combines VizML's fast neural network predictions with KG4Vis' explainable knowledge graph approach, potentially offering both speed, transparency, and explainability.

While VizML's final results are recommended charts based on data characteristics, the ultimate choice still relies on human judgment. Future iterations could leverage AI to further refine recommendations by analyzing contextual factors, such as the user's analytical goal, audience, or dataset nuances to suggest the most effective chart for a specific situation, reducing ambiguity in decision-making.

Resolving the technical challenges that prevented VizML's full implementation remains important to enable a proper comparison between the two methods.

The KG4Vis system could be enhanced by experimenting with more **advanced knowledge graph embedding techniques** or incorporating reinforcement learning could improve adaptability by allowing dynamic updates to the graph based on user feedback or new data trends.

7 Appendix

7.1 Log of the meetings

The project activities were carried out over several weeks with the following key milestones:

- **First Meeting (13/02/25)**: Initial discussion of the VizML paper and verification of the dataset.
- **Second Meeting (20/02/25)**: Analysis of the VizML and KG4Vis papers, making explanatory diagrams for each one, dataset characterization, and proceed to do the software installation.
- **Third Meeting (27/02/25)**: Exploration of neural network training and embedding learning, start running the dataset.
- **Fourth Meeting (06/03/25)**: Deep dive into scalability challenges, computational costs, and model training preparation.
- **Fifth Meeting (13/03/25)**: Researched graph storage methods, explored training tech, calculated computational costs, corrected prior errors, detailed VizML-KG4Vis pipeline, added references, prepared dataset scripts, and initiated report development.
- **Sixth Meeting (20/03/25)**: Dove deeper into the code, analyzed neural network topology and deep learning model, explored Torch variables and data loaders, implemented simple classifiers, documented dataset file types and execution, and reviewed the report structure.
- **Seventh Meeting (03/04/25)**: Filled initial report sections, refined index by removing redundant terms (e.g., "overview"), restructured to prioritize "Concepts" first. Added comparative analysis of papers/code (performance/scalability post-execution), included metrics (accuracy, precision), and pre/post-execution comparisons. Concluded with work distribution.
- **Eighth Meeting (08/04/25)**: Restructured report to discuss approach before dataset, reordered sections (3.3 before 3.9 unless intermediate chapters reference it), added centered image captions. For code, explored applying the functions with the use of AI. Documented issues with functions, noting attempted fixes with a disclaimer about potential deviations from authors' intent.
- **Ninth Meeting (24/04/25)**: After multiple attempts to adjust parameters and test VizML, we concluded it wasn't yielding meaningful results, so we decided to shift focus. Instead, we ran the original KG4Viz code as implemented by the authors.

- **Tenth Meeting (07/05/25)**: Filled Missing report Topics and tried testing KG4Vis code for different dimensions changing some testing parameters (inferior batch size , hidden dimension)
- **Eleventh Meeting (15/05/25) onwards**: Result analysis of different configurations and report writing.

7.2 Command List

1. Clone the repository:
Run:

```
git clone https://github.com/KG4VIS/Knowledge-Graph-4-VIS-Recommendation
```

Or download the source code directly from `https://github.com/KG4VIS/Knowledge-Graph-4-VIS-Recommendation`

2. Install Python 3.7.9
Ensure the system uses Python 3.7.9 (other versions may cause compatibility issues).

3. (Optional) Install CUDA
If supported, install CUDA on the system (NVIDIA GPU required). Otherwise, proceed with CPU-only PyTorch installation.

4. Install the required packages:
Run:

```
pip install scikit-learn==0.21.0 numpy==1.16.3 editdistance==0.5.3 pandas
    ==0.24.2 tqdm==4.66.4
```

If using CUDA:

```
pip install torch==1.7.0+cu110 torchvision==0.8.1+cu110 torchaudio==0.7.0
     -f https://download.pytorch.org/whl/torch_stable.html
```

Otherwise (CPU-only):

```
pip install torch==1.7.0 torchvision==0.8.1 torchaudio==0.7.0
```

5. Dataset download and code execution:
Follow the instructions in the README file of the repository to download the dataset (and possibly features), run the code, and tune training parameters.

References

[1] Kevin Hu, Michiel A. Bakker, Stephen Li, Tim Kraska, and César Hidalgo. Vizml: A machine learning approach to visualization recommendation. In *CHI Conference on Human Factors in Computing Systems Proceedings (CHI 2019)*, pages 1–12, Glasgow, Scotland UK, 2019. ACM.

[2] Haotian Li, Yong Wang, Songheng Zhang, Yangqiu Song, and Huamin Qu. Kg4vis: A knowledge graph-based approach for visualization recommendation. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):195–209, 2021.