Télécom Paris – Institut Polytechnique de Paris

2025/2026 – PRJ_4SD06_TP

Project in Data Science & Artificial Intelligence

Better Embeddings for Better GNNs

Node-feature choice and its impact on GNN performance
for e-commerce co-purchase graphs

Supervisor: Clément Wang

Team (6 members)

Adrien Bertrand
Ala Ellefi
Matheus Schreiber
Guilherme Efgen
Qingyue Deng
Yuang Ding

June 23, 2026

Abstract

Graph Neural Networks (GNNs) have emerged as a powerful paradigm for e-commerce marketplaces, enabling platforms to leverage relational signals—such as product co-purchases to optimize downstream tasks like category classification. However, standard graph benchmarks often rely on superficial, sparse node features (e.g., bag-of-words) that fail to capture rich semantic and visual context. This project systematically investigates the impact of diverse node-feature representations on GNN performance using a co-purchase graph derived from public Amazon datasets. We extract dense product features using frozen, pretrained language models (`TF-IDF`, `BoW`, `sBERT`, `CLIP-text`, and `Qwen3`) alongside deep visual embeddings. Operating under real-world resource constraints, we benchmark a two-layer `GraphSAGE` model against traditional non-graph classifiers (Logistic Regression and Multi-Layer Perceptrons). The empirical findings demonstrate that advanced contextual embeddings substantially outperform classical baselines across all architectures. Furthermore, integrating the topological graph structure provides a consistent accuracy lift over equivalent non-graph counterparts, while multimodal integration yields superior predictive synergy. Finally, we provide a holistic scalability analysis mapping the critical trade-offs between classification accuracy, feature dimensionality, and computational complexity, delivered alongside a fully reproducible, end-to-end pipeline.

Contents

1 Introduction, Hypotheses and Motivation **3**
- 1.1 Context and motivation . . . 3
- 1.2 Hypothesis . . . 3

2 Implementation Process **3**
- 2.1 Dataset . . . 3
- 2.2 Train / Test / Validation split . . . 4
- 2.3 Baseline encoders . . . 5
- 2.4 Text embeddings . . . 5
- 2.5 Image embeddings . . . 5
- 2.6 Multimodal fusion . . . 6
- 2.7 Classifiers . . . 6
  - 2.7.1 Non-graph . . . 6
  - 2.7.2 GNNs . . . 7

3 Challenges and Findings **7**
- 3.1 Challenges encountered . . . 7
- 3.2 Findings . . . 8
  - 3.2.1 Text pipeline results . . . 8
  - 3.2.2 Image pipeline results . . . 10
  - 3.2.3 Multimodal fusion results . . . 10
- 3.3 Cost analysis . . . 11

4 Alignment with the Proposed Plan **13**
4.1 Was the planning followed? 13
4.2 What could have been done differently, and lessons learned 13

5 Code Availability **13**

6 Contributions of Team Members **13**
6.1 Individual contributions 13
6.2 Teamwork and tools 14

7 Use of AI Tools **14**
7.1 In the code 14
7.2 In the report 14

1 Introduction, Hypotheses and Motivation

1.1 Context and motivation

Online marketplaces such as those built by Mirakl connect a large number of sellers and buyers around catalogs of millions of products. Understanding how products relate to one another (how they are used, viewed, and purchased) is central to many downstream tasks, from recommendation and search to catalog organization. A particularly informative signal is the *also-bought* relation: when consumers buy an item, the other products they frequently go on to purchase tend to be semantically related. These interactions naturally form a graph whose nodes are products and whose edges encode this shared transaction behavior. Graph Neural Networks (GNNs) are well suited to this setting, as they propagate information along edges and can predict useful product properties, such as category or attributes.

The performance of a GNN depends not only on its architecture but also on the *node features* it is given. Many public graph benchmarks ship with precomputed bag-of-words or TF-IDF features, which are cheap to obtain but discard much of the semantic content of a product's textual description and ignore its images entirely. In contrast, modern pretrained models (text transformers such as `sBERT` and `Qwen3`, and vision encoders such as `ResNet`, `CLIP`, and `DINOv2`) produce dense embeddings that capture far richer semantics, and can be used as frozen feature extractors with no task-specific fine-tuning. Whether, and by how much, these richer features actually improve downstream GNN performance on an e-commerce co-purchase graph is the question this project addresses.

1.2 Hypothesis

The study goal was to check how the choice of node features affects GNN performance on a co-purchase graph built from public Amazon datasets, for a product classification task. Concretely, the following three main hypotheses were set for this research.

- **H1** GNNs using co-purchase links beat classifiers using the same features without the graph.
- **H2** Combining text and image embeddings beats single-modality features.
- **H3** Modern encoders such as `sBERT`, `Qwen3`, `CLIP` and `DINOv2` beat baselines like `TF-IDF` and `BoW` on GNNs.

2 Implementation Process

2.1 Dataset

The project utilizes the Amazon Co-Purchase Product Graph Dataset (curated by the McAuley[1] Lab at UCSD). In this dataset, individual products serve as nodes, and undirected edges are established between items that are "also bought" by clients. Each product record contains rich text metadata, including the product title, description, and a sequence-nested list of subcategories (ranging from broad top-level parent domains down to fine-grained child leaf classes). This

metadata serves as the raw source material for extracting text features, such as sparse `TF-IDF` vocabularies or dense `Sentence-BERT` and `Qwen3` embeddings.

This project leverages the *Toys & Games* subset, encompassing over `300,000` products (`148` MB of text metadata and `5` GB of image data). Target labels were extracted by taking the first subcategory listed in each product's metadata hierarchy (e.g., classifying a product under the *Puzzles* subcategory). Restricting the label space to this first subcategory level yielded `18` unique target classes, as outlined in Table 1.

Table 1: Subcategories and quantity of products.

#	Subcategory	Products	Percentage
1	*Action Figures & Statues*	48 875	15.40%
2	*Games*	38 817	12.23%
3	*Stuffed Animals & Plush*	34 805	10.97%
4	*Hobbies*	26 322	8.29%
5	*Dolls & Accessories*	22 150	6.98%
6	*Toy Remote Control & Play Vehicles*	19 678	6.20%
7	*Party Supplies*	17 566	5.54%
8	*Arts & Crafts*	14 666	4.62%
9	*Sports & Outdoor Play*	14 542	4.58%
10	*Puzzles*	13 655	4.30%
11	*Learning & Education*	12 541	3.95%
12	*Novelty & Gag Toys*	11 720	3.69%
13	*Building Toys*	10 116	3.19%
14	*Grown-Up Toys*	9 256	2.92%
15	*Dress Up & Pretend Play*	9 066	2.86%
16	*Electronics for Kids*	6 277	1.98%
17	*Baby & Toddler Toys*	5 145	1.62%
18	*Tricycles, Scooters & Wagons*	2 132	0.67%
	Total (Toys & Games)	317 329	100%

2.2 Train / Test / Validation split

The split strategy used for the analysis was `70`% training data, `15`% test data and `15`% validation data. These split choice for these numbers was inspired by the splits made on the reference papers for this study [2–4].

To guarantee experimental consistency, the dataset splits were cached locally and reused across all feature extraction configurations. This tracking policy ensured that all the encoders were trained and evaluated on identical data partitions.

Regarding models evaluations, classification performance was assessed using the `accuracy` metric, computed by averaging the alignment between predicted and ground-truth subcategory labels.

2.3 Baseline encoders

Text features were extracted via a Term Frequency-Inverse Document Frequency (TF-IDF) representation[5], constructed using the `TfidfVectorizer` class from the `scikit-learn`[6] library. To eliminate uninformative grammatical tokens and optimize the feature space, *Scikit-Learn*'s built-in English stop-word dictionary was applied during tokenization. A similar approach was used for the Bag-of-words encoder, but using `CountVectorizer` as the imported library class[7, 8].

2.4 Text embeddings

Beyond the sparse baselines, three families of neural text encoders were evaluated.

- **sBERT** [9] To incorporate a dense semantic representation of textual features, it was used the `SentenceTransformer` library to implement a `Sentence-BERT` (`sBERT`) encoder pipeline. Specifically, the framework leverages the default `"all-MiniLM-L6-v2"` pre-trained checkpoint as a frozen feature extractor, meaning there are no internal parameters to learn during the training stage. The `SBERTEncoder` processes the textual Amazon product dataset by computing vector mappings in mini-batches of 256. The choice of batch size was made empirically.
- **CLIP-text**[10] is the text tower of a contrastively trained text–image model, so its embedding space is aligned with visual concepts.
- **Qwen3** [11] is a recent LLM-based encoder trained with a retrieval objective. For every product, the title and description are concatenated into a single string and passed through the encoder once; the resulting vector becomes the node feature. Tests were conducted with models comprising 0.6B, 4B and $8B$ parameters.

All neural encoders are used strictly as frozen feature extractors (no gradients are propagated through them) which keeps the comparison focused on the *features* rather than on end-to-end fine-tuning, and enables the cache of each embedding matrix to disk so it is computed only once per encoder-dataset pair.

2.5 Image embeddings

Instead of using the precomputed `Caffe` features released with the dataset, image embeddings were extracted directly from the product images using three modern vision encoders. The goal is to discover whether richer visual representations translate into better classification.

A pipeline was built such that it downloads each product image from its `imUrl` field, runs it through the three encoders, and saves the results to a `Parquet` file periodically so that the extraction can be resumed if interrupted. The three encoders are:

- **ResNet-50**[12]: without the classification head and with the average-pooled representation, giving a 2048-dimensional vector per product.

- **CLIP ViT-B/32**[10]: it was used `get_image_features`, which projects the visual encoding into the shared image-text space, giving a 512-dimensional vector. Using `vision_model.pooler_output` directly would instead return the 768-dimensional pre-projection representation, which is *not* the standard CLIP embedding.
- **DINOv2-Base**[13]: we take the `[CLS]` token from the last hidden state, giving a 768-dimensional vector.

Products whose image could not be fetched or decoded are assigned a zero vector, which represents only 0.01% of the data.

2.6 Multimodal fusion

To test whether combining modalities helps, as posted by **H2** (Combining text and image embeddings beats single-modality features), it was attached a 4096-dimensional image vector to each node alongside its text embedding and combine them with one of three fusion strategies before the GNN backbone.

Concatenation simply stacks the two vectors (text $\|$ image); it is parameter-free but inflates the input dimension.

Weighted fusion learns two softmax scalars that mix (optionally projected) text and image vectors, letting the model down-weight the weaker modality.

Gated fusion[14] computes an instance-level gate from the concatenated input and uses it to modulate the contribution of each modality per node. In all three cases the fused representation is fed to the same GNN backbone used for the single-modality runs, so the comparison isolates the effect of adding images.

2.7 Classifiers

2.7.1 Non-graph

The Multi-Layer Perceptron (MLP) baseline architecture was optimized using an arbitrarily determined learning rate of `0.01`. Architecturally, the network employs a sequential feedforward design consisting of two hidden linear layers interleaved with Rectified Linear Unit (ReLU) activation functions to capture non-linear feature interactions, alongside a Dropout regularization layer to mitigate overfitting. Optimization was governed by the `CrossEntropyLoss` function, integrated with an early stopping strategy. This regularization mechanism halted training if the validation loss failed to decrease for `10` consecutive epochs.

Also, a Logistic Regression approach was used, and it was built using the Scikit-Learn's `LogisticRegression`, where the model was initialized with an arbitrarily maximum iteration cap of `500` and configured with an L_2 regularization penalty to prevent overfitting during optimization.

2.7.2 GNNs

Firstly, to model the structural dependencies between products, generation pipeline is implemented. The graph's topology is constructed by mapping unique *Amazon Standard Identification Numbers* (ASINs) to sequential integer identifiers to form a comprehensive node index. Relationships are extracted by analyzing the product metadata ("*also bought*" parameter from dataset). The algorithm parses these records and then pairs each source product with its indexed neighbors while discarding any external references outside the current dataset boundaries. Once the raw directed connections are aggregated, self-loops (where a product points to itself) are systematically removed. Finally, the connection matrix is processed using `PyTorch Geometric`[15] utilities, including the *deduplication* step done by the `coalesce` function. This entire process resulted in a graph with `317329` nodes and `7492956` edges.

Then, with the graph properly in place, a graph-based classification framework is implemented utilizing a custom two-layer `GraphSAGE`[16] neural network alongside an automated Amazon relationship graph extraction routine. The graph topological structure is derived strictly from the "*also bought*" metadata relationships. To counter hardware memory bottlenecks when dealing with high-dimensional embedding spaces, input feature vectors containing more than 512 dimensions or stored as sparse matrices are compressed to a low-dimensional dense matrix of size $d = 128$ via Truncated Singular Value Decomposition (`TruncatedSVD`). Early-stopping was applied to the training with a patience threshold of 10 epochs. Structurally, the `GraphSAGE` model utilizes an input dimension size equal to the reduced or native feature dimensionality, projecting representations into a hidden state vector of size 256 via a localized neighborhood aggregation convolution (`SAGEConv`). This is followed by a ReLU activation function and dropout, and then mapped directly to the unique target class heads, optimizing network updates via the Adam optimizer with a learning rate of `0.01` and a Cross-Entropy Loss optimization target.

3 Challenges and Findings

3.1 Challenges encountered

Some challenges were faced during the making of this study, and they will be discussed in details in this section.

- The initial experiments were conducted on the *Musical Instruments* metadata collection, which contains over `80,000` product records. However, the hierarchical subcategories at the interest level presented extreme sparsity and imbalance (exceeding `142` distinct granular subcategories), which complicated stable model convergence and memory usage. Consequently, a systematic analysis was performed across alternative subsets to establish an optimal balance between subcategory density and per-class product distribution. This ideal distribution was achieved using the *Toys & Games* dataset, which features `18` well-balanced, distinct target classes, at the cost of a substantially larger graph (over `300.000` products).
- Due to the shared resource constraints of the Télécom Paris GPU cluster, specific limitations regarding memory consumption and job scheduling were imposed during model

optimization. Experiments were executed on an *NVIDIA Tesla P100 PCIe 16 GB* GPU. Initial configurations utilizing higher hyperparameter bounds (such as scaling the hidden layer dimension for the uncompressed `sBERT` features) resulted in prohibitive memory allocation demands exceeding 20 GiB, which triggered out-of-memory errors. Consequently, a systematic hyperparameter grid search was performed to settle on the stable configurations presented in this study.

- Also, the hardware memory constraints on the cluster directly impacted the evaluation of the larger `sBERT` variant on the text analysis, hence the study focused its comparative analysis oo the compact `all-MiniLM-L6-v2` variant.
- The initial graph construction framework utilized a dense compilation of structural links, including the *also_bought*, *also_viewed*, *bought_together*, and *buy_after_viewing* relationship types. However, this comprehensive approach generated an exceptionally dense network exceeding `14` million edges, introducing significant topological complexity and computational intractability. To optimize resource utilization and maintain a stable memory footprint, the edge-extraction criteria were subsequently restricted to isolate the *also_bought* co-purchase metric for the final analysis (`7` million edges).
- Due to the large number of edges in the Toys & Games graph, GPU memory became a bottleneck during the message-passing step, as `SAGEConv` needs to materialize a message tensor of shape [num_edges $\times$ d] all at once. We addressed this by switching to mini-batch training via `NeighborLoader`, which samples a fixed neighbourhood per node at each step, keeping memory usage bounded.

3.2 Findings

3.2.1 Text pipeline results

The text-pipeline results support both **H1** (GNNs leveraging co-purchase links beat classifiers using the same features without the graph) and **H3** (Modern encoders such as `sBERT`, `Qwen3`, `CLIP` and `DINOv2` beat baselines like `TF-IDF` and `BoW` on GNNs). It was observed that, among the text encoders, `CLIP-Text` produced the models with highest accuracy on both non-graph approach, while the graph-based relied on `Qwen3` with 8B parameters as the most accurate. This confirms that richer pretrained semantics improve the accuracy of the classification.

Moreover, it is perceptible that adding the graph structure gives a further, consistent gain: `GraphSAGE` improves over the MLP by roughly 2% on average. The Figure 1 shows the comparison between accuracies of each encoding-model pair.

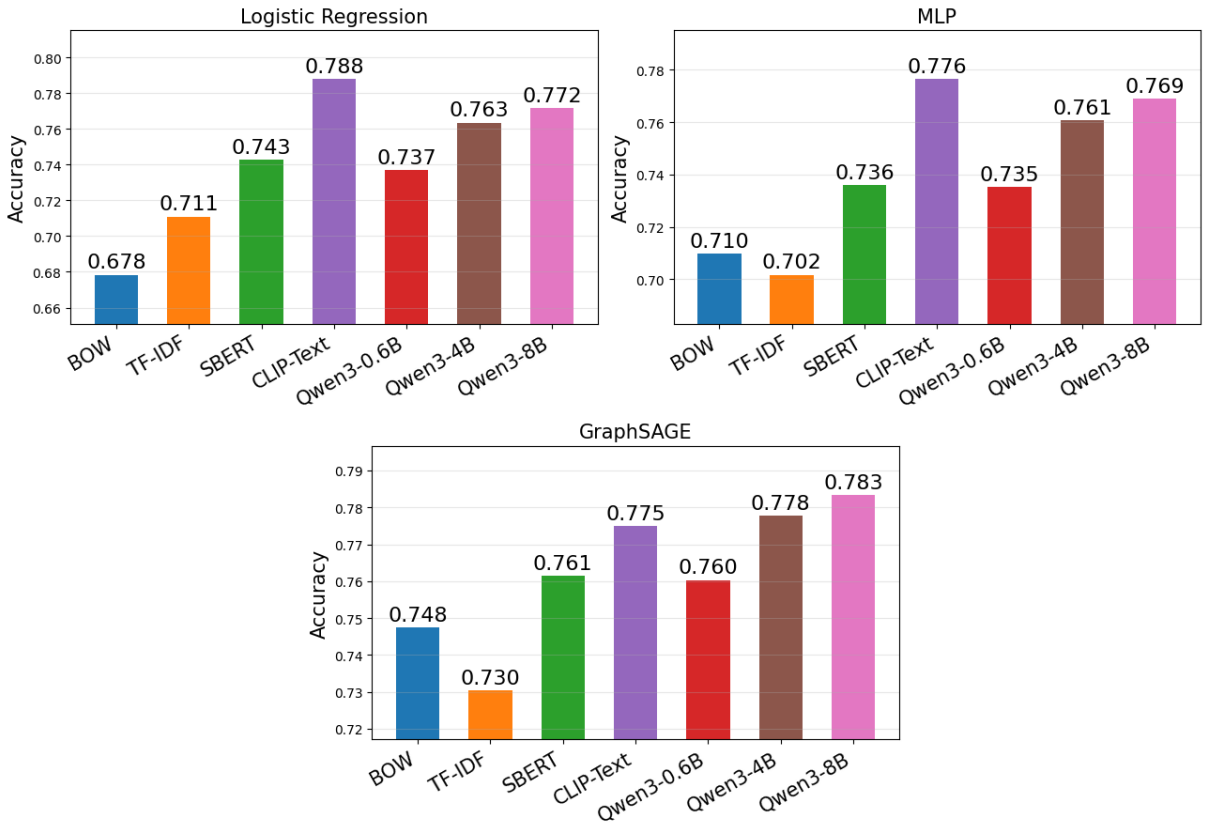


Figure 1: Accuracy over different models and encoding methods.

Additionally, a comparative analysis of the training trajectories is illustrated in Figure 2, contrasting the convergence behaviors of the `MLP` and `GraphSAGE` frameworks. Both architectures exhibit remarkably similar optimization curves, demonstrating comparable convergence velocities despite the added topological message-passing overhead inherent to the graph-based model.

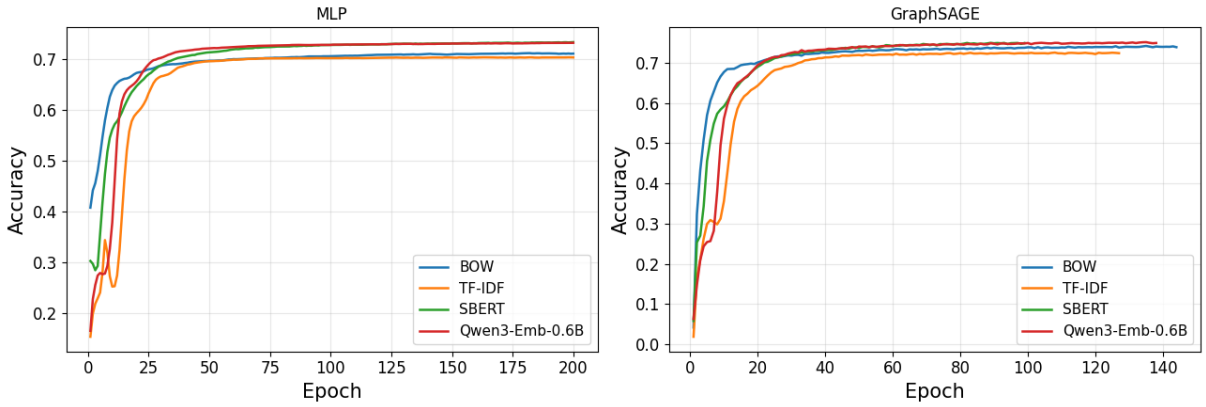


Figure 2: Epoch evolution over MLP and GraphSAGE.

3.2.2 Image pipeline results

Three vision encoders were evaluated as node features for `GraphSAGE`: `ResNet-50`, `CLIP ViT-B/32`, and `DINOv2-Base`. Table 2 reports the classification accuracy using image features only.

Table 2: Image pipeline: GraphSAGE accuracy with image-only features.

Encoder	Test Acc
ResNet-50	0.7080
CLIP ViT-B/32	0.7368
DINOv2-Base	0.7354

All three vision encoders achieve competitive accuracy, with `CLIP` and `DINOv2` performing similarly and both outperforming `ResNet-50`. This confirms that modern self-supervised vision encoders extract richer product representations than the classic CNN architecture.

3.2.3 Multimodal fusion results

To evaluate **H2**, two multimodal combinations were tested using simple concatenation of text and image embeddings as node features for `GraphSAGE`: `CLIP-text + CLIP-image` (1024 dimensions) and `BERT-Large + DINOv2` (1536 dimensions). The convergence curves for both configurations are illustrated in Figures 3 and 4.

Table 3: Multimodal fusion: GraphSAGE with concatenated text and image embeddings.

Text encoder	Image encoder	Val Acc	Test Acc
CLIP-text	CLIP-image	0.7830	0.7857
BERT-Large	DINOv2	0.7668	0.7675

Both multimodal configurations converge smoothly and achieve competitive results. The `CLIP-text + CLIP-image` combination reaches 78.6% test accuracy, benefiting from the shared embedding space in which CLIP text and image representations are jointly trained — making their concatenation particularly coherent. The `BERT-Large + DINOv2` combination achieves 76.8%, combining the strongest text encoder with the most expressive vision model. In both cases, the multimodal features provide a richer input representation than either modality alone, offering complementary signals that the `GraphSAGE` backbone can exploit alongside the co-purchase graph structure.

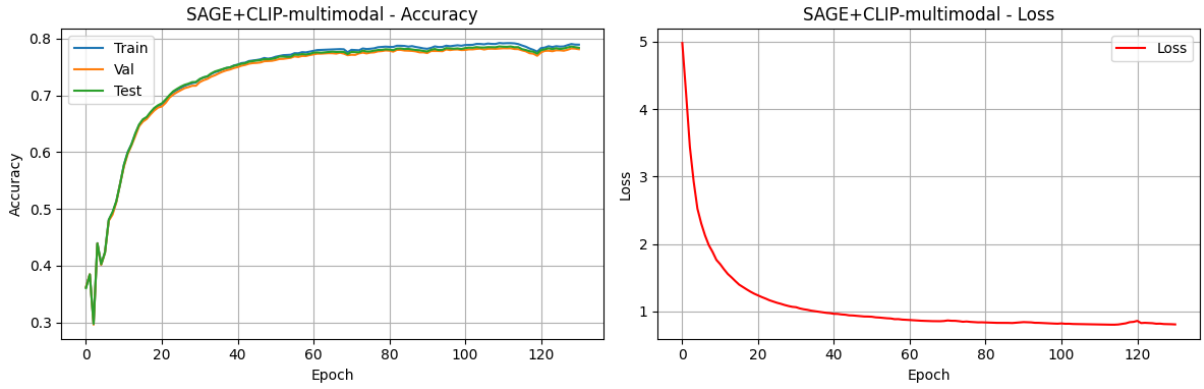


Figure 3: Training curves for GraphSAGE with CLIP text + CLIP image features.

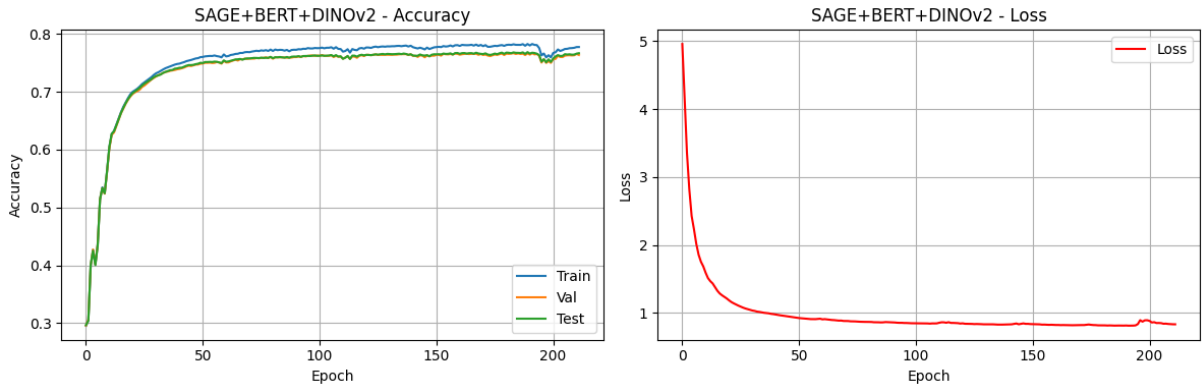


Figure 4: Training curves for GraphSAGE with BERT-Large + DINOv2 features.

3.3 Cost analysis

To comprehensively evaluate the operational viability of the classification framework, a systematic accuracy-versus-cost trade-off analysis was performed. While the final classification accuracies achieved by each configuration are presented in Figure 1, a true determination of efficiency requires examining computational overhead along two distinct metrics: text encoding runtime and training execution time. Specifically, Figure 5 quantifies the initial encoding duration required by each feature extraction method to process the text dataset, illustrating the substantial resource variance between traditional statistical descriptors and deep semantic embedding mechanisms.

Consequently, Figure 5 highlights how the initial text processing duration scales across the different configurations, showing that the `Qwen3` encoding pass demands significantly more execution time than the other methods. This pronounced difference is an expected trade-off for the deep contextual representations generated by a over a half-billion parameter transformer architecture. Meanwhile, the baseline statistical encoders exhibit similar execution profiles, while `sBERT` represents a moderate compromise, incurring noticeable encoding overhead but remaining significantly faster than `Qwen3`.

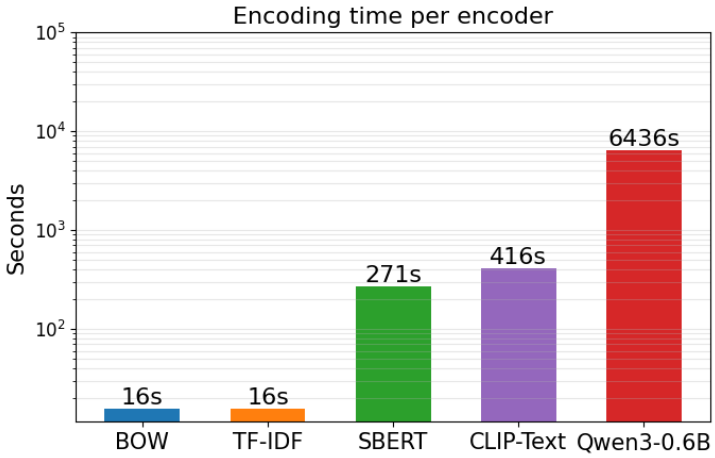


Figure 5: Encoding time for each algorithm.

Complementing this, Figure 6 delineates the downstream model training times recorded for each distinct encoding-model pair. Evaluating these dimensions, alongside the Figure 2 that depicts the epoch and accuracy evolution, reveals a clear computational trade-off: while the graph model generally offer superior predictive capacities (even in the first few epochs), its time complexity can scale significantly with feature dimensionality.

Notably, `sBERT` incurs a high training runtime across all models. This behavior can be attributed to the model's dense feature matrix, which increases the computational overhead during message-passing aggregation.

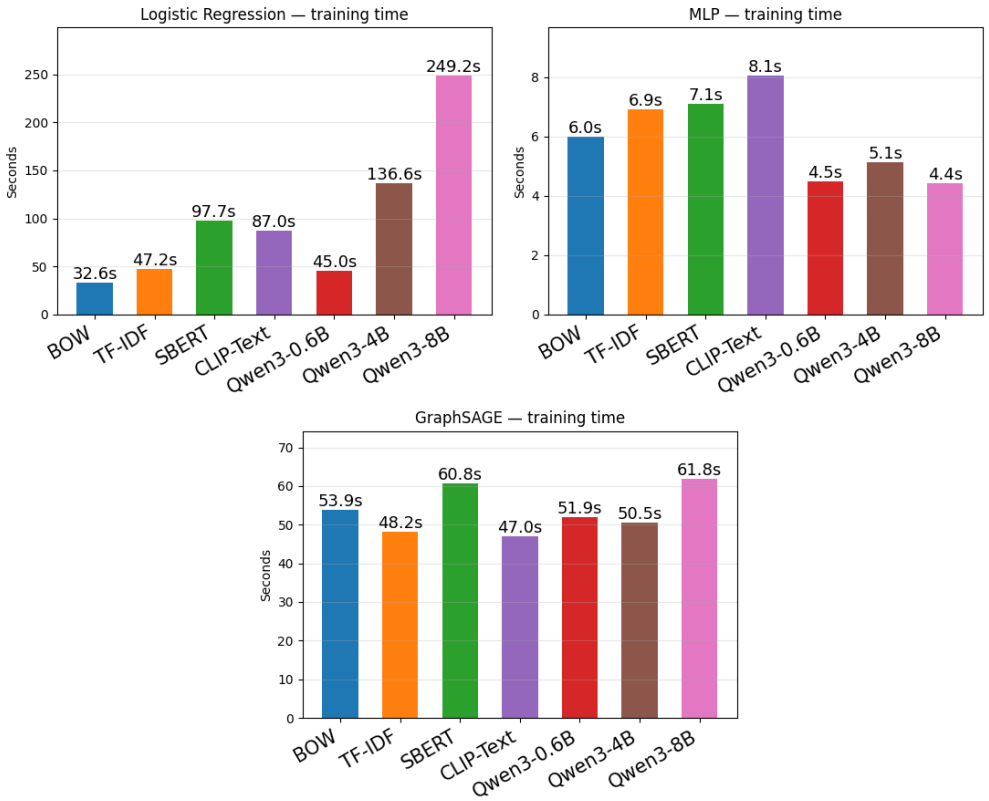


Figure 6: Training duration for each encoding-model pair.

4 Alignment with the Proposed Plan

4.1 Was the planning followed?

The core of the proposed plan was delivered. It was built a reproducible end-to-end pipeline that runs from a raw public Amazon dataset to a co-purchase graph, to cached node features, to a trained and evaluated GNN, with deterministic splits, seed control, and five-seed statistical reporting throughout. On top of it, all three planned comparisons were ran: graph vs. no-graph (**H1**), a multi-modal text+image study with three fusion strategies (**H2**) and a systematic text-encoder benchmark spanning classical and modern encoders (**H3**). The text-encoder axis in particular was explored thoroughly, including the pretraining-objective and model-size sub-axes.

4.2 What could have been done differently, and lessons learned

Future studies could introduce critical adjustments to the evaluation pipeline, notably through a comprehensive hyperparameter tuning process. Optimizing key parameters such as hidden dimensions, learning rates, and graph aggregation steps will help identify the exact operational boundaries where advanced language model encoding allow `GNNs` to significantly outperform baseline methods. This will provide a clearer road map regarding the trade-offs between computational overhead and structural predictive gains.

Furthermore, future work could explore more advanced graph architectures to overcome current relational limitations, enabling the integration of edge types beyond `also_bought` such as `also_viewed` or `bought_together`.

5 Code Availability

The source code and implementation details of our project are publicly available at: https://github.com/guilhermeRaibolt/gnn-embeddings.

6 Contributions of Team Members

6.1 Individual contributions

- **Adrien Bertrand:** Implemented and evaluated a baseline experiment to establish a comparative performance benchmark across multiple encoder-based models. Visualization of experimental results and comparison of the performance. Communication with the project supervisor.
- **Ala Ellefi:** Built the co-purchase graph from Amazon Toys & Games metadata and evaluated the impact of text encoder quality on GraphSAGE node classification performance, comparing TF-IDF, BERT (small and large), and CLIP text encoders. Extended the pipeline to multimodal settings by combining CLIP text embeddings with DINOv2 image features.
- **Matheus Schreiber:** Implementation and evaluation of one of the textual processing infrastructures and the baseline non-graph evaluation frameworks. Construction of non-

graph baseline experiments to establish a comparative benchmark across all encoder families. Systematic synthesis, data visualization plotting, and comprehensive drafting of the technical report detailing empirical findings.

- **Guilherme Efgen:** Implementation of the image embedding extraction pipeline and storage of the resulting embeddings in a compatible format. Construction and evaluation of the image-only node classification experiments across the different vision encoders.
- **Qingyue Deng:** Implementation and evaluation of text embeddings, training of both graph-based and non-graph neural networks. Visualization of experimental results and comparison of the performance across different text encoders.
- **Yuang Ding:** Conducted a systematic study of hyperparameter optimization for both graph-based and non-graph models. Performed exploratory analysis of dataset characteristics and feature distributions to support model design decisions. Contributed to the preparation of the final presentation, and drafting, revision, and refinement of the final technical report.

6.2 Teamwork and tools

The team coordinated through a shared GitHub repository for all code (with issues and pull requests for review), a shared Overleaf project for this report, and a WhatsApp group chat for day-to-day communication. Also, the supervisor, Clément Wang, was regularly consulted during the study period.

7 Use of AI Tools

7.1 In the code

Various state-of-the-art Large Language Models (LLMs) were utilized throughout the life cycle of this study, serving in a consultative capacity to assist with algorithmic design, visualization plotting, and optimization refactoring. Despite leveraging these models to streamline development and explore architectural enhancements, the core code base was authored, curated, and validated by the team.

7.2 In the report

Additionally, the consultative capacity of these models was extended to the refinement of the present manuscript, assisting with rephrasing and linguistic clarification to ensure a more cohesive and readable scientific text.

References

[1] Julian McAuley. Amazon product data: Co-purchase links and metadata. https://cseweb.ucsd.edu/~jmcauley/datasets/amazon/links.html, 2014. Accessed: 2026-06-16.

[2] Netflix Research. Synergistic signals: Exploiting co-engagement and semantic links via graph neural networks. https://research.netflix.com/publication/synergistic-signals-exploiting-co-engagement-and-semantic-links-via-graph, December 2023. Accessed: 2026-06-16.

[3] Amazon Science. Using graph neural networks to recommend related products. https://www.amazon.science/blog/using-graph-neural-networks-to-recommend-related-products, April 2021. Accessed: 2026-06-16.

[4] Fedor Borisyuk, Shihai He, Yunbo Ouyang, Morteza Ramezani, Peng Du, Xiaochen Hou, Chengming Jiang, Nitin Pasumarthy, Priya Bannur, Birjodh Tiwana, Ping Liu, Siddharth Dangi, Daqi Sun, Zhoutao Pei, Xiao Shi, Sirou Zhu, Qianqi Shen, Kuang-Hsuan Lee, David Stein, Baolei Li, Haichao Wei, Amol Ghoting, and Souvik Ghosh. Lignn: Graph neural networks at linkedin, 2024. URL https://arxiv.org/abs/2402.11139.

[5] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, 24(5):513–523, 1988.

[6] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830, 2011.

[7] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, UK, 2008.

[8] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

[9] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992. Association for Computational Linguistics, 2019.

[10] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning (ICML)*, pages 8748–8763. PMLR, 2021.

[11] Qwen Team. Qwen3 technical report. *arXiv preprint*, 2025.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778. IEEE, 2016.

[13] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khali-

dov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

[14] John Arevalo, Thamar Solorio, Manuel Montes-y Gómez, and Fabio A. González. Gated multimodal networks. *Neural Computing and Applications*, 32(14):10209–10228, 2020.

[15] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[16] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, pages 1024–1034. Curran Associates, Inc., 2017.