Attribute-Value Prediction From E-Commerce Product Descriptions

Neel Shah[*1,†] **and Sneh Shah**[†1,†]

[1] *Charotar University of Science and Technology*
[†] *These authors contributed equally to this work*

This manuscript was compile on November 3, 2024

Abstract

This study presents a detailed comparative analysis of five advanced methodologies for automated attribute prediction in e-commerce catalogs, addressing the challenge of extracting and predicting attribute-value pairs from unstructured product descriptions. Our key contributions include: (1) a Multi-Learning Hierarchical Transformer (MLHT) that integrates supervised, reinforcement, and contrastive learning techniques to model complex relationships across Super Group, Group, Module, and Brand attributes; (2) an Expert-Ensemble Multi-Learning Hierarchical Transformer (EE-MLHT) that leverages knowledge distillation and focal loss to handle class imbalance and enhance knowledge transfer between attribute levels; (3) a Selective Pathway Transformer Classifier (SPTC) designed to optimize hierarchical classification pathways; (4) a series of agentic models, including the Description Generation Agent (DDAAC), Smart Search-Driven Agentic Classification (SSDAC), and Hierarchical Option-Based Agentic Classification (HOAC), which simulate human-like reasoning in attribute classification; and (5) a Semantic Search-Driven Agentic Classification (SSDAC) model for enhancing classification accuracy via semantic similarity. Although we have made efforts to achieve the best possible results, computational limitations have prevented the models from being trained to full convergence. This leaves potential for further optimization and convergence under higher computational resources. Experimental evaluations across a diverse dataset reveal the strengths and unique advantages of each approach. This comprehensive study provides critical insights into combining multiple learning paradigms to address hierarchical classification complexities in e-commerce, with detailed implementation strategies included in the technical report.

Keywords: *E-commerce, Product Classification, Hierarchical Learning, Knowledge Distillation, Reinforcement Learning, Contrastive Learning, RAG, Agentic AI*

Corresponding author: Neel Shah *E-mail address:* neeldevenshah.ai@gmail.com

1. Introduction

The explosive growth of e-commerce has fundamentally transformed the retail landscape, with major platforms now managing vast catalogs containing billions of products. This scale presents a significant challenge in maintaining structured product metadata, particularly attribute-value pairs, which are crucial for various downstream applications. While product descriptions are readily available from receipts, invoices, and seller submissions, converting this unstructured text into standardized attribute-value pairs remains a complex challenge that impacts search relevance, recommendation systems, and overall customer experience.

Automated attribute prediction in e-commerce is especially challenging due to several inherent complexities. First, the hierarchical nature of product categorization, spanning from broad Super Groups down to specific Brands, requires models to understand both macro-level category relationships and micro-level product distinctions [2]. Second, the dynamic nature of e-commerce means that new products, brands, and categories emerge continuously, necessitating models that can handle previously unseen values. Third, the variation in product description quality and format across different retailers adds another layer of complexity to the extraction process.

Previous approaches to this problem have often struggled to capture the hierarchical structure of attributes and the need for both broad category understanding and fine-grained distinction [2], [1]. Moreover, they frequently lack the adaptability to leverage knowledge relationships across different attribute levels, limiting their ability to manage new product categories and brands.

Our research addresses these limitations through a multi-faceted approach that combines five distinct methodological frameworks. The cornerstone of our work is an advanced hierarchical classifier that integrates supervised, reinforcement, and contrastive learning techniques to capture complex relationships between attribute levels. This is complemented by an ensemble of expert classifiers that employ knowledge distillation and focal loss to tackle the inherent challenges of hierarchical classification.

We further innovate by introducing agentic approaches, both with and without internet connectivity, that simulate human-like reasoning in product categorization. Our implementation of a Retrieval-Augmented Generation (RAG) system leverages external knowledge bases to enhance the model's understanding of product relationships and brand hierarchies, providing additional contextual support for accurate classification.

This comprehensive approach not only addresses the immediate challenges of attribute prediction but also contributes to the broader field of hierarchical text classification and knowledge extraction. Our research has significant implications for e-commerce platforms seeking to automate product categorization processes while maintaining high accuracy and adaptability to new products and categories.

The remainder of this paper is organized as follows: Section 2 provides an overview of the Multi-Learning Hierarchical Transformer (MLHT). Section 3 describes the Expert-Ensemble Multi-Learning Hierarchical Transformer (EE-MLHT). Section 4 introduces the Selective Pathway Transformer Classifier (SPTC). Section 5 discusses the agent-based approaches, including the Description Generation Agent (DDAAC), Smart Search-Driven Agentic Classification (SSDAC), and Hierarchical Option-Based Agentic Classification (HOAC). Section 6 introduces the Semantic Search-Driven Agentic Classification (SSDAC). Section 7 details the Experimental Infrastructure and Training Strategy. Section 8 highlights key Learnings and Insights. Finally, Section 9 concludes with key findings and future research directions.

2. Multi-Learning Hierarchical Transformer (MLHT)

This section presents an integrated hierarchical classifier that can utilize one based on a DeBERTa and the other on RoBERTa-XML, each tailored for multi-level classification tasks. The core of each architecture features cascaded classifiers and reinforcement learning-based policy networks, structured to predict multiple hierarchical levels accurately. Both models incorporate contrastive learning for class

*Email: neeldevenshah.ai@example.com
†Email: 22aiml049@charusat.edu.in

separation and a few-shot learning mechanism based on prototype distance for adaptability. A joint trainer optimizes the model using a multi-objective loss to ensure computational efficiency. The primary distinction between the two versions lies in the choice of backbone model, DeBERTa or RoBERTa-XML.

2.1. Rationale for backbone 1: Pre-Trained DeBERTa Model

The disentangled attention mechanism is a key innovation in DeBERTa, designed to separate content and positional information for each word [8]. Traditional models like BERT and RoBERTa combine content and position into a single vector, limiting the model's ability to capture nuanced relationships. In contrast, DeBERTa represents each word with two distinct vectors:

- Content Vector: Encodes the semantic meaning of the word, capturing nuances and relationships relevant to understanding the text.
- Position Vector: Encodes the position of each word within the sentence, essential for interpreting word order and structure.

By separating these two components, DeBERTa enhances its attention mechanism. The model computes attention weights using disentangled matrices, which process content and positional vectors independently. This design allows DeBERTa to better capture contextual dependencies and word order, improving comprehension in tasks like sentence classification and language inference.

2.2. Rationale for backbone 2: Pre-Trained XLM-RoBERTa Model

XLM-RoBERTa builds upon the RoBERTa architecture, leveraging advancements for robust cross-lingual performance. Key architectural features include [13]:

- **Transformer Layers**: XLM-RoBERTa incorporates multiple transformer layers, each containing self-attention mechanisms and feed-forward networks. This structure enables the model to effectively capture both local and global dependencies within the input text.
- **Embedding Layers**: Starting with embedding layers, XLM-RoBERTa converts input tokens into continuous vectors. It utilizes Byte Pair Encoding (BPE) to handle subword units, effectively managing out-of-vocabulary words and capturing morphological nuances across multiple languages.
- **Masked Language Modeling (MLM)**: Using a masked language modeling objective, XLM-RoBERTa masks random words in sentences and trains the model to predict these masked tokens based on surrounding context. This bidirectional approach strengthens the model's grasp of language semantics, making it highly effective for multilingual NLP tasks.

2.3. Hierarchical Classifiers

The model performs classification across four hierarchical levels: supergroup, group, module, and brand. For each level, a fully connected classifier is used with input features combining the hidden states and softmax-normalized logits from the previous level's output [5]. This cascading input strategy helps preserve hierarchical dependencies, making higher levels aware of lower-level information.

- **Supergroup Classifier**: Takes H as input.
- **Group Classifier**: Takes concatenated features of H and supergroup logits.
- **Module Classifier**: Takes concatenated features of H, supergroup logits, and group logits.
- **Brand Classifier**: Takes concatenated features of H, supergroup logits, group logits, and module logits.

The input to each classifier at level ℓ can be formalized as:

$$\ell = \text{concat}(H, \text{softmax}(\text{Logits}_1), \ldots, \text{softmax}(\text{Logits}_{\ell-1}))$$

This sequential concatenation allows each classifier to access increasingly refined information as the hierarchy progresses. The hierarchical classifier operates across four distinct levels: supergroup, group, module, and brand. Each level has a fully connected (FC) layer with parameters specific to that level, taking concatenated features from previous levels' logits to create dependencies across the hierarchy.

Let:

- H: denotes the hidden state from DeBERTa.
- Logits_ℓ: denotes the logits output at the ℓ-th level.
- $\text{softmax}(\text{Logits}_\ell)$: denotes the softmax-normalized logits for level ℓ.

Each classifier is structured as follows:

Here's the text with the formulas formatted as requested:

Supergroup Classifier: The supergroup classifier only depends on H and produces the initial level of logits:

$$\text{Logits}_1 = \text{FC}_1(H)$$

where FC_1 is the fully connected layer for the supergroup. The supergroup prediction $\hat{y}_1$ is calculated as:

$$\hat{y}_1 = \arg\max \text{softmax}(\text{Logits}_1)$$

Group Classifier: The group classifier takes both H and the softmax-normalized logits of the supergroup classifier as input, combining information from previous levels to make predictions at the group level. Formally:

$$\text{Logits}_2 = \text{FC}_2\left(\text{concat}\left(H, \text{softmax}(\text{Logits}_1)\right)\right)$$

where FC_2 is the fully connected layer for the group classifier.

Module Classifier: At the module level, the classifier takes H, supergroup logits, and group logits, progressively incorporating each level's output:

$$\text{Logits}_3 = \text{FC}_3\left(\text{concat}\left(H, \text{softmax}(\text{Logits}_1), \text{softmax}(\text{Logits}_2)\right)\right)$$

Brand Classifier: The brand classifier, which is the finest level of prediction, incorporates all previous levels. The resulting logits are:

$$\begin{aligned}\text{Logits}_4 = &\text{FC}_4\left(\text{concat}\left(H, \text{softmax}(\text{Logits}_1),\right.\right.\\ &\left.\left.\text{softmax}(\text{Logits}_2), \text{softmax}(\text{Logits}_3)\right)\right.\end{aligned}$$

The cascading design helps each classifier layer utilize refined information from preceding levels. Each classifier outputs logits that indicate the probability of each possible class at that level.

2.4. Reinforcement Learning Policies for Hierarchical Classifiers

Reinforcement learning (RL) policies are introduced for each hierarchical level to allow the model to make probabilistic decisions, enhancing its flexibility and potential real-time adaptability.

Each RL policy network π_l outputs a probability distribution over actions (possible classes) for the l-th level. These outputs can be interpreted as logits sampled according to an RL policy [12], [5]:

$$\pi_l(a \mid s) = \frac{\exp(\text{Logits}_{l,a})}{\sum_j \exp(\text{Logits}_{l,j})}$$

where a represents an action or class, s is the input state (concatenated hidden representations), and π_l returns a probability distribution over actions for level l.

Each policy network may have an associated reward function R_l used during training, defined as:

$$R_l = 1(y^l = y_l)$$

where y^l is the predicted class, and y_l is the true class label. This binary reward incentivizes accurate predictions and helps calibrate the model's probabilistic outputs.

2.5. Contrastive Learning via Projection Head

To enhance feature separation, we contrastive learning using a projection head. This head maps the embedding H to a lower-dimensional space optimized for contrastive learning. The primary goals are to improve inter-class separation and maintain intra-class compactness, ensuring that samples from the same class remain close together in the embedding space while pushing apart samples from different classes [14], [9].

The projection head consists of two sequential fully connected layers, which transform the initial embedding H as follows:

First Layer The first layer applies a fully connected transformation with ReLU activation:

$$z_1 = \text{ReLU}(W_1 \cdot H + b_1)$$

where W_1 and b_1 are the weights and bias, respectively. Here, $z_1 \in \mathbb{R}^{d'}$, with $d' < d$, where d is the dimensionality of H. This layer reduces dimensionality and allows non-linear transformations for better separation of similar samples.

Second Layer The second layer further reduces dimensionality to fit the requirements of contrastive learning:

$$z_2 = W_2 \cdot z_1 + b_2$$

where $z_2 \in \mathbb{R}^{128}$, representing the final projection used for contrastive learning.

Contrastive Loss The projection z_2 is used in a contrastive loss function to enforce proximity of related samples while repelling unrelated ones. Given projections $z_2(i)$ and $z_2(j)$ for two positive (related) samples, the contrastive loss is defined as:

$$L_{\text{contrastive}} = -\log\left(\frac{\exp\left(\text{sim}(z_2(i), z_2(j))/\tau\right)}{\sum_{k \neq i} \exp\left(\text{sim}(z_2(i), z_2(k))/\tau\right)}\right)$$

where sim denotes cosine similarity, and τ is a temperature parameter controlling separation strength.

Example and Absence of Anchors Typically, contrastive learning requires a positive pair (similar samples) and may include negative samples. In this approach, however, we do not employ traditional anchor-positive-negative triples. Instead, all samples in the batch are potential comparators, using $z_2(i)$ as a query compared against all other embeddings $z_2(k)$. This approach, which avoids predefined anchors, provides flexibility by dynamically choosing sample relationships in the batch and thus reduces dependency on predefined anchor identities. The batch-wise comparison creates a more generalized embedding space, where each sample's proximity to others defines the representation, and temperature τ controls the spread across classes.

For instance, if two samples belong to the same supergroup but different brands, contrastive learning pulls them closer in the supergroup space, while other brands are encouraged to be more distinct, creating a hierarchical structure in the learned representations.

2.6. Few-Shot Learning with Prototypes

In scenarios where classes have few samples, we use prototype-based few-shot learning to represent each class by a prototype vector, or "centroid," in the embedding space. This approach, known as Prototypical Networks, is effective for handling sparse class samples by generalizing from a small support set [4].

Prototype Definition For each class at a specific hierarchical level l, we define a prototype $P_{l,i}$ as the centroid of all available samples for that class. This prototype serves as the idealized representation for the class and is computed as the mean of the embeddings of its samples. Formally, for each level l:

$$P_l = \{P_{l,1}, \ldots, P_{l,k}\}$$

where $P_{l,i}$ represents the prototype of class i at level l, calculated as the average of its sample embeddings.

Classification Using Prototype Distances When a query sample with hidden representation H needs to be classified, the model computes its distance to each prototype $P_{l,i}$ within the relevant level l. The classification is based on the nearest prototype, with the distance serving as a negative logit in the few-shot classification:

$$L_{\text{FS}} = -\|H - P_{l,i}\|_2$$

The model thus assigns the query sample to the class with the closest prototype. Using negative distance as logits aligns with the goal of bringing related samples closer to their class prototype, making it easier to classify even in low-data settings.

Advantage of Prototypical Networks in Few-Shot Scenarios Prototypical Networks enable the model to adapt to new classes quickly, as adding a new class requires only computing an additional prototype. This flexibility is especially useful in hierarchical settings where different classification levels (e.g., supergroup, group, module, brand) require prototypes to capture similarities within each level. As classes expand, prototypes provide an efficient mechanism for hierarchical classification without extensive retraining, making this approach particularly suitable for real-world, few-shot applications.

2.7. Joint Training and Multi-Objective Loss

The model's training objective includes multiple loss terms, with the primary components being cross-entropy loss for supervised classification, contrastive loss, and optional reinforcement learning reward terms.

The objective for each level l is defined as:

$$L(l) = \sum_l \lambda_1 L_{CE,l} + \lambda_2 L_{contrastive,l} + \lambda_3 L_{RL,l} + \lambda_4 L_{FS,l}$$

Here, $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ are hyperparameters that balance the contributions of each loss component.

2.8. Data Preprocessing and Dataset Organization

Our framework employs a comprehensive data preprocessing pipeline that combines hierarchical label encoding with efficient dataset organization. For label encoding, we utilize separate `LabelEncoder` instances for each hierarchical level (Super Group, Group, Module, and Brand), converting categorical labels into unique integer representations that preserve the hierarchical relationships within the data.

The core of our data handling is implemented through a custom `ProductDataset` class that manages both text tokenization and hierarchical label processing. This class leverages transformer-based tokenizers to process product descriptions while maintaining the hierarchical structure of labels. The dataset class works seamlessly with PyTorch's DataLoader, enabling efficient batch processing where each data point consists of tokenized product descriptions paired with their corresponding encoded hierarchical labels.

2.9. Results and Ablation Studies

The Multi-Learning Hierarchical Transformer (MLHT) was evaluated over an extensive training period of more than 280 epochs, primarily utilizing the DeBERTa backbone model. Initially, a concatenated input of "description + retailer" (without spacing) was mistakenly

used as the input feature, which impacted the model's early learning dynamics due to the noise introduced by the unspaced combination. After around 100 epochs, this input was revised to "description only," resulting in a cleaner dataset and significantly improving feature clarity and overall accuracy.

In the main experiment with MLHT on the DeBERTa backbone, all loss components—including supervised, reinforcement, and contrastive—were integrated to capture complex relationships across hierarchical attributes like Super Group, Group, Module, and Brand. This configuration of MLHT with DeBERTa achieved an item accuracy of 0.4112, with CPU processing time at 919.646 ms and CUDA processing time at 277.029 ms. The training GPU FLOPS were recorded at 405.77, while inference per item took 0.7682 seconds with inference GPU FLOPS at 0.7820, demonstrating strong performance with full multi-learning integration.

Following this, two additional experiments were conducted to assess the impact of using hierarchical classifiers exclusively with each backbone model, isolating the hierarchical classification capability from the full MLHT setup.

The first experiment used the DeBERTa backbone with only hierarchical classifiers (excluding reinforcement and contrastive losses), achieved an item accuracy of 0.4087. This setup had a CPU time of 862.201 ms, a CUDA time of 265.606 ms, and training GPU FLOPS of 402.0172. Its inference per item took 0.2601 seconds, with inference GPU FLOPS at 2.7820. The second experiment evaluated XLM-RoBERTa with hierarchical classifiers only, which achieved an item accuracy of 0.3941. This setup had a CPU time of 1.202 s, a CUDA time of 391.745 ms, and training GPU FLOPS of 396.831. Its inference per item took 2.7820 seconds, with inference GPU FLOPS at 2.0872.

This comparative analysis highlights DeBERTa as the main backbone for MLHT with the full multi-learning configuration, and its results suggest that DeBERTa provides competitive performance even with hierarchical classifiers alone. Additionally, ablation studies indicated that "description only" as the input consistently outperformed configurations that included "description + retailer," supporting this as the optimal feature selection for hierarchical classification.

3. Expert-Ensemble Multi-Learning Hierarchical Transformer (EE-MLHT)

To further boost classification accuracy, an ensemble of expert classifiers is introduced at each hierarchy level. Each expert classifier specializes in a particular category level (supergroup, group, module, brand) and these classifiers operate alongside a main classifier and contribute additional predictions through various techniques [6]:

3.1. Expert Classifiers

Each expert classifier E_l for level l is structured with layers of linear transformations, normalization, and activation functions, operating independently while providing secondary predictions for each label level.

3.2. Knowledge Distillation

The outputs of the main classifier M_l are softened through knowledge distillation, aligning the expert classifier's output distributions with those of the primary classifier. This encourages the experts to learn from the main classifier's predictions, improving their generalization to new instances.

The logits from the main classifier at level l are denoted as z_{M_l}. The softened probability distribution is defined as:

$$p_{M_l} = \text{softmax}\left(\frac{z_{M_l}}{T}\right)$$

For the expert classifier E_l, its logits z_{E_l} yield softened predictions:

$$p_{E_l} = \text{softmax}\left(\frac{z_{E_l}}{T}\right)$$

The Knowledge Distillation Loss for expert E_l is defined as the Kullback-Leibler divergence between the softened predictions of M_l and E_l:

$$L_{KD,l} = KL\left(p_{M_l} \| p_{E_l}\right) = \sum_c p_{M_l}(c) \log\left(\frac{p_{M_l}(c)}{p_{E_l}(c)}\right)$$

3.3. Focal Loss for Hard Examples

In hierarchical classification tasks, the model may encounter examples that are difficult to classify accurately, especially in lower levels of the hierarchy. To focus learning on these challenging cases, we utilize focal loss [3]. Focal loss scales the standard cross-entropy loss to emphasize hard-to-classify instances, reducing the influence of easy examples.

The one-hot encoded ground truth label for class c at level l is denoted as $y_{l,c}$, and $\hat{y}_{l,c}$ is the predicted probability. The Focal Loss for level l is defined as:

$$L_{\text{focal},l} = -\sum_c y_{l,c}(1 - \hat{y}_{l,c})^\gamma \log(\hat{y}_{l,c})$$

where γ is the focusing parameter, which we set to $\gamma = 2.0$ to increase emphasis on misclassified instances (i.e., lower values of $\hat{y}_{l,c}$). In this way, the model prioritizes learning from instances that are harder to classify, enabling it to generalize better on such challenging cases within the hierarchical structure.

3.4. Confidence-Weighted Expert Loss

In our hierarchical classifier, each expert classifier's loss contribution is modulated by a **confidence score** w_l, reflecting the model's certainty at each level. This score is produced by a confidence head specific to each level, calculated as:

$$w_l = \sigma(z_{C_l})$$

where z_{C_l} is the logit from the confidence head, and σ represents the sigmoid activation. The Weighted Loss for each level's expert E_l combines the cross-entropy loss $L_{CE,l}$ and the knowledge distillation loss $L_{KD,l}$, calculated as:

$$L_{\text{weighted},l} = w_l \cdot L_{CE,l} + (1 - w_l) \cdot L_{KD,l}$$

This dynamic weighting mechanism enables the model to balance the expert classifiers' predictions with its own, emphasizing confident predictions while down-weighting uncertain ones.

3.5. Joint Loss for Ensemble Expert Classifiers

The total loss L_{total} for the ensemble integrates all classifiers and expert components across each hierarchical level. It is a weighted sum of the various loss components:

$$L_{\text{total}} = \sum_l \lambda_1 L_{CE,l} + \lambda_2 L_{\text{focal},l} + \lambda_3 L_{KD,l} + \lambda_4 L_{\text{weighted},l}$$

Here, $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ are hyperparameters that balance the contributions of each loss component.

Each component serves a distinct purpose:

- Cross-Entropy Loss: Guides overall prediction accuracy.
- Focal Loss: Improves robustness on difficult instances.
- Knowledge Distillation Loss: Aligns experts with the primary classifier.
- Confidence-Weighted Loss: Encourages reliance on confident predictions, thereby improving ensemble accuracy.

By leveraging these techniques, the ensemble model maximizes accuracy through a thoughtful integration of multiple classifiers, focusing on hard examples and using confidence-weighted approaches to refine predictions.

3.6. Initial Prediction Generation

The Final Decision Process in the hierarchical classification model combines predictions from main classifiers and expert classifiers, guided by a series of decision rules. Each prediction level (supergroup, group, module, brand) is determined by comparing the outputs of the main classifier and expert classifier, using confidence and probability thresholds. Here's a detailed breakdown of how this process operates:

First, the `predict` function generates preliminary predictions, confidence scores, and probabilities for each classification level. These predictions are derived as follows:

- **Main Classifier Predictions**: For each level, the main classifier produces logits, from which the most probable class (using `torch.argmax`) is selected.
- **Expert Classifier Predictions**: The expert classifiers, tuned with knowledge distillation, provide an independent set of predictions for each level.
- **Confidence Scores and Probabilities**: The model's confidence in its prediction for each level is obtained from a separate confidence head, and probabilities for each level's prediction are calculated using the softmaxed logits of the main classifier.

3.7. Reconciliation Process for Final Prediction

The reconciliation process compares main and expert predictions across each level to produce a single, refined prediction. For each level (supergroup, group, module, brand), the reconciliation process follows these steps

1. **Consensus Check** If the main and expert classifiers agree on a prediction, this consensus prediction is accepted directly.
2. **Confidence and Probability-Based Decision** If there is disagreement between the main and expert predictions, confidence and probability thresholds determine the final prediction:

$$C_{\text{thresh}} = 0.8$$
$$P_{\text{thresh}} = 0.7$$

   - **Case 1**: If both the main classifier's confidence and probability exceed these thresholds

$$C \geq C_{\text{thresh}}$$
$$P \geq P_{\text{thresh}}$$

     the main classifier's prediction is selected as it indicates a strong belief in the prediction's correctness.
   - **Case 2**: If either the confidence or probability of the main classifier falls below the threshold

$$C < C_{\text{thresh}} \quad \text{or} \quad P < P_{\text{thresh}}$$

     the expert classifier's prediction is used instead. This switch occurs because the main classifier's prediction lacks sufficient certainty, and the expert prediction is preferred as a more reliable fallback.

3. **Weighted Ensemble Decision** If the main prediction has moderate confidence and probability scores, neither high enough to take priority nor low enough to switch to the expert, a weighted ensemble approach is applied:

$$W_{\text{main}} = \frac{C + P}{2}$$

   Here, W_{main} represents the average of confidence and probability. If this average exceeds 0.5, indicating reasonable confidence, the main classifier's prediction is selected. Otherwise, the expert's prediction is chosen.

This ensemble approach balances confidence in the main prediction against the reliability of the expert prediction, ensuring a robust and adaptable final classification across all levels.

3.8. Results and Ablation Studies

The Expert-Ensemble Multi-Learning Hierarchical Transformer (EE-MLHT) was tested across several configurations, with the "description only" input proving to be the most effective. This configuration yielded the best item accuracy of 0.3731, achieving a total CPU time of 644.858 ms, CUDA time of 350.747 ms, GPU FLOPS during training at 412.0799, inference time per item of 0.62638 seconds, and inference GPU FLOPS at 90.2679. The "description only" approach allowed the model to maximize item accuracy by reducing noise, outperforming alternatives with additional features.

Two additional experiments were conducted. First, a "description + space + retailer" setup, which included retailer information, performed worse than the "description only" configuration. The added complexity from retailer data detracted from accuracy, confirming that streamlined inputs were optimal.

Second, we ran an experiment similar to the HOAC explained in detail in section 5.4 setup, which unfortunately yielded the lowest item accuracy at 0.1160. This result highlighted the limitations of the pathway-like approach for EE-MLHT, underscoring the importance of a focused, description-based input for optimal performance.

These results affirmed that using "description only" as the input feature was the most effective strategy for EE-MLHT, enhancing both accuracy and computational efficiency.

4. Selective Pathway Transformer Classifier (SPTC)

The hierarchical classification model for multi-level product classification is designed to predict labels across four levels—Supergroup, Group, Module, and Brand—using a series of transformer-based models with individual classifier heads. Each classifier is constrained by hierarchical mappings, enabling consistency across levels and allowing the model to handle dependencies effectively.

4.1. Rationale for backbone: Pre-Trained BERT-of-Theseus Model

BERT-of-Theseus offers an efficient solution to multi-class hierarchical classification in NLP by balancing model compression with high performance. Key motivations include [11]:

- **Efficiency in Deployment**: The model reduces computational costs through Progressive Module Replacing, making it suitable for resource-limited environments by progressively substituting BERT's components with compact modules.
- **Enhanced Hierarchical Learning**: Direct module replacement fosters an understanding of class hierarchies, improving predictions in parent-child relationships without complex loss functions or distillation overhead.
- **Adaptability to Class Imbalance**: BERT-of-Theseus adapts to both over- and underrepresented classes, preserving key hierarchical information to improve generalization across diverse class distributions.

In sum, BERT-of-Theseus provides a streamlined yet effective approach for tackling the complexities of hierarchical NLP classification.

4.2. Model Architecture

Here's the rewritten version with corrected formatting for each mathematical component:

Base Transformer Architecture A pre-trained transformer model (BERT-of-Theseus-MNLI) is used as the backbone for feature extraction across all levels. Given an input sequence

$$X = \{x_1, x_2, \ldots, x_n\}$$

the transformer generates contextual embeddings

$$H = \{h_1, h_2, \ldots, h_n\}.$$

The final hidden state h_n is passed to classifier heads designed for each hierarchical level.

Classifier Heads Let C_i represent the classifier for level i (where $i \in$ {Supergroup, Group, Module, Brand}). Each classifier head C_i is a fully connected layer with weights W_i and biases b_i. For each hierarchical level, the model outputs logits as:

$$z_i = h_n W_i + b_i$$

where z_i represents the logits for level i, capturing level-specific features.

Hierarchical Mappings and Constraints Hierarchical mappings are defined as:

$$\text{Supergroup} \rightarrow \text{Group} \rightarrow \text{Module} \rightarrow \text{Brand}.$$

We use mapping matrices $M_{SG\rightarrow G}$, $M_{G\rightarrow M}$, and $M_{M\rightarrow B}$ to constrain predictions for each level. These mappings restrict the logits at each level to only valid subclasses, enforcing hierarchical consistency across predictions.

For example, if the model predicts Supergroup SG_j, then the Group classifier C_{Group} only considers Groups that belong to SG_j using $M_{SG\rightarrow G}$. Mathematically:

$$z_{\text{Group}} = z_{\text{Group}} \odot M_{SG\rightarrow G}(SG_j)$$

where $\odot$ represents element-wise multiplication, effectively masking out invalid group logits.

4.3. Pathway Loss Function

A custom Pathway loss function is used to handle constraints based on valid indices within the hierarchy. This ensures that only valid labels contribute to the loss calculation, reducing penalization when predictions fall outside the valid scope of categories.

Let:

- l_i: Predicted label at level i
- t_i: True label at level i
- V_i: Set of valid indices for level i
- α_i: Weight for each level i

The Pathway loss can be defined as:

$$L = \sum_{i=1}^{N} \alpha_i \cdot \text{CrossEntropy}(l_i, t_i) \cdot \delta(l_i \in V_i)$$

Where CrossEntropy is calculated as:

$$\text{CrossEntropy}(l_i, t_i) = -\sum_{k=1}^{C_i} t_{ik} \log(l_{ik})$$

and where $\delta(l_i \in V_i)$ is equal to 1 if l_i is in the valid set V_i (indicating that the prediction is valid) and 0 otherwise. The loss function sums up the cross-entropy loss for each level, ensuring that only valid predictions—those that are included in the set of valid indices—are counted towards the total loss. The weights α_i enable the adjustment of the importance of each level in the hierarchy, allowing for a more nuanced contribution to the overall loss calculation.

4.4. Results and Ablation Studies

The Selective Pathway Transformer Classifier (SPTC) was evaluated using a single configuration: "description + space + retailer" as the input feature. This setup resulted in an item accuracy of 0.3454, indicating moderate performance but not as high as other models that focused solely on the product description. SPTC recorded a total CPU processing time of 995.420 ms, CUDA time of 656.425 ms, GPU FLOPS during training at 654.01557, and inference time per item of 0.6291 seconds with GPU FLOPS at 11.92008.

5. Dynamic Agentic Framework for Structured Classification

In this approach, a multi-agentic approach was employed to tackle the hierarchical classification of product descriptions, using a series of specialized agents that interact sequentially. This approach breaks down complex classification tasks into manageable, distinct stages, where each agent addresses a specific sub-task. This structured workflow allows the pipeline to achieve finer granularity in product classification, with each agent using the outputs of its predecessors to refine its own predictions.

5.1. Rationale for backbone : Llama3.1:8b-instruct-q8 Model

Llama3.1:8b-instruct-q8 is selected for its advanced language generation capabilities, extensive knowledge base, and technical efficiency, all of which are essential for multi-class hierarchical tasks. Key motivations include [15]:

- **Advanced Language Generation:** Generates coherent, contextually relevant text for precise hierarchical labeling; handles nuanced class boundaries effectively; produces high-quality outputs for multi-level differentiation.
- **Extensive Knowledge Base:** Pre-trained on over 15 trillion tokens for broad domain coverage; develops deep understanding of language for accurate labeling; supports eight languages for cross-linguistic applications.
- **Technical Efficiency and Scalability:** Features a 128K context window for extensive document processing; uses grouped query attention for improved inference speed; leverages instruction fine-tuning and few-shot learning for high precision with minimal examples.

5.2. First Approach: Dynamic Description-Driven Agentic Classification (DDAAC)

5.2.1. Description Generation Agent

The first approach began with a Description Generation Agent, responsible for expanding abbreviated product descriptions into detailed narratives. Given the cryptic nature of many labels, this agent used Llama3.1 to interpret the label's intended meaning and generate a comprehensive description. For example, "1 scrnwsh" was expanded into "A 1-liter container of windshield screen wash fluid for automobiles" [7]. Such transformations allowed subsequent agents to operate on clear and contextually rich information, rather than vague or underspecified product labels.

With Llama3.1, the Description Generation Agent was able to incorporate subtle contextual hints into descriptions, greatly enhancing interpretability. However, the model occasionally struggled with highly niche or unfamiliar terms, generating descriptions that were more generalized or slightly inaccurate. This variability in handling unseen products, although relatively rare, presented challenges for downstream agents that relied on exact descriptions.

5.2.2. Classification Agents

After generating expanded descriptions, a series of Classification Agents categorized products across four levels independently:

- **Supergroup Classification Agent:** Based on the expanded description, this agent selected an appropriate Supergroup category from 32 options, such as "Automotive Products."
- **Group Classification Agent:** Using the description, this agent assigned the product to one of 228 Groups, with categories like "Car Care" within a Supergroup such as Automotive.
- **Module Classification Agent:** The Module agent categorized the product's specific function, selecting from 449 Module categories based on intended use, such as "Cleaning Products" for a screen wash.

- **Brand Classification Agent:** Lastly, this agent identified the product's brand from over 5,000 available brand options, finalizing the categorization process.

Each agent leveraged Llama3.1's comprehensive language processing capabilities to classify products based on distinct classification criteria. The hierarchical breakdown enabled each agent to focus on specific details, minimizing errors in classification.

5.3. Second Approach: Smart Search-Driven Agentic Classification (SSDAC)

5.3.1. Google API Agent for Description Retrieval

The second approach replaced the Description Generation Agent with a Google API Agent that queried Google's search engine to obtain accurate, pre-existing product descriptions. By utilizing Google's API, this agent sourced reliable product data, benefiting from Google's comprehensive autocorrect and ranking algorithms. These capabilities effectively addressed the challenge of deciphering cryptic product labels, often returning corrected descriptions from high-ranking e-commerce sites like Amazon. This method circumvented some of the limitations associated with Llama3.1's training data coverage, providing high-quality descriptions for products outside the LLM's domain knowledge.

This Google API Agent processed each product label by sending targeted search queries and retrieving preview text from search results. This minimized inaccuracies by using authoritative descriptions from trusted sources, particularly useful for labels with abbreviations or misspellings. The agent's access to Google's ranking algorithms also ensured that top-ranked results contained the most relevant descriptions, enhancing overall classification accuracy.

5.3.2. Hierarchical Classification Agents

Once the Google API Agent provided detailed descriptions, the same Hierarchical Classification Agents (Supergroup, Group, Module, and Brand) were used to classify products through the multi-stage pipeline, same as that mentioned for the first approach.

This Google API-based approach improved classification accuracy significantly due to the dependable descriptions sourced from Google.

5.4. Third Approach: Hierarchical Option-Based Agentic Classification (HOAC)

In addition to the generative approach, Hierarchical Option-Based Agentic Classification (HOAC) was developed to enhance classification accuracy by incorporating hierarchical option filters at each stage of the classification process. Unlike traditional approaches, where all possible classes are considered, HOAC restricts options based on previously selected categories, guiding each agent through a progressively narrowed set of relevant choices. This structure is reinforced by two hierarchical configurations: one with Supergroup as the primary category and an alternate Brand-Based hierarchy.

5.4.1. Hierarchical Option Filtering

The HOAC system initiates classification with the Supergroup Selection Agent. Upon choosing a Supergroup, the options for subsequent categories (e.g., Group, Module) are filtered to only include choices directly associated with that specific Supergroup. For instance, after categorizing a product within "Automotive Products," the following Group Selection Agent only receives options relevant to automotive categories, streamlining classification and reducing the probability of misclassification. This method continues at each level, narrowing options progressively for the Module and Brand Selection Agents.

This filtering approach was designed to be adaptable, supporting two distinct hierarchical structures derived from the training data:

- **Supergroup-Based Hierarchy:** In this configuration, Supergroup is the primary classification, guiding the agents through a logical narrowing sequence of Group, Module, and finally Brand.
- **Brand-Based Hierarchy:** In contrast, this alternative starts with Brand as the primary classification criterion, filtering subsequent options to specific Groups and Modules associated with that brand.

5.4.2. Evaluation of Hierarchical Configurations

Extensive experimentation on both the Supergroup-Based Hierarchy and the Brand-Based Hierarchy revealed that the former yielded more accurate classifications. Utilizing Supergroup as the initial criterion provided a clearer framework for distinguishing product categories, resulting in a more intuitive hierarchy that outperformed the Brand-Based structure. The Supergroup-first approach allowed for more accurate narrowing of categories based on function and use, particularly for complex or multi-purpose items.

In contrast, the Brand-Based Hierarchy often presented challenges, as the broader range of products associated with a single brand sometimes obscured functional distinctions, leading to inconsistencies in lower-level classifications. This limitation was especially evident with generic brands that produce diverse products across unrelated categories.

5.4.3. Experimentation with Description Variants

To further evaluate classification accuracy, we experimented with both the original descriptions and the Llama3.1-generated expanded descriptions under each hierarchy. This process resulted in four model variants:

- Supergroup-Based Hierarchy with Original Descriptions
- Supergroup-Based Hierarchy with Llama3.1-Generated Descriptions
- Brand-Based Hierarchy with Original Descriptions
- Brand-Based Hierarchy with Llama3.1-Generated Descriptions

The Supergroup-Based Hierarchy using Llama3.1-expanded descriptions demonstrated the highest accuracy across all agents, benefiting from the enhanced interpretability and clarity provided by the expanded narratives. Original descriptions, while occasionally sufficient, sometimes lacked the specificity needed for precise classification, particularly in the context of niche categories. This reinforced the value of hierarchical filtering combined with detailed, generative descriptions, especially for complex or ambiguously labeled products.

5.5. Results and Challenges

In conclusion, the three approaches demonstrated both the benefits and limitations of the agentic framework for hierarchical classification. The LLM-Based Approach, implemented as the Dynamic Description-Driven Agentic Classification (DDAAC), utilized the Llama3.1-backed Description Generation Agent. This approach performed effectively with known product types but encountered challenges with unfamiliar labels and computational scalability. In terms of evaluation metrics, the DDAAC model achieved an item accuracy of 0.2831, with a CPU total time of 16.513 ms, an inference time of 0.0756 seconds, GPU FLOPS during training at 1.02183, and inference GPU FLOPS at 0.12772. However, CUDA time was not available in this setup.

The Google API-Based Approach, applied in the Smart Search-Driven Agentic Classification (SSDAC), delivered highly accurate product descriptions and significantly improved classification outcomes. Despite its potential, rate limiting posed scalability constraints, limiting its feasibility for large datasets without costly workarounds. For inference, the SSDAC model recorded an inference time of 2.530 seconds, but it did not generate predictions in this setup. Additionally, we experimented with other search engines like Bing and Brave, along with some open-source alternatives, but found Google's sophisticated ad placement and ranking algorithms, which prioritize authoritative sources, contributed most to improved description quality and classification accuracy.

Finally, the Hierarchical Option-Based Approach, tested as the Hierarchical Option-Based Agentic Classification (HOAC), showed promise on training and validation data but struggled with test data. This model achieved an item accuracy of 0.29. The approach's hierarchical filtering strategy, while helpful, suffered from cascading errors; any misclassification at the Supergroup level propagated through the hierarchy, affecting all subsequent levels. This highlighted the importance of achieving high initial classification accuracy and emphasized the need for robust error handling to mitigate cascading failures in real-world applications.

6. Semantic Search-Driven Agentic Classification (SSDAC)

In this approach, we leverage semantic search capabilities using the txtai library, specifically utilizing embeddings from the `sentence-transformers/all-MiniLM-L6-v2` model to enhance the classification of products based on their descriptions [10]. The core idea behind this model is to represent product descriptions as high-dimensional vectors, allowing for efficient semantic searching and retrieval of relevant product categories.

6.1. Rationale for Using Sentence-Transformers/all-MiniLM-L6-v2

The Sentence-Transformers/all-MiniLM-L6-v2 model is optimized for natural language processing tasks, particularly effective in encoding sentences and short paragraphs into dense vector representations. Key technical highlights include [16]:

- **High-Quality Sentence Encoding:** Maps sentences to a 384-dimensional dense vector space, capturing semantic meaning crucial for multi-class classification.
- **Semantic Similarity:** Generates embeddings that reflect semantic content, allowing accurate similarity measurement among overlapping categories.
- **Clustering and Retrieval:** Facilitates clustering of similar sentences and retrieval of relevant documents based on semantic similarity, enhancing classification outcomes.
- **Efficient Processing:** Features a lightweight architecture for fast inference, handling input text up to 256 word pieces for real-time applications.
- **Contrastive Learning Objective:** Employs a contrastive learning approach, enhancing the model's ability to distinguish between similar and dissimilar sentences, resulting in robust embeddings that improve classification accuracy.

6.2. Indexing Data

The process begins with indexing the product descriptions to create a searchable dataset. Using a dedicated indexing function, we convert the DataFrame into a suitable format for txtai, ensuring that all relevant product descriptions are indexed and ready for semantic search. The indexing process is designed to efficiently manage large datasets while maintaining the integrity and relevance of the product information.

6.3. Semantic Search Functionality

The search functionality enables querying based on user input. When a search query is executed, the function retrieves the top three most relevant matches based on the embedded representations of the product descriptions. It extracts the relevant attributes (supergroup, group, module, and brand) from the best matching result, enabling a streamlined classification process. This approach enhances the classification model by allowing it to capture semantic similarities between product descriptions and user queries, thus improving the overall accuracy and relevance of the classification results.

6.4. Results and Challenges

The Semantic Search-Driven Agentic Classification (SSDAC) model was tested with an indexing setup that concatenated all available features into a single string, separated by spaces. For querying, however, only the product description was used as input. This approach yielded an item accuracy of 0.2875, reflecting moderate performance. The model required 8.6787 seconds per item for inference, with GPU FLOPS not applicable for this configuration.

7. Experimental Infrastructure and Training Strategy

The computational demands of training complex transformer-based models for hierarchical product classification necessitated a carefully orchestrated training strategy. This section details our approach to managing computational resources while ensuring model optimization and convergence.

Our training methodology leveraged Kaggle's P100 GPU infrastructure through an iterative approach. The training process was systematically divided into 12-hour sessions to optimize the platform's weekly GPU quota of 30 hours. Between sessions, model weights were preserved and reloaded, ensuring training continuity. This process was iterated 10-15 times (with some of the best models trained on the description + retailer (without space), and then they were replaced with only description as this was a mistake), allowing for comprehensive model exploration and optimization. The approach proved particularly effective in identifying the optimal model state, striking a balance between underfitting and overfitting scenarios. An

The implementation was built on a foundation of modern deep learning frameworks, primarily utilizing the transformers library by hugging face and PyTorch. The training infrastructure required specific computational resources: NVIDIA Tesla P100 GPUs with 15GB GPU RAM and approximately 30GB CPU RAM, which aligned with Kaggle's provided environment. The docker file that can be used to replicate our results is `gcr.io/kaggle-gpu-images/python`. This setup enabled efficient batch processing and model optimization while working within platform constraints.

Our sequential training strategy not only addressed the platform's computational limitations but also provided additional benefits for model development. The intermediate checkpoints allowed for careful monitoring of model convergence and performance metrics, enabling timely interventions in the training process when necessary. This methodical approach to training management proved crucial in developing robust models for the complex task of hierarchical product classification.

8. Learnings and Insights

This datathon offered valuable insights and challenges, allowing us to test various approaches and refine our strategies for high-accuracy classification.

1. SVMs for Multi-Class Classification: We initially experimented with single and multi-class SVMs but found them resource-intensive, requiring excessive RAM and leading to overfitting. Attempts to leverage GPU memory with cuML also fell short, reinforcing the need for scalable, memory-efficient models in extreme classification tasks.
2. Impact of Retailer Information: Adding retailer data as a feature, alongside product descriptions, surprisingly decreased model accuracy. This experiment highlighted that additional features are not always beneficial; focusing solely on descriptions proved to be the most effective approach.
3. Challenges in Scaling Agentic Approaches: Our exploration of agent-based methods, especially internet-connected agents, was promising but limited by scalability challenges. We believe that fully deploying this approach could significantly improve accuracy, though it underscored the importance of scaling in multi-agent architectures.

4. Model Size and Computational Limits: Constrained to base models with 8 billion parameters, we noted that access to larger models (up to 40 billion parameters) might have boosted accuracy, underscoring the impact of computational resources on model performance in deep learning tasks.
5. Technical and Process Insights: The datathon exposed us to advanced techniques like extreme classification. Additionally, we found that fine-tuning even pre-trained models, especially on similar datasets, enhanced performance, emphasizing the benefits of custom training.

In summary, this datathon strengthened our technical skills, expanded our knowledge of scalable classification models, and highlighted the power of resilience and continuous learning in competitive settings.

9. Conclusion

In this study, we evaluated several advanced methodologies for automated attribute prediction in e-commerce, with each approach contributing to overall item accuracy based on its order in the classification hierarchy. All the results are shown in Table 1. The Multi-Learning Hierarchical Transformer (MLHT) achieved the highest item accuracy, narrowly outperforming the Expert-Ensemble Multi-Learning Hierarchical Transformer (EE-MLHT). Although the difference between MLHT and EE-MLHT was minimal, MLHT ultimately demonstrated superior performance, proving most effective for capturing complex hierarchical relationships in product attributes. Following these, the Selective Pathway Transformer Classifier (SPTC) delivered solid results, optimizing accuracy through constraint-based pathways that enhance classification efficiency at distinct attribute levels.

The agent-based approaches—Description Generation Agent (DDAAC), Smart Search-Driven Agentic Classification (SSDAC), and Hierarchical Option-Based Agentic Classification (HOAC)—provided innovative strategies for handling dynamic product descriptions, with item accuracy reflecting adaptability across different retrieval conditions. Finally, the Semantic Search-Driven Agentic Classification (SSDAC) model, while slightly lower in item accuracy compared to the earlier models, demonstrated value in situations where semantic context was essential.

Although we aimed to achieve the best possible results, computational limitations prevented some models from being trained to their optimal points, indicating a potential for further convergence with additional resources. This structured approach, with item accuracy as the core metric, allowed each method to contribute meaningfully, with earlier models yielding consistently higher accuracy. Our findings underscore the effectiveness of combining diverse learning paradigms in a tiered framework, offering a scalable solution for hierarchical classification challenges in e-commerce. Ultimately, this approach not only advances current capabilities but also sets a foundation for future innovations in automated, high-accuracy attribute prediction.

10. Acknowledgement

We would like to extend our heartfelt gratitude to the IndoML 2024 team (especially Rajdeep Mukherjee, Shubhadip Nag, and Soumi Das for their motivating support) for providing an exceptional experience during the datathon. We also appreciate Kaggle for offering free GPU resources, which greatly enhanced our experimentation and model training efforts. Additionally, we thank NielsenIQ for sponsoring this outstanding datathon, fostering innovation and collaboration in the field.

References

[1] R. Trietsch, “Product attribute value classification from unstructured text in e-commerce”, 2016.

[2] H. D. N. Harshika, K. Sugiura, N. Yamada, M. Nishi, and N. Fukuta, “An approach for extracting and predicting instance–specific attribute values from e–commerce sitesfor used products”, *International Journal of Smart Computing and Artificial Intelligence*, vol. 1, no. 2, pp. 41–57, 2017.

[3] T.-Y. Ross and G. Dollár, “Focal loss for dense object detection”, in *proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2980–2988.

[4] J. Snell, K. Swersky, and R. Zemel, “Prototypical networks for few-shot learning”, *Advances in neural information processing systems*, vol. 30, 2017.

[5] A. A. Mohammed and V. Umaashankar, “Effectiveness of hierarchical softmax in large scale classification tasks”, in *2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, IEEE, 2018, pp. 1090–1094.

[6] S. Park and N. Kwak, “Feed: Feature-level ensemble for knowledge distillation”, *arXiv preprint arXiv:1909.10754*, 2019.

[7] D. Chai, W. Wu, Q. Han, F. Wu, and J. Li, “Description based text classification with reinforcement learning”, in *International conference on machine learning*, PMLR, 2020, pp. 1371–1382.

[8] P. He, X. Liu, J. Gao, and W. Chen, “Deberta: Decoding-enhanced bert with disentangled attention”, *arXiv preprint arXiv:2006.03654*, 2020.

[9] M. Kim, J. Tack, and S. J. Hwang, “Adversarial self-supervised contrastive learning”, *Advances in neural information processing systems*, vol. 33, pp. 2983–2994, 2020.

[10] D. Mezzetti, *txtai: the all-in-one embeddings database*, Aug. 2020. [Online]. Available: https://github.com/neuml/txtai.

[11] C. Xu, W. Zhou, T. Ge, F. Wei, and M. Zhou, “Bert-of-theseus: Compressing bert by progressive module replacing”, *arXiv preprint arXiv:2002.02925*, 2020.

[12] A. G. Barto, “Reinforcement learning: An introduction. by richard’s sutton”, *SIAM Rev*, vol. 6, no. 2, p. 423, 2021.

[13] B. Li, Y. He, and W. Xu, “Cross-lingual named entity recognition using parallel corpus: A new approach using xlm-roberta alignment”, *arXiv preprint arXiv:2101.11112*, 2021.

[14] X. Liu, F. Zhang, Z. Hou, *et al.*, “Self-supervised learning: Generative or contrastive”, *IEEE transactions on knowledge and data engineering*, vol. 35, no. 1, pp. 857–876, 2021.

[15] A. Dubey, A. Jauhri, A. Pandey, *et al.*, “The llama 3 herd of models”, *arXiv preprint arXiv:2407.21783*, 2024.

[16] C. Yin and Z. Zhang, “A study of sentence similarity based on the all-minilm-l6-v2 model with “same semantics, different structure” after fine tuning”, in *2024 2nd International Conference on Image, Algorithms and Artificial Intelligence (ICIAAI 2024)*, Atlantis Press, 2024, pp. 677–684.

Table 1. Performance Metrics of Different Models.

Model	Item Accuracy	CPU Time (ms)	CUDA Time (ms)	GPU FLOPS (Train)	Inference Time (s)	GPU FLOPS (Infer)
MLHT (DeBERTa) (RL)	0.4112	919.646	277.029	405.7756	0.7682	0.7820
MLHT (DeBERTa) (HC only)	0.4087	862.201	265.606	402.0172	0.2501	2.7820
MLHT (XLM-RoBERTa) (HC only)	0.3941	1.220 (s)	391.745	396.831	0.2301	2.0872
EE-MLHT	0.3731	644.858	350.747	412.0799	0.6264	90.2679
MLHT (XLM-R Pathway)*	0.1160	644.858	350.747	412.0799	0.6264	90.2679
SPTC	0.3454	995.420	656.425	654.0156	0.6291	11.9201
SSDAC (Semantic)	0.2875	NA	NA	NA	8.6787	0.0
SSDAC (Google API)	NA	NA	NA	NA	2.5300	NA
DDAAC (LLM)	0.2831	16.513	NA	1.0218	0.0756	0.1277
HOAC (Hierarchical)	0.2900	NA	NA	NA	NA	NA