Prompt-Tuned Muti-Task Taxonomic Transformer (PTMTTaxoFormer)

Rajashekar Vasantha
Amazon Web Services
rajasvas@amazon.com

Nhan Nguyen
Amazon Web Services
ntrnguye@amazon.com

Yue Zhang
Amazon Web Services
zhangany@amazon.com

Abstract

Hierarchical Text Classification (HTC) is a subclass of multi-label classification, it is challenging because the hierarchy typically has a large number of diverse topics. Existing methods for HTC fall within two categories, local methods (a classifier for each level, node, or parent) or global methods (a single classifier for everything). Local methods are computationally expensive, whereas global methods often require complex explicit injection of the hierarchy, verbalizers, and/or prompt engineering. In this work, we propose Prompt Tuned Multi Task Taxonomic Transformer, a single classifier that uses a multi-task objective to predict one or more topics. The approach is capable of understanding the hierarchy during training without explicit injection, complex heads, verbalizers, or prompt engineering. PTMTTaxoFormer is a novel model architecture and training paradigm using differentiable prompts and labels that are learnt through backpropagation. PTMTTaxoFormer achieves state of the art results on several HTC benchmarks that span a range of topics consistently. Compared to most other HTC models, it has a simpler yet effective architecture, making it more production-friendly in terms of latency requirements (a factor of 2-5 lower latency). It is also robust and label-efficient, outperforming other models with 15%-50% less training data.

1 Introduction

Hierarchical text classification (HTC) is a specialized form of text classification that involves categorizing text documents into a hierarchical structure of labels or categories. In HTC, the labels are organized in a tree-like structure or a directed acyclic graph (DAG) (Wang et al. (2022a); Peng et al. (2018)), where parent-child relationships exist between categories at different levels. HTC allows for more nuanced and detailed classification of text compared to flat classification systems.

HTC is used in various applications where organizing and categorizing large amounts of textual information is necessary (Zangari et al., 2024). Common use cases include document organization, content management, e-commerce product categorization, knowledge management and scientific literature classification (Sadat and Caragea, 2022). These applications benefit from the hierarchy, which allows for more precise and granular categorization, particularly in domains with complex, multi-level category structures. HTC models aim to capture both the content of the text and the relationships between different levels of categories, making them effective for organizing and retrieving information in hierarchical systems.

HTC approaches are generally divided into two groups – *local* and *global* approaches. *Local* approaches tend to partially or fully ignore the hierarchical structure in the training paradigm, which lead to information loss (Punera and Ghosh (2008); Cerri et al. (2011)). Local models also have massive trainable parameters as they consist of multiple models. *Global* methods overcome the limitations above by taking into consideration the hierarchy and using the hierarchy information either explicitly or implicitly (Silla and Freitas (2011)). We refer the readers to Zangari et al. (2024) for an in-depth review of these approaches.

Prompt-based learning methods are a subset of the *global* methods. Recent advancements in prompt-based learning (Liu et al. (2023), Zhang et al. (2021), Wang et al. (2022b), Liu et al. (2023)) have shown that prompt-based approaches produce better results than vanilla finetuning of the encoded text representation. Various prompt-based learning techniques have been explored, including In-Context Learning (Brown et al. (2020)), Prompt Based Few Shot Learning (Jian et al. (2022)), Multi-prompt learning (Schick and Schütze (2020), Gao et al. (2020)) and Prompt Based Training (Li and Liang (2021)). One of the main drawbacks

to these methods is that the performance is sensitive to the prompts chosen. To mitigate this issue, we adopted differentiable prompts that are learnt through backpropagation during training, similar to Zhang et al. (2021). Another key aspect in the aforementioned approach is the treatment of labels in the hierarchy, where most earlier approaches leveraged verbalizers chosen ad-hoc (Schick and Schütze, 2020; Ji et al., 2023). However, in a complex hierarchy, especially for a highly-specialized domain, labels could have nuanced semantic differences that are difficult to capture by the verbalizer. In the meantime, it is not viable to come up with and maintain the mapping. To this end, we represent each label in the hierarchy with a new, differentiable token and these tokens are also learned through backpropagation during training.

Our method uses text input appended with prompt tokens and $[MASK]$ tokens (one for each level of hierarchy). The model is then trained to unmask the $[MASK]$ tokens to output the correct label tokens (again, one / more for each level of hierarchy). The prompt tokens, the label tokens and the model parameters are jointly optimized during the training process. We describe the process in further detail in subsection 2.1. The objective function we use to optimize the above parameters is a combination of Weighted Asymmetric Loss, L1 Loss and MLM loss, as described in subsection 2.2.

We summarize our contributions as follows:

- We proposed Prompt-Tuned Multi-Task Taxonomic Transformer – a prompt tuning method for multi-label hierarchical text classification that does not require complex prompt engineering, verbalizers or explicit hierarchy injection during the training process.
- We implemented Weighted Asymmetric Focal Loss (WASL), a new loss function that is capable of handling the class imbalance in the hierarchical setting.
- Our method achieved state-of-the-art results on several benchmark datasets, and is applicable to any text domains. The approch is simpler compared to other methods, leading to a factor of 2 to 5 lower latency requirements in production.
- Our method is label-efficient and robust in low resource settings, outperforming other models with around 15% to 50% less training data. This is critical for many industrial applications, as getting training data is expensive.

2 Methodology

The problem of hierarchical multi-label prediction consists of two components: Hierarchical Prediction and Multi-Label Prediction. This section describes our approach in solving both components.

2.1 Hierarchical Prediction through Hierarchical Differentiable Prompt Tuning

We attempt the hierarchical prediction by prompting a language model to predict the labels at different levels of the hierarchy. Choosing a prompt through prompt engineering is a tedious process with infinitely many possibilities. Hence, we use differentiable prompts that are optimized through backpropagation as proposed by Zhang et al. (2021) and extend it to hierarchical prediction. The following section explains this method in more detail.

Figure 1 shows the architecture of our approach during training. Our experiments were performed with BERT (Devlin et al. (2018a)) as the backbone. The approach can, however, be extended to almost all large language models including RoBERTa (Liu et al. (2019)), ALBERT (Lan et al. (2019)) and DeBERTa (He et al. (2020)) that are encoder-based, T-5 (Raffel et al. (2020)) and BART (Lewis et al. (2019)) that are encoder-decoder-based and GPT-like (Radford et al. (2019)) models that are decoder-based with minor modifications to the architecture.

We follow the prompting scheme proposed by Zhang et al. (2021) and set some of the unused tokens in the vocabulary as prompt tokens. To extend our approach to use backbones like RoBERTa (Liu et al. (2019)) and ALBERT (Lan et al. (2019)) that do not have unused tokens, we simply have to add new tokens to the vocabulary and use them as prompt tokens. The embeddings corresponding to these tokens are learnt during the training process. We tokenize the input text using the regular tokenizer corresponding to the chosen backbone and append to it the following tokens:

- Prompt Tokens
- $[MASK]$ Tokens

The number of prompt tokens(m) is a hyperparameter to tune. The number of $[MASK]$ tokens added is equal to the number of levels in the hierarchy.

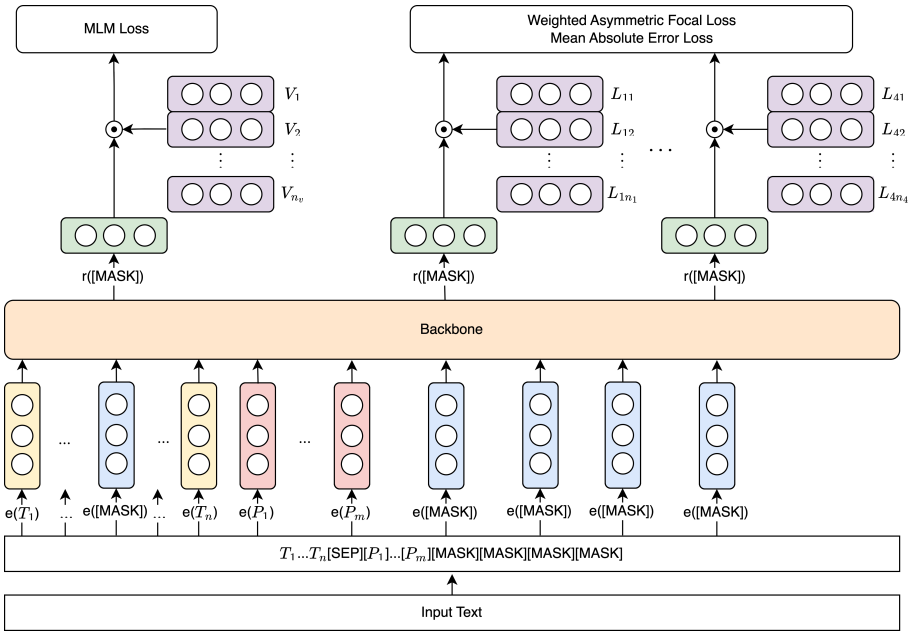

Figure 1: **The architecture of Prompt-Tuned Muti-Task Taxonomic Transformer during training**. PTMTTaxoFormer transforms hierarchical multi label classification into [MASK] token prediction problem. The input text is tokenized, appended with differentiable prompt tokens and [MASK] tokens. The number of [MASK] tokens is equal to the number of levels in the hierarchy. The model learns to unmask these tokens and predict the correct label-tokens (also differentiable). We use a linear combination of Weighted Asymmetric Focal Loss (Equation 1), Mean Absolute Error and Masked Language Modeling Loss as the final loss function. The diagram above shows an example scenario with a taxonomy of four levels ($d = 4$)

After appending these tokens, the processed input token sequence looks like this:

$$[CLS], [T_1], \ldots, [T_n], [SEP], [P_1], \ldots, [P_m], \underbrace{[MASK], \ldots [MASK]}_{\text{No. of levels in hierarchy}}, [SEP]$$

where $[CLS]$ is a special token used in BERT (Devlin et al. (2018a)), $[T_1], \ldots, [T_n]$ represent the tokens of the input sentence and $[P_1], \ldots, [P_m]$ represent the learnable prompt tokens.

The model learns to fill the $[MASK]$ tokens with labels corresponding to different levels of the hierarchy - the first $[MASK]$ token corresponding to the label in the first level of the hierarchy, the second $[MASK]$ token corresponding to the label of the second level and so on. We flatten labels at each level of the hierarchy and train the model to make predictions in this space. The flattened labels are mapped to tokens in the vocabulary (can be unused tokens or newly added tokens as described above for the prompt tokens) and the training procedure optimizes the model to predict the correct tokens from this set of tokens. This procedure is repeated for each level in the hierarchy.

For example, consider a hierarchy with depth d and each level having $n_1, \ldots n_d$ number of flattened nodes, respectively. The label-tokens for each level can be represented as follows:

$$\begin{aligned}
&\text{Level 1: } [L_{11}], \ldots [L_{1n_1}] \\
&\text{Level 2: } [L_{21}], \ldots [L_{2n_2}] \\
&\vdots \\
&\text{Level } d\text{: } [L_{d1}], \ldots [L_{dn_d}]
\end{aligned}$$

$[L_{ij}]$ here refers to a token in the vocabulary. If we consider a hierarchy with *four* levels and an input sentence with $l_{12}, l_{25}, l_{31}, l_{44}$ as the true labels corresponding to the four levels in its hierarchy, the model learns to predict the above *four* tokens in place of the *four* $[MASK]$ tokens during inference. Note that there is no explicit hierarchy injection during training. During inference, we prune the paths predicted by the above method to include only valid paths present in the hierarchy. The architecture at inference is shown in Figure 3.

During training, the input tokens are masked with a small probability (Table 4). This ensures association between the prompt, the label and the input tokens while retaining the language understanding ability of the pretrained model. All the parameters are jointly optimized using the objectives described in the following section.

2.2 Training Objectives

As our task is to predict multiple labels, we reduce it to a series of binary classification tasks as commonly done in multi-label classification. We train the model using a combination of three objective functions - Asymmetric Focal Loss, L1 Loss, and Masked Language Modeling Loss. Each of them is described below.

- **Weighted Asymmetric Focal Loss:** We follow Ben-Baruch et al. (2020) and extend their *Asymmetric Loss* to what we call *Weighted Asymmetric focal Loss (WASL)* and use it independently for each level in the hierarchy. A typical hierarchy in the datasets we experiment with consists of several hundred nodes. Out of these, for a datapoint, only a few will be correct (positive) and the rest incorrect (negative) resulting in a severe positive-negative imbalance. This leads to under emphasis (Lin et al. (2017)) of gradients from positive labels during the training process. We use the focal loss to focus more on positive labels to mitigate this problem.

  As the hierarchy branches out, there are fewer training examples at lower levels compared to higher levels. To overcome this imbalance, we additionally add an increased weight on the positive class. This weight increases as we go down the hierarchy levels.

  The logits corresponding to the label-tokens are first independently computed using the sigmoid function. Then, the *WASL* is applied to the logits and labels corresponding to each level in the hierarchy. The different losses at each level are then summed up.

  We define *WASL* for each logit as follows. Here, i refers to the level and j refers to one flattened label in that level:

$$\mathcal{L}_{WASL_{i,j}} = \begin{cases} \mathcal{L}_+ = w(1-p)^{\gamma_+}\log(p) \\ \mathcal{L}_- = (p_m)^{\gamma_-}\log(1-p_m) \end{cases} \quad (1)$$

  where, p is the probability after sigmoid, γ_+ and γ_- are the positive and negative focusing parameters respectively and $p_m = \max(p-m, 0)$. p_m is the shifted probability and $m > 0$ is the probability margin. More information about the values of these hyperparameters is given in subsection A.4. w is the weight assigned to the positive class. *WASL* for a level is the sum of the *WASL* for each logit. *WASL* for each level is then summed across all levels to get the total loss.

  The total *WASL* is given by the below equation. d is the number of levels in the hierarchy and n_i is the number of labels in the i^{th} level.

$$\mathcal{L}_{WASL} = \sum_{i=1}^{d}\sum_{j=1}^{n_i}\mathcal{L}_{WASL_{i,j}} \quad (2)$$

- **L1-Loss:** The problem of multi-label hierarchical prediction presents a challenge where we do not know the number of correct labels for a given example in advance. We want to explicitly penalize too few predictions as well as too many predictions. To do this, we regularize the number of predictions using L1-loss between the sum of logits after sigmoid and the true number of labels.

- **Masked Language Modeling Loss:** During training, we mask each non-label-token with a probability of p_{mlm} and we mask each label-token with a probabilty of p_{label}. Allowing some of the label-tokens to be unmasked during training is more effective compared to masking all of them as shown in Zhang et al. (2021). We use the regular cross-entropy loss on the logits corresponding to the masked tokens. The masking strategy used is the same as described in Devlin et al. (2018b).

The total loss optimized is computed as below:

$$\mathcal{L} = \alpha\mathcal{L}_{mlm} + (1-\alpha)(\mathcal{L}_{WASL} + \alpha_{l1}\mathcal{L}_{l1}) \quad (3)$$

An illustration of the methodology using a toy example has been provided in subsection A.7.

The above method can also be reduced to single label prediction either by using the WASL for single label prediction or the regular cross-entropy loss over the logits of the label-tokens. We will have to compute one loss for each level of the hierarchy in this case too.

2.3 Hierarchy Injection

The hierarchy is implicitly injected during training and explicitly used during inference. During training, the $[MASK]$ tokens corresponding to the label positions attend to each other in addition to attending to the prompt and input tokens. We hypothesize that the bidirectional attention mechanism

implicitly injects hierarchical information during training. During inference, we explicitly use the hierarchical label structure to eliminate the predicted paths that are not a part of the hierarchy, if any.

3 Experiment Setup

3.1 Datasets

Our methods are evaluated on four HTC datasets that cover a diverse array of topics and difficulties (Zangari et al. (2024)): Web of Science (Kowsari (2018)), Blurb Genre Collection (Aly et al. (2019)), Linux Bugs Dataset (Lyubinets et al. (2018)), and Amazon 5 $\times$ 5 (Zangari et al. (2024); Ni et al. (2019)). The diversity of the selected datasets serves as a rigorous test for our proposed approach to assess its robustness and scalability in handling intricate hierarchical structures. More details about the datasets are presented in subsection A.2.

3.2 Implementation Details

We use the off-the-shelf BERT (bert-base-uncased) model (Devlin et al. (2018a)) as the backbone in our architecture (Figure 1) to ensure a fair comparison with other reported metrics. The AdamW (Loshchilov and Hutter (2017)) optimizer is used in training, and we perform hyperparameter optimization as described in subsection A.4. We train the model with train set and evaluate on development set after every epoch and stop training if the macro-h-$F1$ does not increase for 20 epochs.

3.3 Evaluation metrics

A number of evaluation metrics for HTC have been proposed in the literature (Zhang and Zhou (2014); Zangari et al. (2024)). We provide a brief summary below and explain the reason for our choice.

Many researchers evaluate their HTC models by flattening the hierarchical labels to apply the standard classification scores (e.g., accuracy, precision, recall and $F1$ scores. The standard classification scores, however, disregard the label hierarchy and treat each of the flattened labels independently. Yet in practical applications, correctly classifying higher level nodes is typically more consequential than the lower level ones. In enterprise customer support, for example, the higher level nodes decide the ticket delegation (routing), while lower level nodes aim to provide more context for triggering the right automation. An incorrect prediction on higher level nodes may result in routing to the wrong expert and should receive more penalty. Kiritchenko et al. (2006) introduced hierarchical metrics to calculate the metrics on predicted and true labels both augmented with all ancestors to mitigate the issues discussed above.

For reasons discussed above, we focus only on the hierarchical metrics in the current study. To calculate an overall metrics for all categories, we implemented the macro-average and reported the macro-h-$F1$ score. See subsection A.5 for details. Suffice it to note that a model trained to achieve high scores on hierarchical metrics does not guarantee the best performance on other metrics. Thus, it is critical to determining an appropriate evaluation metrics based on business needs and adjust the training script accordingly.

4 Results

4.1 Performance Benchmark

Table 1 compares our results on the datasets we experiment with, to the state-of-the-art results. We obtained the state-of-the-art results from Zangari et al. (2024) and compare our approach to the top 3 HTC models listed – HBGL (Jiang et al. (2022)), GACaps (Bang et al. (2023)) and BERT + ML (Zangari et al. (2024)). Interestingly, it was observed that the best reported model HBGL outperformed GACaps and BERT + ML on three of the four datasets (Bugs, WOS and BGC), but the performance degraded notably on the Amazon dataset (4.2% lower than BERT + ML). In contrast, our approach exhibited more consistent performance: it outperformed all models on Bugs, BGC and Amazon datasets and achieved comparable results to HBGL on WOS dataset (only 0.2% lower).

4.2 Performance on Low Resource Setting

Further, we conducted experiments to see how our approach performance against low resource training settings. For both BGC and Amazon 5 $\times$ 5, we fixed the validation and test set for all runs, and sampled a subset of training data randomly. As can be seen in Fig. 2, the performance increases rapidly in the low-data regime (below 20% of the full training data volume) and saturates after around 50% of the training volume. This observation shows that our approach is still effective with fewer training labels. Such label efficiency is critical in practical applications as training data is rather expensive to collect. To further illustrate this, we also marked the performance of other three models trained with the full training set. It was found that our approach

Table 1: **Results on common HTC benchmarks.** We compare our approach to the top 3 HTC models Zangari et al. (2024) name – HBGL (Jiang et al. (2022)), GACaps (Bang et al. (2023)) and BERT + ML (Zangari et al. (2024)). We report macro-h-$F1$ (higher the better) in the table below. We achieve state-of-the-art results on Bugs, BGC and Amazon 5 $\times$ 5 datasets while achieving near state-of-the-art results on WOS dataset.

Model	Bugs	WOS	BGC	Amazon
PTMTTaxoFormer (Ours)	**0.5727**	0.8201	**0.6903**	**0.9327**
HBGL (Jiang et al. (2022))	0.5710	**0.8221**	0.6779	0.8796
GACaps (Bang et al. (2023))	0.5445	0.8067	0.6446	0.9057
BERT + ML (Zangari et al. (2024))	0.5172	0.7974	0.6119	0.9214

could outperform other models even with a subset of the training data, further showing that our approach is robust and label efficient.

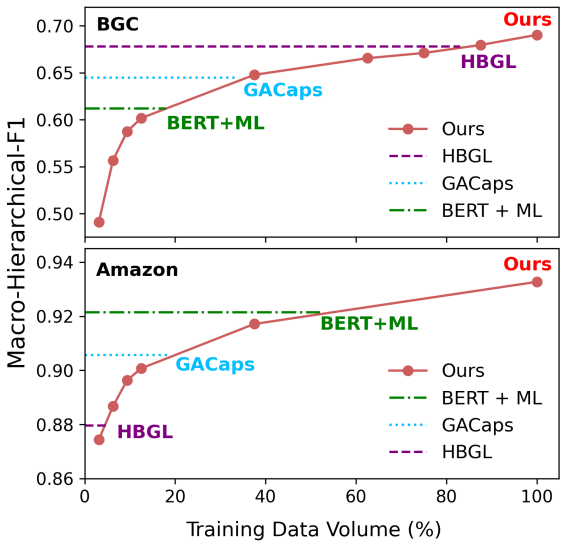

Figure 2: Performance of PTMTTaxoFormer (●—●) as training data volume decreases , compared with the performance of other models trained with full training data. Top panel: Blurb Genre Collection (BGC); Bottom panel: Amazon 5 $\times$ 5 dataset. HBGL (- - - - -); GACaps (· · · · · ·); BERT + ML (–·–·–)

4.3 Complexity and Inference Time

In many industrial applications, real-time predictions are needed, and thus the model architecture could not be overly complicated. In a recent review, Zangari et al. (2024) compared the complexity and performance trade-off of several HTC model architectures. The reported best-performing HBGL model has high complexity and latency (e.g., HBGL and GACaps has latency 73.8 ms and 26.7 ms on Bugs dataset using NVIDIA RTX 2080Ti GPU, while BERT only requires 11.3 ms). The latency gaps between BERT and HBGL could be explained by the model architecture. For HBGL, BERT has to be used to generate contextualized embedding, while the BERT model is used as a multi-class, multi-label classifier. In addition, HBGL make predictions in an auto-regressive manner, indicating that the number of BERT encoding scales with the number of levels in the hierarchy. Our PTMTTaxoFormer model achieves performance better than or comparable to HBGL with BERT-like inference time and complexity irrespective of number of levels in hierarchy, thus rendering it more friendly for real-time industry applications.

5 Discussion and Future Work

This paper presents PTMTTaxoFormer - a simple framework that can be extended to any hierarchical text prediction task using any pretraining language model available. Our method does not require any complex heads or modules for hierarchy injections, verbalizers or engineering of prompts. While retaining simplicity, our framework also attains state-of-the-art performance. We attribute the improvement in performance to the fact that the classification task is made to resemble the pretraining task (MLM in case of BERT) and it utilizes the ability of the model to perform pretraining tasks well. During training, attention between the label tokens allows the model to learn the hierarchy implicitly – the bi-directional self attention proves to be a powerful architecture to implicitly learn hierarchies. As a part of the future work, we'd like to extend our approach to use a decoder or encoder-decoder-based approach as we expect next token prediction problem to also be a good proxy for explicit hierarchy injection. The prompts used in our method are the same for all samples. A path worth exploring would be to have a small language model to predict the prompt tokens making the prompts customized to each sample.

References

Rami Aly, Steffen Remus, and Chris Biemann. 2019. Hierarchical multi-label classification of text with capsule networks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop*, pages 323–330.

Jinhyun Bang, Jonghun Park, and Jonghyuk Park. 2023. Gacaps-htc: graph attention capsule network for hierarchical text classification. *Applied Intelligence*, 53(17):20577–20594.

Emanuel Ben-Baruch, Tal Ridnik, Nadav Zamir, Asaf Noy, Itamar Friedman, Matan Protter, and Lihi Zelnik-Manor. 2020. Asymmetric loss for multi-label classification. *arXiv preprint arXiv:2009.14119*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Ricardo Cerri, Rodrigo C Barros, and André CPLF de Carvalho. 2011. Hierarchical multi-label classification for protein function prediction: A local approach based on neural networks. In *2011 11th International Conference on Intelligent Systems Design and Applications*, pages 337–343. IEEE.

Huiyao Chen, Yu Zhao, Zulong Chen, Mengjia Wang, Liangyue Li, Meishan Zhang, and Min Zhang. 2024. Retrieval-style in-context learning for few-shot hierarchical text classification. *Transactions of the Association for Computational Linguistics*, 12:1214–1231.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018a. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina N. Toutanova. 2018b. Bert: Pre-training of deep bidirectional transformers for language understanding.

Tianyu Gao, Adam Fisch, and Danqi Chen. 2020. Making pre-trained language models better few-shot learners. *arXiv preprint arXiv:2012.15723*.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2020. Deberta: Decoding-enhanced bert with disentangled attention. *arXiv preprint arXiv:2006.03654*.

Ke Ji, Yixin Lian, Jingsheng Gao, and Baoyuan Wang. 2023. Hierarchical verbalizer for few-shot hierarchical text classification. *arXiv preprint arXiv:2305.16885*.

Yiren Jian, Chongyang Gao, and Soroush Vosoughi. 2022. Contrastive learning for prompt-based few-shot language learners. *arXiv preprint arXiv:2205.01308*.

Ting Jiang, Deqing Wang, Leilei Sun, Zhongzhi Chen, Fuzhen Zhuang, and Qinghong Yang. 2022. Exploiting global and local hierarchies for hierarchical text classification. *arXiv preprint arXiv:2205.02613*.

Svetlana Kiritchenko, Stan Matwin, Richard Nock, and A Fazel Famili. 2006. Learning and evaluation in the presence of class hierarchies: Application to text categorization. In *Advances in Artificial Intelligence: 19th Conference of the Canadian Society for Computational Studies of Intelligence, Canadian AI 2006, Québec City, Québec, Canada, June 7-9, 2006. Proceedings 19*, pages 395–406. Springer.

Kamran Kowsari. 2018. Web of science dataset.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*.

Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2017. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Volodymyr Lyubinets, Taras Boiko, and Deon Nicholas. 2018. Automated labeling of bugs and tickets using attention-based mechanisms in recurrent neural networks. In *2018 IEEE Second International Conference on Data Stream Mining & Processing (DSMP)*, pages 271–275. IEEE.

Jianmo Ni, Jiacheng Li, and Julian McAuley. 2019. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. 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 188–197.

Hao Peng, Jianxin Li, Yu He, Yaopeng Liu, Mengjiao Bao, Lihong Wang, Yangqiu Song, and Qiang Yang. 2018. Large-scale hierarchical text classification with recursively regularized deep graph-cnn. In *Proceedings of the 2018 world wide web conference*, pages 1063–1072.

Kunal Punera and Joydeep Ghosh. 2008. Enhanced hierarchical classification via isotonic smoothing. In *Proceedings of the 17th international conference on World Wide Web*, pages 151–160.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551.

Mobashir Sadat and Cornelia Caragea. 2022. Hierarchical multi-label classification of scientific documents. *arXiv preprint arXiv:2211.02810*.

David Salinas, Matthias Seeger, Aaron Klein, Valerio Perrone, Martin Wistuba, and Cedric Archambeau. 2022. Syne tune: A library for large scale hyperparameter tuning and reproducible research. In *International Conference on Automated Machine Learning*, pages 16–1. PMLR.

Timo Schick and Hinrich Schütze. 2020. Exploiting cloze questions for few shot text classification and natural language inference. *arXiv preprint arXiv:2001.07676*.

Carlos N Silla and Alex A Freitas. 2011. A survey of hierarchical classification across different application domains. *Data mining and knowledge discovery*, 22:31–72.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25.

Zihan Wang, Peiyi Wang, Lianzhe Huang, Xin Sun, and Houfeng Wang. 2022a. Incorporating hierarchy into text encoder: a contrastive learning approach for hierarchical text classification. *arXiv preprint arXiv:2203.03825*.

Zihan Wang, Peiyi Wang, Tianyu Liu, Yunbo Cao, Zhifang Sui, and Houfeng Wang. 2022b. Hpt: Hierarchy-aware prompt tuning for hierarchical text classification. *arXiv preprint arXiv:2204.13413*.

Alessandro Zangari, Matteo Marcuzzo, Matteo Rizzo, Lorenzo Giudice, Andrea Albarelli, and Andrea Gasparetto. 2024. Hierarchical text classification and its foundations: A review of current research. *Electronics*, 13(7):1199.

Min-Ling Zhang and Zhi-Hua Zhou. 2014. A review on multi-label learning algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 26(8):1819–1837.

Ningyu Zhang, Luoqiu Li, Xiang Chen, Shumin Deng, Zhen Bi, Chuanqi Tan, Fei Huang, and Huajun Chen. 2021. Differentiable prompt makes pre-trained language models better few-shot learners. *arXiv preprint arXiv:2108.13161*.

A Appendix

A.1 Architecture during inference

Figure 3 illustrates the architecture for inference.

A.2 Dataset details

In the current work, we considered four public datasets commonly used in HTC. These datasets are decribed and summarized in detail in a recent review paper (Zangari et al. (2024)). In what follows, we provide a brief description for each dataset and provide the statistics of the each dataset in Table 2.

- **Web of Science (WOS):** The Web of Science (WOS) dataset, was introduced by Kowsari (2018) and consists of abstracts from scholarly papers published on the Web of Science platform (`https://www.webofscience.com`). We use the WOS-46985 version of the dataset. This consists of text from 46985 abstracts. They are classified into 7 domains that are further divided into 134 subdomains.
- **Blurb Genre Collection (BGC):** The Blurb Genre Collection (BGC) (Aly et al. (2019)) is a dataset consisting of advertising descriptions of books - so called blurbs - for the English language. Each blurb is categorized into one or multiple categories. This dataset was obtained from Penguin Random House webpage that contains both the blurb and the genre or category for each book. This dataset contains 91,892 samples with the genre hierarchy consisting of 146 classes.
- **Linux Bugs Dataset (Bugs):** The Linux Bugs dataset (Bugs) was introduced by Aly et al. (2019) and comprises bugs scraped from the Linux kernel bugtracker (`https://bugzilla.kernel.org`). The text is derived from the support tickets and are classified into "Product" at the parent level and "Component" at the child level. Zangari et al. (2024) extend this dataset to increase its size. We utilize the extended dataset in our experiments. This dataset contains very noisy text, grammatical errors, technical jargon and strongly unbalanced labels which makes it a good candidate to test our methodology.
- **Amazon 5 $\times$ 5 (Amazon):** This dataset was introduced by Ni et al. (2019) and subsequently utilized by Zangari et al. (2024). From the original dataset, they curated product reviews spanning five categories: "Arts, Crafts and Sewing", "Electronics", "Grocery and Gourmet Food", "Musical Instruments", and "Video Games". For each of these overarching categories, they further extracted five subcategories. Notably, this dataset exhibits a balanced label distribution, enabling us to test the hypothesis that our proposed model will also achieve superior performance on balanced data.

A.3 Ablation Studies

We perform ablation studies by removing several modules / loss functions from our training paradigm. Table 3 shows the results of these studies. We choose *BGC* dataset for ablation studies. *r.m* means that this module or loss function was removed and *r.p* means that the modules were replced with an alternative. Whenever we remove or replace a module, we train the model with the same hyperparameter optimization scheme used when the module was not removed or replaced. We see that removing MLM loss leads to the maximum drop in macro-h-$F1$. MLM objective ensures association among all the tokens (along with newly added label and prompt tokens), helps maintain the ability of language understanding and also acts as a regularizer preventing overfitting. Removing MAE loss resulted in a minimal drop of macro-h-$F1$ but adding this prevented an initial training collapse where we observed all the labels were predicted as true. We also replace *WASL* with the standard binary cross-entropy loss. Although the drop in macro-h-$F1$ for this dataset was minimal, we observed a significant improvement of the metric when this was repeated on internal datasets.

A.4 Hyperparameters

We use Syne Tune (Salinas et al. (2022)) to tune hyperparameters for our methodology and architecture. Table 4 lists all the hyperparameters considered and their respective values / ranges used during training. In the table, *Range of Values* gives the lower and upper bounds if the parameter is included in hyperparameter optimization. If it is a single number, then this hyperparameter was not tuned. For all runs, we used Bayesian Optimization (Snoek et al. (2012)) as the hyperparameter optimization algorithm.

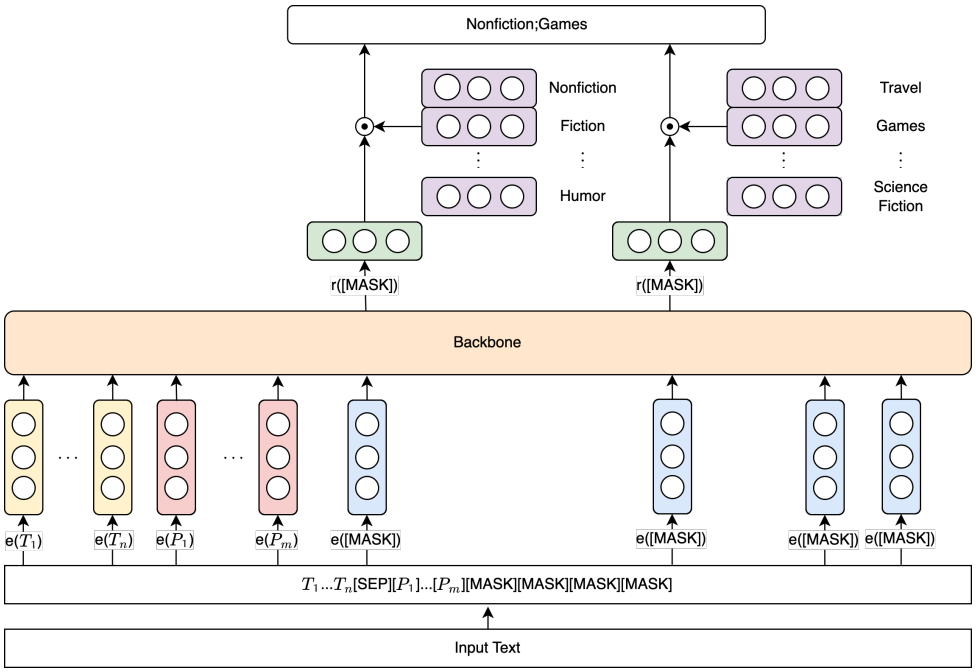

Figure 3: **The architecture of Prompt Tuned Muti Task Taxonomic Transformer during inference.** During inference, the input text is tokenized, appended with differentiable prompt tokens and $[MASK]$ tokens. The number of $[MASK]$ tokens is equal to the number of levels in the hierarchy. The model unmasks these tokens to predict the correct label-tokens. We take the inner product of representation corresponding to the $[MASK]$ token with all the label-tokens at the corresponding level, followed by sigmoid. The resulting scores are thresholded to determine a set of possible labels at each level. These labels are then pruned using the true taxonomy to eliminate incorrect paths.

Table 2: **Dataset Statistics**

	Bugs	**WOS**	**BGC**	**Amazon**
Size	35,050	46,960	91,894	500,000
Depth	2	2	4	2
Labels overall	102	145	146	30
Labels per level	17-85	7-138	7-46-77-16	5-25
Average # characters	2026	1376	996	2194
Train	18,692	31,306	58,715	266,666
Validation	4674	6262	14,785	66,667
Test	11,684	15,654	18,394	166,667

A.5 Equations for evaluation metrics

The equations below define the evaluation metrics used in the current work. $\hat{Y}_{aug}$ and Y_{aug} are the predicted labels and ground truth labels, respectively. Both sets are augmented by the ancestors to account for the hierarchy.

$$h\text{-}Pr = \frac{\sum_i |\hat{Y}_{aug} \cap Y_{aug}|}{\sum_i |\hat{Y}_{aug}|} \tag{4}$$

$$h\text{-}Re = \frac{\sum_i |\hat{Y}_{aug} \cap Y_{aug}|}{\sum_i |Y_{aug}|} \tag{5}$$

To derive an overall statistics, we considered macro-average of precision and recall for each label and report the macro-h-$F1$ score for model evaluation.

$$h\text{-}Pr_{macro} = \frac{\sum_{i=1}^{m} h\text{-}Pr_i}{m} \tag{6}$$

$$h\text{-}Re_{macro} = \frac{\sum_{i=1}^{m} h\text{-}Re_i}{m} \tag{7}$$

$$h\text{-}F1 = 2 \cdot \frac{h\text{-}Pr \cdot h\text{-}Re}{h\text{-}Pr + h\text{-}Re} \tag{8}$$

A.6 Performance under low-resource settings

Table 5 shows the performance of PTMTTaxo-Former architecture under low training data regime. We studied two selected datasets (BGC and Amazon). For each run, the validation and test sets are

Table 3: **Ablation Studies for BGC dataset**

Ablation Description	macro-h-$F1$
PTMT TaxoFormer	**0.6903**
r.m. MLM Loss	0.6712
r.m. MAE Loss	0.6878
r.p. WASL with Weighted Binary Cross-Entropy Loss	0.6870

Table 4: **Hyperparameter Settings**

Hyperparameter	**Range of Values**	**Sampling Method**
α	$[0.7, 1]$	Uniform
γ_+	$[1, 10]$	Random Integer
γ_-	$[1, 10]$	Random Integer
m	$[0.02, 0.1]$	Uniform
Learning Rate	$[1e-6, 1e-2]$	Log Uniform
Number of Prompt Tokens	$[2, 10]$	Random Integer
Batch Size	$[2, 32]$	Random Integer
Input Token Mask Probability	0.15	N/A
Early Stopping Epochs	10	N/A
α_{l1}	1	N/A
Warm Up Ratio	0.05	N/A
Weight Decay	0.01	N/A
Maximum Gradient Norm	5	N/A

fixed and training data were sub-sampled with the volume ratio specified in the first columns (from around 3% to 100%). Sub-samples are chosen randomly. For amazon dataset, we focused more on smaller training volume ratios due to the larger size of the full Amazon training set.

Table 5: **Performance of training PTMT TaxoFormer on BGC and Amazon on random subset of training data.** Note that for all runs the validation set and test sets are fixed, while training data is sub-sampled randomly.

Training volume ratio	**BGC**	**Amazon**
1/32 (3.125%)	0.4910	0.8744
2/32 (6.25%)	0.5566	0.8867
3/32 (9.375%)	0.5874	0.8963
4/32 (12.5%)	0.6017	0.9007
12/32 (37.5%)	0.6477	0.9171
20/32 (62.5%)	0.6654	-
24/32 (75%)	0.6710	-
28/32 (87.5%)	0.6795	-
32/32 (100%)	0.6903	0.9327

A.7 Illustration of the model architecture, training and inference process with a toy example

Consider a hierarchy with a maximum of 3 levels as shown in Figure 4. Consider a toy BERT model with vocabulary given in Table 6.

Assume the dataset we are training on has two items:

1. `MNO with labels A -> AA -> AAA`
2. `XYZ with labels C`

The first item has a label that goes to the third level of the hierarchy and the second item has a label that ends at the first level of the hierarchy.

During training, the input text is appended with prompt tokens and mask tokens. The transformed output looks like below:

1. `MNO[SEP][P_1][P_2][P_3][SEP][MASK][MASK][MASK]`
2. `XYZ[SEP][P_1][P_2][P_3][SEP][MASK][MASK][MASK]`

These are then converted into embedding

vectors which look like the below:

1. $e_{13}, e_{14}, e_{15}, e_{51}, e_{53}, e_{54}, e_{55}, e_{52}, e_{52}, e_{52}$
2. $e_{24}, e_{25}, e_{26}, e_{51}, e_{53}, e_{54}, e_{55}, e_{52}, e_{52}, e_{52}$

After the forward pass through the encoder, we end up with logit vectors for each of these tokens which are then passed through a sigmoid layer. Let us call the output vectors after sigmoid as activation vectors. We are particularly interested in the activation vectors corresponding to the [MASK] tokens. These activation vectors have the same dimension as the vocubulary of the model.

For the first example, consider a_{11}, a_{21}, a_{31} to be the activation vectors corresponding to the [MASK] tokens. We want the model to learn to predict [LABEL_A],[LABEL_AA],[LABEL_AAA] in place of the [MASK],[MASK],[MASK] tokens. Post training, we want the model to have the following elements of the activation vectors: $a_{11,103}, a_{21,106}, a_{31,109}$ to be close to 1 and all the other elements of the activation vectors to be close to zero.

Similarly, for the second example, consider a_{12}, a_{22}, a_{32} to be the activation vectors corresponding to the [MASK] tokens. We want the model to learn to predict [LABEL_C],[], [] in place of the [MASK],[MASK],[MASK] tokens. [] means that the model does not predict anything. Post training, we want the model to have $a_{12,105}$ to be close to 1 and all the other elements of the activation vectors to be close to zero.

We minimize the loss in Equation 3 to achieve this.

During inference, we take all the elements of the activation vectors that are above a defined threshold (0.5 for simplicity) corresponding to the [MASK] tokens. We then eliminate those whose parent element (obtained from the hierarchy) is not present in this list. The elimination step only occurs for the second level and below. This leaves us with elements / predictions that adhere to the hierarchy structure.

A.8 Selection of datasets

We selected datasets based on a recent HTC review (Zangari et al. (2024)), where five datasets were studied (BGC, Bugs, WOS, Amazon and RCV1-v2). For a fair comparison, we benchmarked our approach using the same data split provided by Zangari et al. (2024). RCV1-v2 dataset was not provided in the original paper appendix. RCV1-v2 is also not publicly available and needs request to obtain. RCV1-v2 is similar to BGC. BGC adapted RCV1-v2's properties, and was constructed to mimic its setting. The dataset statistics are comparable, e.g., overall labels are 103 vs 146, label per level is 4-55-43-1 vs 7-46-77-16) [1]. Given the similarity, we decided to choose BGC to benchmark our results as an alternative.

A.9 Comparison with Zeroshot / Fewshot inference using more recent generative LLMs

We have been working on comparing our approach with zeroshot and fewshot inference using more recent generative LLMs and intend to publish our findings in a subsequent work. At a higher level, we argue that LLMs and supervised small models both have pros and cons.

- LLMs work better with very few examples (< 50). However, with sufficient data, our approach outperforms the zero-shot LLMs by a large margin. Prompting techniques improves LLM to approach results from small models, yet still under performs. For example, our experiments on BGC show Claude Haiku and Claude V2 have scores of 0.346 and 0.363 using zero-shot, chain-of-thought prompting, while our PTMT small LM shows 0.6903. Chen et al. Chen et al. (2024) showed that zero-shot ChatGPT on WOS has a score of 0.4479. With ICL and retrieval techniques, the best reported LLM score is 0.7408. In comparison, our approach gets 0.8221.

- LLMs have notably higher latency (5-10 seconds vs milliseconds for our approach) and cost.

- LLM outputs are inconsistent even with the same input. We found around at least 10-15% of the predictions inconsistent on multiple runs even with temperature = 0.

In brief, our approach and LLMs have pros and cons and could be used in different scenarios.

A.10 Scalability of the Approach

The public datasets we tested on, have hierarchies that contain 134 to 145 nodes. Our internal datasets had around 200-300 nodes per hierarchy and we observed similar performance improvement over the existing methods here as well. We have tested our method on 150 hierarchies that we have internally.

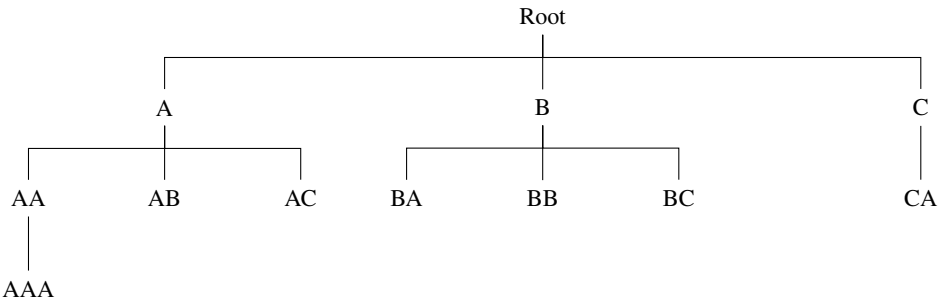

Figure 4: **Toy example of a taxonomy** - This taxonomy contains a maximum of 3 levels.

Table 6: **Sample vocabulary** This table shows tokens and the corresponding embedding lookup key for a sample vocabulary.

Token	Key	Token	Key	Token	Key	Token	Key
A	1	N	14	[CLS]	50	[LABEL_A]	103
B	2	O	15	[SEP]	51	[LABEL_B]	104
C	3	P	16	[MASK]	52	[LABEL_C]	105
D	4	Q	17	[P_1]	53	[LABEL_AA]	106
E	5	R	18	[P_2]	54	[LABEL_AB]	107
F	6	S	19	[P_3]	55	[LABEL_AC]	108
G	7	T	20			[LABEL_AAA]	109
H	8	U	21			[LABEL_BA]	110
I	9	V	22			[LABEL_BB]	111
J	10	W	23			[LABEL_BC]	112
K	11	X	24			[LABEL_CA]	113
L	12	Y	25				
M	13	Z	26				

One of the future works we have lined up is to measure the performance of this method when all these hierarchies are combined into a single hierarchy and a single model is trained. This would lead to a massive hierarchy with 30,000 nodes which would be a good test of scalability.

A.11 Effectiveness differentiable labels

A fair measure of effectiveness of differentiable label tokens would be to measure the performance of the method with and without differentiable tokens for each label. There would be two alternative approaches to using differentiable label tokens, they are 1) to use a verbalizer that maps labels to existing tokens or 2) using new fixed non differentiable tokens. The first alternative is particularly hard and not scalable for the following reasons:

- One label will most probably be split into multiple tokens. This can be due to the label being a phrase or due to the behavior of the tokenizer. In the Bugs dataset that we experimented with, File-System was one of the labels. This would be split into multiple tokens making it difficult to use a verbalizer. Another label was ReiserFS which was split into rei, ##ser and ##fs which again meant that using verbalizer was not feasible.
- The internal datasets we have are more technical in nature, which made the creation of a verbalizer even harder and not scalable.

The second alternative is sensitive to the choice of initialization.

Due to the limitations in the alternatives we decided to forgo an accuracy comparison as there was no viable alternatives.

We do propose the following set of experiments as future work to understand the differentiable tokens and what their embedding after training represents.

- An in-depth examination of how token embeddings capture semantic nuances in highly

specialized domains possibly by using similarity between embeddings or by visualization in a low dimension space.

- A study on a complex hierarchy to demonstrate the model's ability to distinguish between closely related labels.