ECAI 2024
U. Endriss et al. (Eds.)

doi:10.3233/FAIA240661

RADAr: A Transformer-Based Autoregressive Decoder Architecture for Hierarchical Text Classification

Yousef Younes [a,d,*], **Lukas Galke**[b,c] **and Ansgar Scherp**[d]

[a]GESIS – Leibniz-Institute for the Social Sciences
[b]Max Planck Institute for Psycholiguistics
[c]University of Southern Denmark
[d]University of Ulm
ORCID (Yousef Younes): https://orcid.org/0000-0003-1271-3633, ORCID (Lukas Galke): https://orcid.org/0000-0001-6124-1092, ORCID (Ansgar Scherp): https://orcid.org/0000-0002-2653-9245

Abstract. Recent approaches in hierarchical text classification (HTC) rely on the capabilities of a pre-trained transformer model and exploit the label semantics and a graph encoder for the label hierarchy. In this paper, we introduce an effective hierarchical text classifier RADAr (Transformer-based Autoregressive Decoder Architecture) that is based only on an off-the-shelf RoBERTa transformer to process the input and a custom autoregressive decoder with two decoder layers for generating the classification output. Thus, unlike existing approaches for HTC, the encoder of RADAr has no explicit encoding of the label hierarchy and the decoder solely relies on the samples' label sequences observed during training. We demonstrate on three benchmark datasets that RADAr achieves results competitive to the state of the art with less training and inference time. Our model consistently performs better when organizing the label sequences from children to parents versus the inverse, as done in existing HTC approaches. Our experiments show that neither the label semantics nor an explicit graph encoder for the hierarchy is needed. This has strong practical implications for HTC as the architecture has fewer requirements and provides a speed-up by a factor of 2 at inference time. Moreover, training a separate decoder from scratch in conjunction with fine-tuning the encoder allows future researchers and practitioners to exchange the encoder part as new models arise. The source code is available at https://github.com/yousef-younes/RADAr.

1 Introduction

Hierarchical Text Classification (HTC) deals with the task of labeling a text sample based on a semantic hierarchy where labels, i. e., classes, exhibit a generalization-specialization relationship [19, 30, 31, 28]. In contrast to multi-label text classification, where the labels are treated as a flat set [10], the models in HTC should exploit not only the text samples but also learn the labels' hierarchy during training.

Formally, in HTC, the set of labels H is organized according to a label hierarchy where the labels are connected with a parent-child relationship [28]. Each label represents a topical category, with the semantics of the relationship being that parent labels represent a broader category while child labels represent more specific categories. When the hierarchy H is organized as a tree, each label $c \in H$, except the root label, has one and only one parent. HTC approaches aim to predict a subset $L \subseteq H$ for a given text instance. This subset contains one or more labels relevant to the text, taking the hierarchy into account.

HTC approaches can be categorized into global and local methods based on how they utilize hierarchical information during training and how many classifiers they employ. Local approaches employ one classifier per label or per level [4, 29], whereas global approaches introduce a single classifier for all labels. Global approaches generally demonstrate superior performance compared to local methods [46]. Recent approaches in HTC often utilize a pre-trained transformer model as the foundation of their architecture [14, 37, 42].

Global HTC models such as [46, 9, 6] often model the label hierarchy and text separately, and then find a mixed representation for the text and labels. To do that, each text sample interacts with the entire label hierarchy, leading to unnecessary calculations [46, 6]. HGCLR [37] partially solves this problem by using Graphormer [41] to encode the hierarchy with the text and produce a hierarchy-aware text representation, which makes the Graphormer dispensable during testing [37]. A recent global model called HBGL [14] achieves state-of-the-art results by integrating hierarchical information into BERT. It first pre-trains label embeddings on random paths sampled from the hierarchy and subsequently employs conditional generation to predict the labels, given the text.

In addition, Large Language Models (LLMs) such as GPT-4 [1] have recently been used to tackle the HTC task modeled as a multiple choice question [8]. The study shows that even for a very small hierarchy of 17 labels, the model was not able to achieve more than 50% accuracy. Nevertheless, the literature on text classification shows that LLMs are strong text classifiers [45, 32, 44, 20, 43], especially given only a few in-context learning examples, but this performance comes with the extreme costs of dealing with large numbers of parameters while not generally outperforming smaller models. For instance, Yu et al. [43] confirm that by comparing the performance of the decoder-only models GPT-4 [1] and Llama 2 [34] with the encoder-only RoBERTa [22] on three classification tasks and concluded that encoder-only models are better.

The success of HTC models is largely attributed to incorporating linguistic knowledge found in the label names (label semantics) and explicitly encoding the label hierarchy using a graph encoder [6, 14].

* Corresponding Author. Email: yousef.younes@gesis.org

For instance, popular HTC approaches such as HiAGM [46], HiMatch [6], and HGCLR [37] employ graph neural networks to process the hierarchy, while HBGL uses the adjacency matrix of the hierarchy to constrain the attention mechanism of transformer layers. We question the necessity of such features and propose an effective sequence-to-sequence model RADAr (Transformer-based Autoregressive Decoder Architecture) based on an off-the-shelf pretrained transformer as an encoder and a comparably simple autoregressive decoder. Thus, there is no graph encoder for the label hierarchy. Our decoder learns the sequence of labels organized according to the hierarchy based on the samples. Since symbolic labels are used instead of the original text of the labels, they provide no additional knowledge to the trained models [28]. Thus, we do not exploit label semantics. Our main contributions are:

- We propose RADAr for hierarchical text classification. We use RoBERTa as an encoder and a custom autoregressive decoder with two decoder layers. Unlike existing approaches, RADAr does not need a graph encoder for the label hierarchy nor rely on label semantics.
- Experimental results on three HTC datasets confirm the effectiveness of the RADAr in obtaining results competitive to the state of the art with considerably less training and half inference time.
- Unlike existing HTC approaches, which sort the labels from parents to children, we show that the inverse order from children to parents consistently produces better results.

Below, we summarize the related work. Section 3 introduces the RADAr model. The experimental apparatus is described in Section 4. Section 5 reports an overview of the results. Section 6 discusses the results and analyzes the model, before we conclude.

2 Related Work

There are two types of HTC models, namely local and global ones [46]. Recently, the focus has been on global models thanks to the advent of word embeddings and pre-trained language models, which encourage general solutions. We focus on global methods, as the best-performing HTC models like [14, 37] belong to this type.

We present a set of models that were shown to be effective for HTC, irrespective of their initial design intent. These models are grouped into encoder-only and encoder-decoder based on their architecture, with chronological ordering within each category. We compare the models regarding five distinguishing features shown in Table 1. The features are the (i) use of the label hierarchy provided by the dataset (or considering the labels as set), (ii) employment of a graph encoder to represent the hierarchy, (iii) information on the order in which the model receives the labels, which is either parents to children or vice versa, (iv) use of label semantics, i. e., the textual description of the label, and (v) the used loss function.

Encoder-only Models Encoder-only transformer models like BERT [15] and RoBERTa [22] have shown superior classification performance in supervised settings [43]. They perform well on the HTC task even without considering the label hierarchy [10].

BERT performs the classification task by encoding the labels as a multi-hot vector and feeding them along with their associated texts into the model, which uses a classification head to do the prediction [15]. By this, BERT ignores the hierarchy i. e., treats the labels as a set like in a multi-label task.

RoBERTa is an optimized version of BERT that drops the next sentence objective and uses more data for training [22]. Due to their strong performance, the encoder-only models are used as a foundation by many HTC methods.

Another strong approach is HiAGM, which uses two encoders, one for text and one for encoding the label hierarchy [46]. The model provides two variants for the hierarchy encoder: a Tree-LSTM and GCN. The label dependencies are modeled bidirectionally, i. e., from parents to children and vice versa. As the loss function, this model combines recursive regularization [11] for the parameters of the final fully connected layer with binary cross-entropy (BCE).

HTCInfoMax builds on a variant of the HiAGM model with a modified loss function, which combines text-label Mutual Information Maximization (MIM), label prior matching, and BCE. The MIM helps to improve the text-label association. HTCInfoMax addresses the inherent label imbalance issue in HTC by considering statistical constraints on the structural encoder to improve the representation of low-frequency labels [9].

HiMatch uses GCN as its graph encoder [6]. It addresses the text-label association in terms of semantic matching. The system utilizes a three-component loss function. The joint embedding loss aligns text and label semantics, while the hierarchy-aware matching loss emphasizes semantic proximity of child labels to text semantics over parent labels. In addition, cross-entropy loss is used to learn the relationship between the text and the labels.

HGCLR [37] incorporates the hierarchy using a modified Graphormer into a BERT representation of the text [41]. It uses the label hierarchy to construct positive samples for contrastive learning. The NT-Xent loss [7] is applied to learn hierarchy-aware text representations by pulling together the text input close to its positive samples. In prediction, the model only needs the text encoder, not the Graphormer, so it becomes a BERT-encoder with classification head.

The state-of-the-art hierarchical text classifier, HBGL, utilizes the encoder-only BERT model in two steps. Initially, it pre-trains label vectors based on the hierarchy. Subsequently, the BERT model and these pre-trained label vectors are fine-tuned to predict the corresponding labels. This prediction occurs level-wise by constraining the attention matrix [14]. The training process employs BCE loss.

Encoder-Decoder Models An encoder-decoder model approaches the HTC task as a text generation problem. It comprises an encoder for processing the text input and a decoder for generating the corresponding labels.

SGM is a Bi-LSTM encoder-decoder model with attention mechanism [40]. The encoder uses the attention mechanism to produce a context vector focusing on informative words. The decoder uses the context vector, hidden state of the previous step, and label embedding vector to generate the hidden state of the current step. SGM has no graph encoder but benefits from label semantics and uses cross-entropy loss.

BART is a transformer-based encoder-decoder with an encoder similar to BERT and an autoregressive decoder similar to GPT [26]. Even though it does not have a graph encoder, we use it for HTC by framing the task as text generation since BART is effective when fine-tuned for text generation [18]. The model generates the labels in whatever order we use during fine-tuning but utilizes label semantics. It is trained using cross-entropy loss.

T5 is also a transformer-based encoder-decoder model that supports various NLP tasks in a text-to-text format with a different prefix for every task [27]. Like BART, T5 frames the HTC task as text generation, benefits from label semantics, and uses cross-entropy loss.

Similar to BERT and RoBERTa, the encoder-decoder models BART and T5 are not designed for HTC tasks. They do not provide an explicit graph encoder. Thus, for the hierarchical multi-label

Table 1. Model comparison. Graph encoder is the component the model uses to encode the label hierarchy. Label order refers to the order the model receives the labels. Label semantics indicates using the linguistic knowledge found in the label names. CE stands for Cross-Entropy

Model	Label Hierarchy	Graph Encoder	Label Order	Label Semantics	Loss Function	Description
Encoder-Only Baselines						
BERT	No	No	No	No	CE	Transformer Model
RoBERTa	No	No	No	No	CE	Transformer Model
Encoder-Decoder Baselines						
BART	No	No	No	Yes	CE	Transformer-based, denoising autoencoder
T5	No	No	No	Yes	CE	Transformer-based, text-to-text framework
Hierarchical Text Classifiers (based on pre-trained encoder-only models)						
HiAGM	Yes	Tree-LSTM, GCN	Parent-Child	Yes	BCE and Recursive Regularization	Obtains label-wise text features by fusing text classification model with a hierarchy encoder
HTCInfoMax	Yes	Tree-LSTM, GCN	Parent-Child	Yes	Text-label MIM, Label prior matching, and BCE.	Uses MIM to improve text-label association and label representation.
HiMatch	Yes	GCN	Parent-Child	Yes	Joint embedding, Hierarchy-aware matching, and CE	Models the text-label semantics relation as a semantic matching problem.
HGCLR	Yes	Modified Graphormer	Parent-Child	Yes	NT-Xent and BCE	Uses contrastive learning to produce hierarchy-ware text representation
HBGL	Yes	BERT	Parent-Child	Yes	BCE	Uses Attention Mask in self-attention layers to feed BERT with label graph
Hierarchical Encoder-Decoder Models						
SGM	Yes	No	Parent-Child	Yes	CE	Uses label correlations and text-label relations
Seq2Tree	Yes	No	Parent-Child	Yes	CE	Uses T5 with the decoder constrained by the hierarchy
RADAr	Yes	No	Child-Parent	No	Focal Cross Entropy	Uses RoBERTa as encoder and an autoregressive decoder on symbolic labels

classification task, the labels are treated as a set. We include these models as baselines since they have shown strong performance despite not considering the hierarchy information [10].

An approach considering the label hierarchy is Seq2Tree [42]. It is a variant of the T5-base model and captures the hierarchical information using the Depth-First Search (DFS) [33] over the hierarchy to linearize the labels. The decoder uses the encoder output to generate the DFS label sequence one at a time, considering the hierarchical information. In other words, the candidate labels for one step are the children of the parent predicted in the previous step.

3 The RADAr Model

In this section, we introduce the RADAr model depicted in Figure 1. RADAr is a transformer-based encoder-decoder model. Any encoder-only model can be used as the encoder of RADAr, but we choose to use RoBERTa base. The decoder is an autoregressive transformer-based model trained from scratch to generate a sequence of labels. The output of the next label is conditioned on the previous labels to generate the labels along the hierarchy for the HTC task. The model employs a modified version of focal loss [21] to pay more attention to difficult samples with high loss. In the following, we briefly describe the use of RoBERTa as an encoder. Then, we explain the decoder and the training and testing procedures.

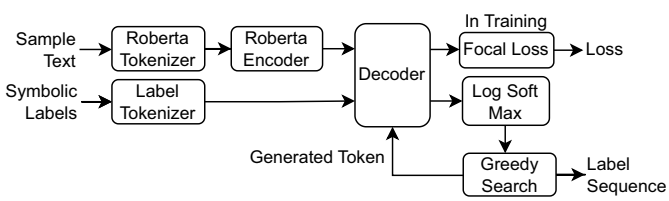

Figure 1. The RADAr Model Architecture

Encoder The model uses the RoBERTa base as its encoder part. The encoder is responsible for producing a fixed-size context tensor that captures the linguistic information found in the input text. To obtain the context tensor, the text is fed into the RoBERTa tokenizer to produce the token indices and attention masks. These are then fed into the RoBERTa model to produce an output. We use the last hidden state of that output, a tensor of size $(512, 768)$, as input to the decoder. For the decoder, we expand the dimensions of the attention masks to avoid attending to padding tokens.

Decoder The decoder is an autoregressive transformer model whose vocabulary corresponds to the label set. As such, the decoder's vocabulary is limited to the symbolic labels [28], i. e., internal label identifiers and special tokens (see Figure 2). Our decoder iteratively uses the previously generated tokens and the encoder's output as condition to generate the next token. It uses greedy search that picks the token with the highest probability at each decoding step without considering future consequences [39]. It stops when it reaches the special token $\texttt{</s>}$, indicating the end of the sequence.

In more detail, the decoder consists of embedding and position embedding layers, followed by decoder layers and a linear layer to produce the logits. The whole procedure of the decoder is shown in Algorithm 1. It receives four inputs during training: the sample's labels $\mathrm{L} \subseteq \mathrm{H}$, the labels' mask $\mathrm{l_m}$, the last hidden state of the encoder output h, and the encoder mask m. It starts by producing the absolute position embeddings $\mathrm{l_e}$ for the labels to capture the hierarchical information from the ordered label sequence. Then, it passes the obtained label embeddings $\mathrm{l_e}$ along with h, m, and $\mathrm{l_m}$ through a decoder layer which performs three operations. First, it computes the multi-head self-attention att of the label embedding. Next, it passes the obtained att through a layer normalization [3] and dropout [13] layers to compute the query q. After that, the encoder's last hidden state h is used as the key and value and sent together with the ob-

```
a)Original labels             : "Top/Features" "Top/Classifiers"
                                "Top/Features/Magazine" "Top/Features/Movies" "Top/Features/Theater" "Top/Classifieds/Job Market"
                                "Top/Features/Movies/News and Features" "Top/Classifieds/Job Market/Job Categories"
                                "Top/Classifieds/Job Market/Job Categories/Media, Entertainment and Publishing"
b)Symbolic Labels             : [a_23] [a_0] [a_35] [a_36] [a_42] [a_1] [a_37] [a_2] [a_14]
c)Level-wise organized        : [a_23] [a_0] <unk> [a_35] [a_36] [a_42] [a_1] <unk> [a_37] [a_2] <unk> [a_14] <unk>
d)Reversed level-wise         : ['[a_14]', '<unk>', '[a_2]', '[a_37]', '<unk>', '[a_1]', '[a_42]', '[a_36]', '[a_35]', '<unk>', '[a_0]', '[a_23]', '<unk>']
e)Tokenizer output:           : [ 1, 37,  3, 11, 87,  3, 10, 82, 49, 65,  3,  9,  5,  3,  2,  0,  0,  0, 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,...]
```

Figure 2. Label preprocessing and tokenization. Line a) contains the original labels. Line b) maps the original labels to the symbolic labels. Line c) adds the level separator token <unk>. Line d) organizes the labels level-wise from children to parents. Line e) contains the padded tokenizer output.

tained query and the encoder mask m to a transformer block to perform cross attention [36] and produce the output of the decoder layer. The encoder attention mask m controls the decoder attention on the encoder hidden states.

The output of a decoder layer is used as the label embedding for the next decoder layer that repeats the same operations. Furthermore, a linear layer is used to produce the logits on the output of the last decoder layer. Finally, the logits are used to compute the focal loss, a scaled cross-entropy that pays more attention to difficult samples.

Algorithm 1 Forward Pass of RADAr Decoder

1: **Input:** labels L, label masks l_m
   encoder output h, encoder mask m
2: **Output:** loss
3: $l_e \leftarrow \mathrm{Dropout}(\text{word-embed}(L) + \text{pos-embed}(L))$
4: **for** $i = 1$ **to** N_{layers} **do**
5: $\quad \text{att} \leftarrow \mathrm{MultiheadSelfAttention_i}(l_e, l_e, l_e, l_m)$
6: $\quad q \leftarrow \mathrm{Dropout}(\mathrm{LayerNorm_i}(\text{att} + l_e))$
7: $\quad l_e \leftarrow \mathrm{TransformerBlock_i}(h, h, q, m)$
8: **end for**
9: $\text{logits} \leftarrow \mathrm{linear}(l_e)$
10: $\text{loss} \leftarrow \text{cross-entropy}(L, \text{logits})$
11: $\text{loss} \leftarrow (1 - e^{-\text{loss}})^{\gamma} \cdot \text{loss}$ ▷ focal loss adjustments
12: **return** loss

Using the WOS dataset, we conducted pre-experiments regarding the number of decoder layers and the features of its transformer blocks [36]. We observe that the optimal choice is when the decoder has two decoder layers. Each layer has a transformer block with eight attention heads, a feedforward network and uses a dropout of 0.2.

Training and Testing The labels in HTC datasets are typically very imbalanced [35]. To compensate for this imbalance, we employ a batch-level focal loss [21], i. e., the model aims to optimize batches that are hard to classify. Those usually include samples with low-frequent labels in the training data.

We optimize the model to minimize cross-entropy on the batch level with the ground truth and apply focal loss scaling (line 11 in Algorithm 1). To achieve that, we calculate the model confidence on the batch $e^{-\text{loss}}$. Then, we compute the model perplexity $1 - e^{-\text{loss}}$. After that, we raise the perplexity to the power of the focusing parameter γ to get the modulating factor. We use the modulating factor to adjust the batch cross-entropy loss based on the batch difficulty. If the loss is high, the model confidence is low, and its perplexity is high, resulting in a higher final loss. On the contrary, when the loss is low, the confidence is high, and the perplexity is low, resulting in a lower final loss.

During testing, the decoder starts with the beginning-of-sequence token <s>, generates tokens autoregressively, one at a time, and stops generating labels after reaching the end-of-sequence token </s>. RADAr uses greedy search to generate the tokens, and using beam search [39] does not improve the results. When the model works on a batch of samples, the </s> token is predicted for each sample independently.

4 Experimental Apparatus

Datasets and Metrics We use three common HTC datasets. Those are Web-of-Science (WOS) [16], NY-Times (NYT) [29], and RCV1-V2 [17]. The WOS dataset has a hierarchy of two levels. It has exactly two labels per sample, one child and its parent, which makes it easier than the other two datasets. The NYT and RCV1-V2 datasets have eight-level and four-level hierarchies, respectively. The RCV1-V2 dataset has the largest test set compared to the others. Table 2 shows statistics about these datasets.

We measure the experimental results using Micro-F1 and Macro-F1 metrics. The impact of rare labels is higher for Macro-F1 score than for Micro-F1. For RADAr experiments, we report the mean and standard deviation over five runs with different random seeds.

Data Preprocessing The labels in the hierarchies of the datasets are usually provided with plain, human-readable text. Previous works [37, 14] tried to benefit from the linguistic knowledge found in the human-readable label names, known as label semantics [28]. On the contrary, our model does not rely on such label semantics.

Rather, we replace the original labels with symbolic labels. For example, the symbolic label `[a_14]` represents the label "Top/Classifieds/Job Market/Job Categories/Media, Entertainment and Publishing" in the NYT dataset as shown in Figure 2.

The label tokenizer's vocabulary consists of the set of symbolic labels of the dataset. The label tokenizer is responsible for tokenizing the input symbolic labels and generating a fixed-size vector whose size accommodates the maximum possible number of labels (not the average) for the dataset and other special tokens. This vector size differs from one dataset to another; it is 6, 48, 22 for the WOS, NYT, and RCV1-V2 datasets.

The tokenizer uses the <s> and </s> tokens to mark the beginning and end of the label sequence. It also uses the <pad> token to fill the rest of the vector after the </s> token. In addition, the <unk> token separates hierarchy levels.

The exact purpose of the <unk> token is demonstrated in an example shown in Figure 2. It shows the label preprocessing and tokenization on one label set corresponding to a sample from the NYT dataset. The line a) contains the original labels organized hierarchy level-wise from parents to children. Line b) contains the symbolic labels corresponding to the original labels. Line c) shows how the <unk> token separates the labels from different levels. Line d) represents the tokenizer's input and contains the labels organized level-wise from children to parents. Line e) expresses the tokenizer's output given line d) as input.

Procedure and Hyperparameters During training, we use the AdamW optimizer [23] with default settings and learning rates of $5 \cdot 10^{-5}$ and $3 \cdot 10^{-4}$ for the encoder and decoder, respectively. Two learning rates are necessary because the encoder is pre-trained while the decoder is trained from scratch. We also use the ReduceLROnPlateau scheduler [2] with patience of 3. After each epoch, the average validation loss is used to measure progress. If the validation loss does not decrease for three epochs, the scheduler multiplies the learning rates by a factor of 0.1 for both the encoder and decoder.

Table 2. Statistics of the datasets. $|H|$ is the number of Labels. D is the maximum level of hierarchy. $\mathrm{Avg}(L_i)$ is the average number of labels per sample. $\mathrm{Avg}(PL_i)$ is the average number of parent labels per sample. $\mathrm{Avg}(LL_i)$ is the average number of leaf labels per sample. These average fields show the statistics for train, dev, and test sets.

Dataset	$\|H\|$	D	$\mathrm{Avg}(L_i)$	$\mathrm{Avg}(PL_i)$	$\mathrm{Avg}(LL_i)$	Train	Dev	Test
WOS	141	2	2.0/2.0/2.0	1.0/1.0/1.0	1.0/1.0/1.0	30,070	7,518	9,397
NYT	166	8	7.6/7.6/7.5	5.9/5.9/5.9	1.7/1.6/1.7	23,345	5,834	7,292
RCV1-V2	103	4	3.18/3.18/3.24	1.98/1.98/2.02	1.20/1.21/1.23	20,833	2,316	781,265

The encoder's learning rate decreases until it is lower than $5 \cdot 10^{-7}$. At that point, the encoder is frozen, and the decoder continues to learn alone. The training loop has a patience of 10, so it stops if the average validation loss does not decrease for ten consecutive epochs. We use focal loss with focusing parameter $\gamma = 2$ as described in Section 3. Focal loss depends on cross-entropy with label smoothing of 0.1. We use teacher forcing [38] and gradient accumulation over 2 batches (32 examples each). For reproducibility, we use a fixed random seed. With these settings, the model took 29, 30, and 40 training epochs on the WOS, NYT, and RCV1-V2 datasets, respectively. With a patience of 10, the model saturates at 23 epochs on average. The experiments were implemented in PyTorch and run on an NVIDIA A100-SXM4-40GB GPU.

The hyperparameter values like learning rates, batch size, focusing parameter, and label smoothing are selected per dataset based on the validation sets using wandb [5] as follows. We specify ranges for the values depending on the common values from the literature and also based on our pre-experiments. For example, $5 \cdot 10^{-5}$ is known to be a reasonable learning rate for RoBERTa [10], so we specify the range $[5 \cdot 10^{-4}, 5 \cdot 10^{-6}]$ to contain that value. Then, using these ranges, we initialize each hyperparameter randomly over 20 experiments and select the best value. We use the best values for individual hyperparameters and test all their possible combinations, i. e., grid search [5], again on the validation sets. The values used to select the hyperparameters are in the code repository associated with the paper.

5 Results

Table 3 shows Micro-F1 and Macro-F1 on three datasets. We compare our model to the models from the literature (see Section 2). Some hierarchical text classification models like HiMatch and HiAGM did originally not use pre-trained language models as text encoder. Yet, the HGCLR paper [37] shows that replacing the text encoder in such models with a BERT model leads to improved results, which we use for our comparison.

Table 3 is organized into five groups, depending on the models' architecture. The first group reports the results for the encoder-only baseline models BERT [15] and RoBERTa [22], which treat the labels as a flat set. The second group includes encoder-decoder baseline models T5 [27] and BART [18], which yield lower performance than the encoder-only baselines. The third group comprises HTC models using encoder-only models, namely HiAGM [46], HTCInfoMax [9], HiMatch [6], HGCLR [37], and HBGL [14]. We report the results of HiAGM, HTCInfoMax, and HiMatch models with their original text encoder replaced with BERT. The highest scoring model in this group is HBGL. We report the results of HBGL using both BERT and RoBERTa.

The fourth group reports the results of the following hierarchical encoder-decoder models SGM, SGM-T5, and Seq2Tree. SGM uses its own vocabulary and trains its word embedding from scratch [40]. Nevertheless, SGM beats all other models on the Macro-F1 score of the NYT dataset. SGM-T5 outperforms SGM. It is a modified version of SGM that uses T5 as its encoder-decoder backbone [42]. The Seq2Tree model is the best-performing model in this group. It even achieves a Macro-F1 score better than all other models on the WOS dataset. Because SGM and Seq2Tree models outperform HBGL only in isolated metrics across specific datasets, we regard HBGL as the current state-of-the-art.

The fifth group contains the results of our RADAr model. It outperforms the encoder-only baselines by achieving improvements in Macro-F1 of 2.77%, 3%, and 2.62% on WOS, NYT, and RCV1-V2, respectively. Compared to HBGL, the RADAr model achieves a similar Micro-F1 score on the RCV1-V2 dataset. For the other two datasets, it achieves results with differences ranging from 0% to 1.55% in all measures. Here, we should indicate that we compare RADAr to HBGL with the BERT encoder since it is the one used in the original paper. However, using RoBERTa as HBGL's encoder does not make a big difference in the results. Nevertheless, including those results is important to highlight the competitiveness of RADAr w. r. t. HBGL. Finally, our RADAr model outperforms most HTC models, while it does not use label semantics or a graph encoder for the hierarchy.

6 Discussion

The results show that RADAr is competitive with other HTC methods, providing a strong indication that an explicit graph encoder is not needed as long as the labels are ordered according to the hierarchy. This conceptual advantage has an immediate effect on training and inference times. Although RADAr has more parameters than HBGL (137M vs. 110M), it requires less time for training and inference. Averaged over our three datasets, HBGL needs 48 epochs, each taking 11.67 minutes, while RADAr needs 33 epochs, each taking 8.7 minutes. Comparing inference times, we found that HBGL requires approximately 135, 184, and 12,406 seconds compared to 74, 82, and 6,716 on the WOS, NYT, and RCV1-V2 test sets, respectively. On average, RADAr provides a speed-up of 97.03% for inference time relative to its strongest competitor HBGL – effectively doubling the throughput for practical applications.

The primary distinction of our RADAr model from previous models is that it does not use a graph encoder or label semantics. Instead, RADAr captures hierarchical information from the organization of the label sequences it is trained on. To understand the impact of label organization and the components of RADAr, we conduct an ablation study (Section 6.1) and an error analysis (Section 6.2).

6.1 *Ablation Study*

We analyze the different components of RADAr to understand their impact. The ablation study covers different variants of label sequence organization, the effect of focal loss, the effect of label semantics, and employing different text encoders. Table 4 summarizes the results of our ablation studies, beginning with the best scores achieved with our RADAr, which we describe in the following.

Table 3. Hierarchical text classification results. The RADAr model results are the average over five runs and are reported along with the standard deviation. For numbers from the literature, "-" means not available in the original paper.

Model	WOS		NYT		RCV1-V2		Provenance
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1	
Encoder-Only Baselines							
BERT-base	85.63	79.07	78.24	65.62	85.65	67.02	[37]
RoBERTa	-	-	77.05	55.53	86.99	62.29	[10]
Encoder-Decoder Baselines							
BART	84.08	77.43	19.21	6.49	86.20	65.11	our experiment
T5	82.03	74.62	46.71	20.06	84.9	57.01	our experiment
Hierarchical Encoder-Only Classifiers							
HGCLR	87.11	81.20	78.86	67.96	86.49	68.31	[37]
BERT+HiAGM	86.04	80.19	78.64	66.76	85.58	67.93	[37]
BERT+HTCInfoMax	86.30	79.97	78.75	67.31	85.53	67.09	[37]
BERT+HiMatch	86.70	81.06	-	-	86.33	68.66	[6]
HBGL+RoBERTa	87.66	81.96	79.98	70.57	87.52	70.52	our experiment
HBGL+BERT	**87.36**	82.00	**80.47**	70.19	**87.23**	**71.07**	[14]
Hierarchical Encoder-Decoder Models							
SGM	67.74	74.01	64.68	**72.78**	71.85	35.29	our experiment
SGM-T5	85.83	80.79	-	-	84.39	65.09	[42]
Seq2Tree	87.20	**82.50**	-	-	86.88	70.01	[42]
RADAr	$87.17_{(0.04)}$	$81.84_{(0.08)}$	$79.84_{(0.07)}$	$68.64_{(0.28)}$	$87.23_{(0.05)}$	$69.64_{(0.12)}$	our experiment
difference to HBGL	-0.19	-0.16	-0.63	-1.55	0.0	-1.43	

Table 4. The ablation experiment's nomenclature reflects feature deviation from our RADAr model. RADAr features include using RoBERTa as an encoder and focal loss on the batch level. It also include working on labels ordered from child-parent hierarchy level-wise using the <unk> token to separate labels from different levels and without using label semantics.

Experiment	WOS		NYT		RCV1-V2	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
RADAr model...	87.19	81.84	79.88	**69.09**	87.22	69.48
... but parent-to-child ordering	86.55	81.20	79.22	66.43	87.04	68.57
... w/o <unk> token separators	**87.20**	**81.85**	76.28	65.40	85.88	64.05
... using <unk> to separate paths instead of levels	87.19	81.84	77.35	62.95	79.35	63.64
... but shuffled labels w/o <unk>	86.16	80.77	67.08	47.50	86.12	64.30
... children only + hierarchy	87.11	81.61	**79.94**	68.49	87.13	**69.61**
... focal loss on label level	86.64	81.10	80.11	69.0	87.10	69.36
... w/o focal loss	87.01	81.41	79.55	68.37	**87.25**	69.40
... with labels semantics	86.58	81.49	79.60	67.52	86.96	69.51
... BERT Encoder	86.51	80.77	78.93	67.19	87.05	68.64
... XLNET-base Encoder	86.98	81.60	79.32	67.26	86.60	68.15
... Sentence Transformer Encoder	86.76	81.15	76.77	63.89	84.68	62.31

First, we compare the results of organizing the label sequence from parents to children versus children to parents (see also the example in Figure 2). Our ablation demonstrates that ordering the labels from parents to children decreases all measures over all datasets. This drop reaches its maximum 2.66% less Macro-F1 score on the NYT. Ordering the labels children to parent positively affects all measures. This effect relates to the depth of the hierarchy, i. e., the deeper the hierarchy is, the more remarkable the effect is.

The experiment without the use of the <unk> token as a separator (see line " w/o <unk> ") shows a small negative impact on the WOS dataset but is essential for the NYT and RCV1-V2 datasets. In this experiment, the model runs on the labels organized level-wise from child to parent but does not use the <unk> token to indicate the different levels of the hierarchy. The small effect on WOS since is expected because WOS has exactly two labels: one parent and one child. Adding the <unk> token is not very helpful in this case.

The RADAr model uses <unk> as a level separator to delimit different hierarchy levels. Instead one could use the <unk> token to separate different paths of the same sample (see line "using <unk> to separate paths instead of levels"). The model runs on the full paths, where a path is constructed starting from a child label up to the root. As the results show, organizing the labels in this way makes the problem more difficult for datasets with complex hierarchies. Again, it does not affect the WOS dataset, which only has a two-level hierarchy and only one assigned leaf label per document.

To show the influence of organizing the label sequence, the "shuffled labels w/o <unk>" experiment fine-tunes the model on shuffled label sequences. The results show that organizing the labels is essential for the model to perform well. Since the model cannot pick up a random but fixed order of labels, it also shows that the order in terms of the child to parent actually is a simplified form of entailment where the child is the premise and its parent is the hypothesis. In other words, the model is doing a form of inference on the symbolic labels [24]. Again, the effect of organized labels depends on the depth of the label hierarchy in the dataset. For example, the Macro-F1 score drops by 1.07%, 5.18%, and 21.59% for the WOS, RCV1-V2, and NYT datasets, respectively. This also indicates that the deeper the hierarchy is, the more effect the label organization has.

Next, we simplify the HTC task using the label hierarchy since every child label has only one parent. If the child label is predicted, we can use the hierarchy to extract all its parents up to the root. The "children only + hierarchy" experiment runs the model

on the datasets by removing all the labels that are extractable from the hierarchy based on other labels. For every sample, we start from the children labels in the sequence and include a label if it has no children in the sequence. Applying this method on line d) of Figure 2 results in `[[a_14], <unk>, [a_37], <unk>, [a_42], [a_35], <unk>]`, which contains the minimum label set necessary to reconstruct line d) using the hierarchy. Such a scenario is feasible if the hierarchy is available at test time, which is commonly the case as these hierarchies are typically openly available [12, 25]. After the model generates the minimized label set, we use the hierarchy to extract the remaining labels up to the root. The model performs well in this scenario and produces a slightly better Micro-F1 score on the NYT dataset than our best-performing model. This improvement is due to the problem's simplification, which is more noticeable given the complexity of the NYT dataset.

The third group of experiments in Table 4 investigates the effect of focal loss. In the "focal loss on label level" experiment, the model runs with the focal loss computed on every label. That increased the Micro-F1 score of the NYT dataset but decreased other measures. The "w/o focal loss" experiment removes the adjustment of focal loss, which seems to have a good effect on the RCV1-V2 dataset. For that dataset, the Micro-F1 is slightly better than the state-of-the-art in Table 3 but worse than our best-performing model on other measures. In general, we conclude that it is helpful for the RADAr model to pay attention to the difficult samples, but too much attention has a negative impact. Thus, focal loss on batch level is the best choice.

The "with labels semantics" experiment uses the linguistic information in the label names, as done in previous work [14]. We exploit the label semantics by initializing the decoder with RoBERTa embeddings of the original label text. For each symbolic label, we feed its associated text into RoBERTa to obtain token embeddings. We compute the mean over these token embeddings and use it to initialize the corresponding vector in the decoder's embedding and output layer. Exploiting label semantics results in slightly lower scores than our best-performing model. Perhaps the reason is that the decoder is simpler than the encoder and uses a different vocabulary, so using label semantics in the decoder has a negative impact.

Lastly, we examine the effect of using a different encoder. Using BERT instead of RoBERTa results in around 1% decrease on all measures. We have also evaluated different encoder models like Sentence Transformer and XLNET-base, but RoBERTa performed best.

6.2 *Error Analysis*

This section analyzes the type of errors our RADAr model produces. The examples presented here are from our best-performing model.

WOS The hierarchy of this dataset is only two levels, with each sample having exactly two labels, one from each level. In $1,630$ out of $9,397$ test samples, the RADAr model made one or more mistakes while generating labels. Among them were 852 cases in which the model mispredicted the child label, but the parent label was correct. For example, on one sample, the model predicted `[[a_11], <unk>, [a_79],<unk>]` but the ground truth is `[[a_72], <unk>, [a_79], <unk>]`. Looking at the hierarchy, we found that the predicted child label `[a_11]` and the corresponding ground truth label `[a_72]` share the same parent `[a_79]`, which explains why the model was able to get the parent right. For the remaining 778 wrong cases, the model mispredicted both labels. So, the model always predicts the parent label correctly, but when it fails to predict the correct child label, it builds on that error and mispredicts its parent. This problem is known as exposure bias [40]. In other words, the model has learned the dataset's child-to-parent relationships but faces difficulties generating the correct child labels in the first place.

NYT Unlike WOS, the NYT dataset has an eight-level hierarchy. We observe different types of errors in $3,791$ out of $7,292$ test samples. Among the errors, the model generates shorter sequences for $1,805$ samples and longer sequences for $1,699$ samples, while only 287 samples have the same number of tokens as the gold standard. This type of error indicates that the model has difficulty generating the end-of-sequence token correctly. Let us further investigate the three cases. For shorter sequence cases, the model fails to predict a label that starts a new path somewhere toward the end of the sequence as in the following ground truth sequence: `[[a_52], <unk>, [a_48], [a_30], <unk>, [a_47], [a_29], <unk>, [a_46], [a_36], [a_24], <unk>, [a_23], <unk>]`. In this label sequence, the model fails to predict the label `[a_36]`, which comes before the root node with no previous children, but correctly predicts two labels after the first generated label. This phenomenon indicates that the model depends on the sample text at the beginning of the sequence but gradually shifts its focus to depend on the predicted labels as it generates more of them.

For longer sequences, the model fails to select the correct starting level from the eight hierarchy levels. For example, given the ground truth sequence `[[a_99], <unk>, [a_85] <unk>]`, it predicts `[[a_12], <unk>, [a_2], <unk>, [a_1][a_99], <unk>,[a_0][a_85], <unk>]`, which includes the ground truth but adds extra labels. This shows that the model captures the hierarchy but generates labels from deeper levels than necessary.

Finally, for sequences with the same length, the model suffers from the exposure bias problem like on the WOS dataset.

RCV1-V2 This dataset has a four-level hierarchy. The model makes different errors generating the labels in $260,415$ out of $781,265$ test samples. Among them, it generates shorter sequences for $124,622$ samples and longer sequences for $104,336$ samples, while only $31,457$ samples have the same length as the gold standard. Investigating these three types of mistakes, we find similar errors to those of NYT. When comparing the errors with the ones on the NYT dataset, we notice that the model, in general, erroneously produces many shorter sequences than longer ones.

7 Conclusion and Future Work

The paper introduces RADAr, a sequence-to-sequence model using RoBERTa as an encoder and an autoregressive decoder for the HTC task. It does not have an encoder for the label hierarchy but learns it from the label sequences during training. The model shows that organizing label sequences from children to parents, rather than the opposite order [37, 14, 46, 9, 6] (see Table 1), is more effective. RADAr demonstrates that label semantics or encoding the label hierarchy are not necessary for good performance. As a result, RADAr is a flexible model with fewer requirements and easy-to-replace components. In future work, we plan to further investigate the exposure bias problem and also evaluate the model on non-hierarchical multi-label classification tasks.

Limitations

Our RADAr model is an English model because we have fine-tuned it on three English benchmark datasets. Nevertheless, the model is extendable to other languages since the decoder uses symbolic labels and does not utilize label semantics. Also, the encoder could be easily replaced by a multilingual transformer encoder.

Acknowledgements

This work was co-funded by the DFG as part of the UnknownData Project - Grant No. 460676019.

References

[1] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] A. Al-Kababji, F. Bensaali, and S. P. Dakua. Scheduling techniques for liver segmentation: ReduceLRonPlateau vs OneCycleLR. In *Intelligent Systems and Pattern Recognition*, pages 204–212, Cham, 2022. Springer.

[3] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[4] S. Banerjee, C. Akkaya, F. Perez-Sorrosal, and K. Tsioutsiouliklis. Hierarchical transfer learning for multi-label text classification. In *ACL*, pages 6295–6300, 2019.

[5] L. Biewald. Experiment tracking with weights and biases, 2020. URL https://www.wandb.com/. Software available from wandb.com.

[6] H. Chen, Q. Ma, Z. Lin, and J. Yan. Hierarchy-aware label semantics matching network for hierarchical text classification. In *ACL-IJCNLP*, pages 4370–4379. ACL, 2021. doi: 10.18653/v1/2021.acl-long.337.

[7] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[8] A. De Silva, J. L. Wijekoon, R. Liyanarachchi, R. Panchendrarajan, and W. Rajapaksha. AI insights: A case study on utilizing chatgpt intelligence for research paper analysis. *arXiv preprint arXiv:2403.03293*, 2024.

[9] Z. Deng, H. Peng, D. He, J. Li, and P. Yu. HTCInfoMax: A global model for hierarchical text classification via information maximization. In K. Toutanova, A. Rumshisky, L. Zettlemoyer, D. Hakkani-Tur, I. Beltagy, S. Bethard, R. Cotterell, T. Chakraborty, and Y. Zhou, editors, *In Proceedings of NAACL: HLT*, pages 3259–3265. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.naacl-main.260.

[10] L. Galke, A. Diera, B. X. Lin, B. Khera, T. Meuser, T. Singhal, F. Karl, and A. Scherp. Are we really making much progress in text classification? A comparative review. *arXiv preprint arXiv:2204.03954*, 2023.

[11] S. Gopal and Y. Yang. Recursive regularization for large-scale classification with hierarchical and graphical dependencies. In *SIGKDD*, pages 257–265, 2013.

[12] G. Große-Bölting, C. Nishioka, and A. Scherp. A comparison of different strategies for automated semantic document annotation. In *K-CAP 2015*, pages 8:1–8:8. ACM, 2015. doi: 10.1145/2815833.2815838.

[13] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[14] T. Jiang, D. Wang, L. Sun, Z. Chen, F. Zhuang, and Q. Yang. Exploiting global and local hierarchies for hierarchical text classification. In *EMNLP*, pages 4030–4039, Abu Dhabi, United Arab Emirates, Dec. 2022. ACL.

[15] J. D. M.-W. C. Kenton and L. K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186, 2019.

[16] K. Kowsari, D. E. Brown, M. Heidarysafa, K. J. Meimandi, M. S. Gerber, and L. E. Barnes. HDLTex: Hierarchical deep learning for text classification. In *2017 16th IEEE international conference on machine learning and applications (ICMLA)*, pages 364–371. IEEE, 2017.

[17] D. D. Lewis, Y. Yang, T. Russell-Rose, and F. Li. RCV1: A new benchmark collection for text categorization research. *Journal of machine learning research*, 5(Apr):361–397, 2004.

[18] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *ACL*, pages 7871–7880, Online, July 2020. ACL. doi: 10.18653/v1/2020.acl-main.703.

[19] Q. Li, H. Peng, J. Li, C. Xia, R. Yang, L. Sun, P. S. Yu, and L. He. A survey on text classification: From traditional to deep learning. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 13:1 – 41, 2020. URL https://api.semanticscholar.org/CorpusID:220961531.

[20] X. Li, S. Chan, X. Zhu, Y. Pei, Z. Ma, X. Liu, and S. Shah. Are ChatGPT and GPT-4 general-purpose solvers for financial text analytics? a study on several typical tasks. In *Conference on Empirical Methods in Natural Language Processing*, 2023.

[21] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. In *ICCV*, pages 2980–2988, 2017.

[22] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[23] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[24] B. MacCartney. *Natural language inference*. Stanford University, 2009.

[25] F. Mai, L. Galke, and A. Scherp. Using deep learning for title-based semantic subject indexing to reach competitive performance to full-text. In *JCDL*, pages 169–178. ACM, 2018. doi: 10.1145/3197026.3197039.

[26] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[27] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.

[28] F. Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47, 2002.

[29] K. Shimura, J. Li, and F. Fukumoto. HFT-CNN: Learning hierarchical category structure for multi-label short text categorization. In *EMNLP*, pages 811–816, Brussels, Belgium, Oct.-Nov. 2018. ACL. doi: 10.18653/v1/D18-1093.

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

[31] A. Sun and E.-P. Lim. Hierarchical text classification and evaluation. In *Proceedings 2001 IEEE ICDM*, pages 521–528. IEEE, 2001.

[32] X. Sun, X. Li, J. Li, F. Wu, S. Guo, T. Zhang, and G. Wang. Text classification via large language models. *arXiv preprint arXiv:2305.08377*, 2023.

[33] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972. doi: 10.1137/0201010.

[34] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[35] I. Vagliano, L. Galke, and A. Scherp. Recommendations for item set completion: on the semantics of item co-occurrence with data sparsity, input size, and input modalities. *Inf. Retr. J.*, 25(3):269–305, 2022. doi: 10.1007/S10791-022-09408-9.

[36] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems (NeurIPS)*, 30, 2017.

[37] Z. Wang, P. Wang, L. Huang, X. Sun, and H. Wang. Incorporating hierarchy into text encoder: a contrastive learning approach for hierarchical text classification. In *ACL*, pages 7109–7119, Dublin, Ireland, May 2022. ACL. doi: 10.18653/v1/2022.acl-long.491.

[38] R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989. doi: 10.1162/neco.1989.1.2.270.

[39] C. Wilt, J. Thayer, and W. Ruml. A comparison of greedy search algorithms. In *Proceedings of the International Symposium on Combinatorial Search*, volume 1, pages 129–136, 2010.

[40] P. Yang, X. Sun, W. Li, S. Ma, W. Wu, and H. Wang. SGM: Sequence generation model for multi-label classification. In *COLING*, pages 3915–3926, Santa Fe, New Mexico, USA, Aug. 2018. ACL.

[41] C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T.-Y. Liu. Do transformers really perform badly for graph representation? In *NeurIPS*, volume 34, pages 28877–28888. Curran Associates, Inc., 2021.

[42] C. Yu, Y. Shen, and Y. Mao. Constrained sequence-to-tree generation for hierarchical text classification. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1865–1869, 2022.

[43] H. Yu, Z. Yang, K. Pelrine, J. F. Godbout, and R. Rabbany. Open, closed, or small language models for text classification? *arXiv preprint arXiv:2308.10092*, 2023.

[44] L. Yuan, Y. Chen, G. Cui, H. Gao, F. Zou, X. Cheng, H. Ji, Z. Liu, and M. Sun. Revisiting out-of-distribution robustness in NLP: Benchmark, analysis, and LLMs evaluations. *arXiv preprint arXiv:2306.04618*, 2023.

[45] Y. Zhang, M. Wang, C. Ren, Q. Li, P. Tiwari, B. Wang, and J. Qin. Pushing the limit of LLM capacity for text classification. *CoRR*, abs/2402.07470, 2024. doi: 10.48550/ARXIV.2402.07470.

[46] J. Zhou, C. Ma, D. Long, G. Xu, N. Ding, H. Zhang, P. Xie, and G. Liu. Hierarchy-aware global model for hierarchical text classification. In *ACL*, pages 1106–1117, 2020. doi: 10.18653/v1/2020.acl-main.104.