CascadeXML: End-to-end Multi-Resolution Learning for Extreme Multi-Label Text Classification

Anonymous Author(s)
Affiliation
Address
email

Abstract

Extreme Multi-label Text Classification (XMC) involves learning a classifier that can assign an input with a subset of most relevant labels from millions of label choices. Recent approaches, such as XR-Transformer and LightXML, leverage a transformer instance to achieve state-of-the-art performance. However, in this process, these approaches need to make various trade-offs between performance and computational requirements. A major shortcoming, as compared to the Bi-LSTM based AttentionXML, is that they fail to keep separate feature representations for each resolution in a label tree. We thus propose CascadeXML, an end-to-end multi-resolution learning pipeline, which can harness the multi-layered architecture of a transformer model for attending to different label resolutions with separate feature representations. CascadeXML significantly outperforms all existing approaches with non-trivial gains obtained on benchmark datasets consisting of up to three million labels. Code for CascadeXML will be made publicly available.

1 Introduction

Extreme multi-label text classification (XMC) deals with the problem of predicting the most relevant subset of labels from an enormously large label space spanning up to millions of labels. Over the years, extreme classification has found many applications in e-commerce, like product recommendation [7] and dynamic search advertisement [14], document tagging [1] and open-domain question answering [6, 24] and thus, practical solutions to this problem can have significant and far-reaching impact.

The output space in extreme classification is not only large but also, the labels follow Zipf's law where most of the labels are *tail labels* [2, 15]. For example, in a dataset of Wikipedia corpus containing 500K labels, only 2.1% labels annotate more than 100 data points and 60% of the labels annotate less than 10 data points! This inherently makes learning meaningful representations at an extreme scale a challenging problem. Further, as the memory and computational requirements grow linearly with the size of the label space, naive treatment by the application of classical methods and off-the-shelf solvers fails to deal with such large output spaces [8, 48].

In extreme scenarios with millions of labels, a *surrogate* step of shortlisting candidate labels becomes crucial to perform extreme classification and has been adapted by many popular approaches [10, 17, 21, 46, 51]. For each data point, these approaches first solve a simpler task with coarse label-clusters or *meta-labels* as the label space to create a shortlist of relevant labels for the extreme classification task. This effectively reduces the training time and computational complexity of the extreme task to that of the shortlisting step which is $\mathcal{O}(\sqrt{L})$, where L is the number of labels.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

	AttentionXML	LightXML	XR-Transformer	CascadeXML
Base Model	Bi-LSTM	Transformer	Transformer	Transformer
Attention Maps	adaptive	shared	shared	adaptive
Single Encoder	×	✓	✓	✓
Multi-level HLT	✓	×	✓	✓
End-to-End	×	✓	×	✓
Full Resolution	✓	✓	×	✓

Table 1: Strengths and limitations of current XMC methods: LightXML makes end-to-end training possible, but only admits a single level of the HLT. XR-Transformer allows for multiple levels, but its iterative feature learning does not train the transformer encoder at the highest resolution at all [51, Table 6]. However, by changing from multiple Bi-LSTMs to a single transformer, both LightXML and XR-Transformer lost AttentionXML's ability to adapt attention maps to each resolution.

Existing works leverage label representation as a centroid of their annotated instances to create a Hierarchical Label Tree (HLT) [34, 49]. Works like [8, 10, 17, 21, 46] use label clusters at a certain level of the HLT as meta-labels which are in turn used to shortlist candidate labels for the extreme task. In contrast, XR-Transformer [51] and AttentionXML [49] leverage multiple levels of the HLT, such that each level corresponds to a certain label *resolution* in the tree structure [51].

1.1 Motivation: Strengths and Weaknesses of Current Approaches

The earliest work to successfully combine label-tree based shortlisting and attention-based deep encoders is AttentionXML [49], which employs multiple Bi-LSTMs models that are trained sequentially for each tree resolution. More recent approaches [8, 17, 46, 51] replaced the model architecture with a more powerful transformer model [41] and fine-tune a pre-trained instance such as Bert [11]. Without careful designs, such models are computationally very expensive and earlier works [8, 46] could not effectively leverage transformers for both computation and performance on XMC tasks. Next, we discuss two contemporary works LightXML [17] and XR-Transformer [51].

LightXML introduces the concept of dynamic negative sampling, which replaces pre-computed label shortlists with a dynamically calculated shortlist that changes as the model's weights get updated. This enables end-to-end training with a single model by using the final feature representation of the transformer encoder for both the meta- and the extreme-classification task. Unfortunately, these two tasks appear to interfere with one another [21]. We hypothesize that this is because the meta task needs the attention maps to focus on different tokens than the extreme task. Furthermore, it only uses a single-level tree, which prevents scaling to the largest datasets.

XR-Transformer takes inspiration from multi-resolution approaches in computer vision like super resolution [23] and progressive growing of GANs [18, 19], and enables multiple resolutions through iterative training. However, unlike progressively grown GANs, which predict only at the highest resolution, XR-Transformer needs predictions across all resolutions for its progressive shortlisting pipeline, but uses representations trained at a single resolution. In practice, this leads to XR-Transformer having a complex multi-stage pipeline where the transformer model is iteratively trained up to a certain resolution and then frozen. This is followed by a *re-clustering* and *re-training* of multiple classifiers, working at different resolutions, with the same *fixed* transformer features [51, Alg. 2]. For datasets with over 500K labels, the transformer is only trained on up to 2^{15} clusters [51, Table 6], and the resulting features are used for the extreme task.

Unlike AttentionXML, using multiple instances of transformer models becomes undesirable due to their computational overhead. This pushes LightXML and XR-Transformer to make different trade-offs when leveraging a single transformer model for XMC tasks compared to AttentionXML, see Table 1. In this paper, we present a method, *CascadeXML*, that combines the strengths of these approaches, thus creating an end to-end trainable multi-resolution learning pipeline which trains a single transformer model across multiple resolutions in a way that allows the creation of label resolution specific attention maps and feature embeddings.

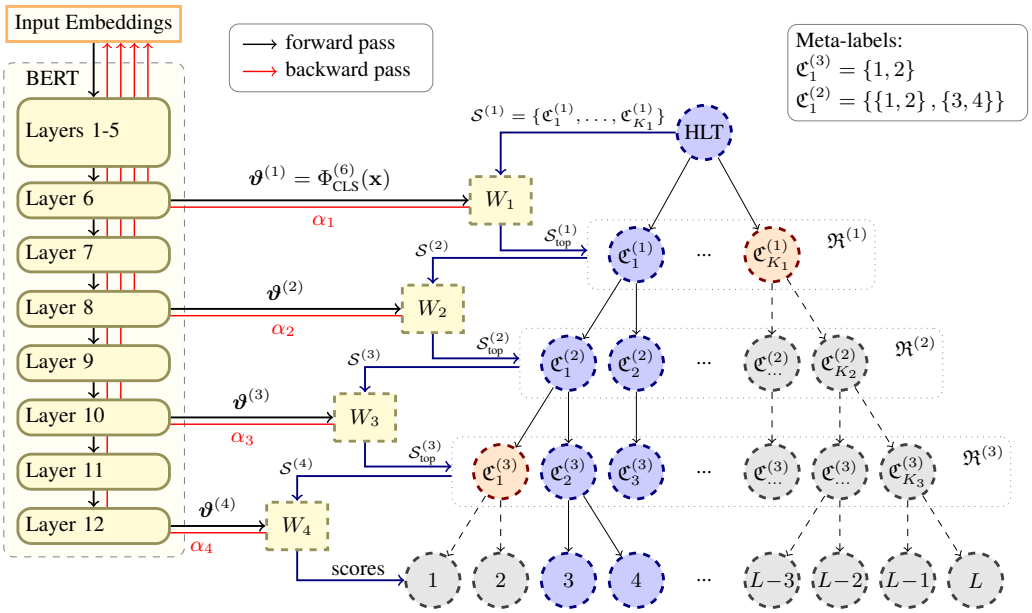

Figure 1: Overview over CascadeXML. The meta-classifiers W_1 - W_3 use intermediate BERT features $\boldsymbol{\vartheta}^{(t)}$ to discard all meta-labels except the highest scoring at that level (marked in red), so their descendants (gray) need not be considered at the next level. More details in section 3 and algorithm 1.

1.2 CascadeXML in a Nutshell

The key insights that enable calculating resolution-specific representations with a single forward pass through the transformer are **a)** that the shortlisting tasks get easier for lower label resolution and **b)** that the intermediate transformer representations are already quite powerful discriminators [43]. Consequently, CascadeXML extracts features for classification at the coarser meta-label resolution from earlier transformer layers and uses the last layer only for classification at the extreme resolution. The advantage of this is twofold: First, we postulate that the gain in classification accuracy due to enabling different attention maps at the different tree resolutions outweighs the loss in accuracy due to the slightly less powerful representations at the earlier transformer layers. Second, this ensures that the representation capacity of the later layers is exclusively available to the more difficult tasks.

The main idea of CascadeXML is illustrated in Figure 1. An input text is propagated through the transformer model until the first shortlisting level is reached. There, the embedding $\boldsymbol{\vartheta}^{(1)}$ corresponding to the [CLS] token is extracted, and used to determine scores for each level-1 meta-label. The highest-scoring meta-labels are selected to create a shortlist of candidate labels $\mathcal{S}_{\text{top}}^{(1)}$. The input is propagated further to perform another shortlisting step at the next resolution, at which a new [CLS] embedding $\boldsymbol{\vartheta}^{(2)}$ is extracted. Crucially, $\boldsymbol{\vartheta}^{(2)}$ is the result of a different self-attention operations, and its attention map is a differently weighted combination of the embeddings of other tokens than that of $\boldsymbol{\vartheta}^{(1)}$. The new embedding $\boldsymbol{\vartheta}^{(2)}$ is used to refine the shortlist by selecting the top-scoring level-2 meta-labels that are children of the shortlisted level-1 meta-labels. This process is repeated until at the final layer of the transformer, where classification is performed at the full label resolution.

Contributions In this paper, our contributions are the following:

- We introduce a novel paradigm which can harness the multi-layered architecture of transformers for learning representations at multiple resolutions in an HLT. This is in contrast to the existing approaches which treat transformer as merely a black box encoder,
- Implemented via CascadeXML, it is an end-to-end trainable multi-resolution learning pipeline which is quick to train and simple to implement. It eliminates the need of *bells*

and whistles like multi-stage procedures, reclustering, and bootstrap training to achieve state-of-the-art performance, and

- CascadeXML not only scales to dataset with millions of labels but also improves the current state-of-the-art in terms of prediction performance by upto $\sim 6\%$ on benchmark datasets. In addition to this, it is computationally most efficient to train and clocks multi-fold improvement in inference time as compared to existing transformer-based approaches.

2 Notation & Preliminaries

Problem setup: In extreme classification, an instance (document or query) $\mathbf{x} \in \mathcal{X}$ from some input space $\mathcal{X}$ is mapped to a subset of labels $\mathsf{y} \subset [L]$ out of the L available labels identified through the integers $[L] \coloneqq \{1, \ldots, L\}$, such that $L \sim 10^6$. Usually, we identify the labels through a binary vector $\mathbf{y} \in \{0,1\}^L$, where $\mathbf{y}_j = 1 \Leftrightarrow j \in \mathsf{y}$. The instances and labels are jointly distributed according to an unknown distribution $(\mathbf{x}, \mathbf{y}) \sim \mathbb{P}$ of which we have a i.i.d. sample $\mathcal{D} = \left\{\left(\mathbf{x}^i, \mathbf{y}^i\right) : i \in [n]\right\}$ available as a training set. Even though the number of labels can be of the order of millions, the label vectors are sparse, i.e., $\mathbb{E}[\|\mathbf{y}\|_1] \ll L$. For text data, an instance is a sequence of tokens $\mathbf{x} = (t_k)_{k=1}^{l}$, from a vocabulary $t_k \in \mathcal{V}$.

As part of the model pipeline, a text vectorizer $\Phi : \mathcal{X} \mapsto \mathbb{R}^d$ projects the documents to a d-dimensional embedding space. These vectorizers can be simple bag-of-words (BOW) or TF-IDF transformations $\Phi_{\text{tf-idf}}$, or a learnable model Φ_{dnn} that encodes the documents to the embedding space using weights learnt over the dataset. The final classification of an instance $\mathbf{x}$ is achieved by computing per label scores $\langle \mathbf{w}_l, \Phi(\mathbf{x}) \rangle$ using label-wise weight vectors $\mathbf{w}_l \in \mathbb{R}^d$ which are either (i) *jointly* learnt with Φ in a deep architecture [8, 10, 17, 31, 46, 49], or (ii) *solely* learnt in shallow classifiers [2, 50].

Transformer Encoder: More recent approaches in extreme classification have achieved state-of-the-art performance using pre-trained transformers as text vectorizers [17, 46, 51]. Popular transformer encoders for NLP [11, 27, 45] consist of an embedding layer, followed by a series of stacked self-attention layers and task-specific heads that can be fine-tuned along with the encoder. For a sequence of m input words $\mathbf{x} \in \mathcal{V}^m$, each layer a in these models produces a sequence of embeddings $\left(\Phi_0^{(a)}(\mathbf{x}), \ldots, \Phi_m^{(a)}(\mathbf{x})\right)$, where $\Phi_i^{(a)}(\mathbf{x}) \in \mathbb{R}^d$ is a representation for the i'th token in the input sequence at the a'th layer, and $\Phi_{\text{CLS}}^{(a)}(\mathbf{x}) \coloneqq \Phi_0^{(a)}(\mathbf{x})$ is an embedding corresponding to a special [CLS] token that has been pre-trained to capture a representation of the entire input [11, 27].

Label Shortlisting: Computing scores $\langle \mathbf{w}_l, \Phi(\mathbf{x}) \rangle$ for each label $l \in [L]$ in an extremely large output space is computationally very demanding, especially when the feature space $\mathbb{R}^d \ni \Phi(\mathbf{x})$ is high dimensional to allow for expressive features. To alleviate this problem, the labels L are grouped together under K *meta-labels* $\mathfrak{R} = \{\mathfrak{C}_1, \ldots, \mathfrak{C}_K\}$, with $K \approx \mathcal{O}(\sqrt{L})$. Each meta-label is represented as the set of its child labels, $\mathfrak{C}_k = \{l \in [L] : l \text{ is child of } k\}$ and each target set $\mathsf{y} \subset [L]$ corresponds to a meta-target $\tilde{\mathsf{y}} = \mathfrak{R}(\mathbf{y}) = \{\mathfrak{C}_k \mid \exists i \in \mathsf{y} : i \in \mathfrak{C}_k\}$. A meta-label $\mathfrak{C}_l$ is relevant to a data point if it contains at least one relevant label.

The label vectors are so sparse that even the much coarser meta-labels still contain mostly zeros, $|\tilde{\mathsf{y}}| \ll K$. This means that it is enough to explicitly calculate the scores only for the descendants of relevant meta-labels. In practice, an algorithm will first predict scores for each meta-labels and select the top-scoring ones as a *shortlist* for further classification. The meta- and extreme-classification tasks can either be learnt sequentially [8, 10] or jointly through dynamic label-shortlisting [17, 21].

Ideally, the labels grouped under one meta-label should be semantically similar to ensure the common representations learnt for meta- and extreme- tasks are relevant to both tasks [10]. This has the added advantage that the negative labels selected by the shortlisting procedure will be the most confusing ones, leading to hard negative sampling and improved learning of the extreme classifier [37].

For extremely large label spaces, a single cluster size K is not optimal [51]. It is either large enough that the meta-task itself becomes very expensive, or its individual clusters are so large that the shortlist

contains too many candidate labels [51]. Thus, methods like AttentionXML and XR-Transformer use a multi-level shortlisting procedure based on a *Hierarchical Label Tree (HLT)*.

Label Trees: An HLT $\mathfrak{H} = \left\{\mathfrak{R}^{(1)}, \ldots, \mathfrak{R}^{(T)}\right\}$ is a hierarchical clustering of labels into T levels of successively refined clusters $\mathfrak{R}^{(t)} = \left\{\mathfrak{C}_1^{(t)}, \ldots, \mathfrak{C}_{K_t}^{(t)}\right\}$. The clusters $\mathfrak{C}_k^{(t)}$ of each level form a disjoint partitioning of the clusters of the next, $\mathfrak{C}_k^{(t)} \subset \mathfrak{R}^{(t+1)}$, with the convention that the $T+1^{\text{th}}$ cluster is the full label space $\mathfrak{R}^{(T+1)} = [L]$. As such, each cluster $\mathfrak{C}_k^{(T)}$ (or meta-label) at level T is represented by a set of labels, each cluster at level $T-1$ is a set of level-T meta-labels, and thus a set of sets of labels, and so on. In practice, label representations $\mathbf{z}_l$ as an aggregate of their positive instance features are leveraged to create an HLT. Typically, $\mathbf{z}_l$ is constructed using $\Phi_{\text{tf-idf}}$ as

$$\mathbf{z}_l \coloneqq \frac{\mathbf{v}_l}{\|\mathbf{v}_l\|}, \quad \text{where } \mathbf{v}_l \coloneqq \sum_{i:\mathbf{y}_l^{(i)}=1} \Phi_{\text{tf-idf}}(\mathbf{x}^{(i)}).$$

The HLT is often constructed by recursively partitioning the label-space using balanced k-means clustering in a top-down fashion [8, 17, 49, 51]. For example, an HLT constructed using balanced 2-means clustering has $\lfloor \log_2 L \rfloor$ levels, out of which we select T levels corresponding to different meta-label resolutions.

3 Method: CascadeXML

At its core, the CascadeXML model consists of three components: A pre-trained language model Φ which allows to extract a document representation $\Phi_{\text{CLS}}^{(a)}(\mathbf{x}) \in \mathbb{R}^d$ at different layers $a \in [A]$, an HLT $\mathfrak{H} = \left\{\mathfrak{R}^{(1)}, \ldots, \mathfrak{R}^{(T)}\right\}$ that divides the label space into $K_1 < \ldots < K_T < L = K_{T+1}$ successively refined clusters, and a set of linear classifiers $\left\{\mathbf{W}^{(t)} = [\mathbf{w}_1^{(t)}, \ldots, \mathbf{w}_{K_t}^{(t)}] \in \mathbb{R}^{d \times K_t} : t \in [T+1]\right\}$. We select one layer for each meta-classifier to extract resolution-specific feature embeddings $\boldsymbol{\vartheta}^{(t)} \coloneqq \Phi_{\text{CLS}}^{(a_t)}(\mathbf{x})$ with $1 < a_1 < \ldots < a_T < A = a_{T+1}$.

Refining and Coarsening Label Vectors: In order to transfer the representation of a shortlist of a label vector between different layers of the HTL, we need to define two operations. The first is refining the resolution of a shortlist: Given a shortlist $\mathcal{S} = \{\mathfrak{C}_{k_1}^{(t)}, \ldots, \mathfrak{C}_{k_s}^{(t)}\} \subset \mathfrak{R}^{(t)}$ of clusters of the resolution at level t, we want to calculate the representation of this shortlist in the next layer, denoted $\mathfrak{R}_{\downarrow}^{(t)}(\mathcal{S})$. Since each cluster is defined as the set of its descendants, this is achieved by taking the union over the clusters in the list. The second operation is to find the cover of meta-labels that cover a given set of labels, $\mathfrak{R}_{\uparrow}^{(t)}(\mathcal{S})$. This is achieved by identifying each of the clusters in the layer above for which there exists at least one descendant in the shortlist. This leads to

$$\mathfrak{R}_{\downarrow}^{(t)}(\mathcal{S}) = \bigcup_{C \in \mathcal{S}} C \subset \mathfrak{R}^{(t+1)}, \qquad \mathfrak{R}_{\uparrow}^{(t)}(\mathcal{S}) = \left\{\mathfrak{C}_k^{(t-1)} \mid \exists C \in \mathcal{S} : C \in \mathfrak{C}_k^{(t-1)}\right\}. \tag{1}$$

We define $\mathfrak{R}^{(t)}(\mathbf{y}) = \left(\mathfrak{R}_{\uparrow}^{(t+1)} \circ \ldots \circ \mathfrak{R}_{\uparrow}^{(T)}\right)(\mathbf{y})$ to be the level-t cover of the label vector $\mathbf{y} \in [L]$.

Multi-resolution Dynamic Label Shortlisting: Concisely, we adapt dynamic label shortlisting [17] for multi-resolution training approaches. Formally, given a shortlist of s clusters $\{\mathfrak{C}_{k_1}^{(t)}, \ldots, \mathfrak{C}_{k_s}^{(t)}\} =: \mathcal{S}^{(t)} \subset \mathfrak{R}^{(t)}$ at level t, and the corresponding classification features $\boldsymbol{\vartheta}^{(t)}$, a new shortlist $\mathcal{S}^{(t+1)} \subset \mathfrak{R}^{(t+1)}$ is generated as follows: First, the k highest-scoring meta-labels are selected into $\mathcal{S}_{\text{top}}^{(t)}$ and the next shortlist $\mathcal{S}^{(t+1)}$ is then given as the refinement:

$$\mathcal{S}_{\text{top}}^{(t)} = \left\{\mathfrak{C}_{k_j}^{(t)} : j \in \text{Top}_k\left(\left\{\langle \mathbf{w}_{k_i}^{(t)}, \boldsymbol{\vartheta}^{(t)} \rangle\right\}\right)\right\}, \quad \mathcal{S}^{(t+1)} = \mathfrak{R}_{\downarrow}^{(t)}(\mathcal{S}_{\text{top}}^{(t)}) \tag{2}$$

At the first level, however, the shortlist is initialized to consider all meta labels $\mathcal{S}^{(1)} = \mathfrak{R}^{(1)}$. It is of essence that all true positive (meta-)labels appear at every level during training time [8, 17, 49] and is difficult to ensure intrinsically due to imperfect recall rate of classifier at resolution t. Hence, during

Algorithm 1: Training step in CASCADEXML

Input: instance $\mathbf{x}$, labels $\mathbf{y}$, clusters $\mathfrak{C}$, features Φ, classifiers $\mathbf{W}$, weights $\boldsymbol{\alpha}$

$\boldsymbol{\vartheta}^{(t)} \leftarrow \Phi^{(t)}_{\text{CLS}}(\mathbf{x})\ \forall t \in [T+1]$ `/* Forward pass through Transformer */`

loss $\leftarrow 0$

$\mathcal{S}^{(1)} \leftarrow \mathfrak{C}^{(1)}$ `/* Initialize shortlist */`

for t *in* $1, 2, ..., T$ **do**

- $\mathbf{y}^{(t)} \leftarrow \mathfrak{R}^{(t)}(\mathbf{y})$ `/* Ground-truth meta-labels */`
- $p_i \leftarrow \langle \mathbf{w}^{(t)}_{k_i}, \boldsymbol{\vartheta}^{(t)} \rangle\ \forall i \in \mathcal{S}^{(t)}$ `/* Sparse prediction */`
- loss $\leftarrow$ loss $+ \frac{\alpha_t}{|\mathcal{S}^{(t)}|} \sum_{l \in \mathcal{S}^{(t)}} \mathcal{L}^{(t)}_{\text{BC}}\left(p_l, y^{(t)}_l\right)$
- $\mathcal{S}^{(t)}_{\text{top}} \leftarrow \left\{ \mathfrak{C}^{(t)}_{k_j} : j \in \text{Top}_k\left(\{p_i \mid i \in \mathcal{S}^{(t)}\}\right) \right\}.$ `/* Top-K selection */`
- $\mathcal{S}^{(t+1)} \leftarrow \mathfrak{R}^{(t)}_{\downarrow}(\mathcal{S}^{(t)}_{\text{top}}) \cup \mathfrak{R}^{(t+1)}(\mathbf{y})$ `/* Add ground-truth positives */`

$p_i \leftarrow \langle \mathbf{w}^{(T+1)}_{k_i}, \boldsymbol{\vartheta}^{(T+1)} \rangle \quad \forall i \in \mathcal{S}^{(T+1)}$ `/* Final prediction */`

loss $\leftarrow$ loss $+ \frac{\alpha_{T+1}}{|\mathcal{S}^{(T+1)}|} \sum_{l \in \mathcal{S}} \mathcal{L}^{(T+1)}_{\text{BC}}(p_l, y_l)$

adjust Φ and $\mathbf{W}$ to reduce loss

training all meta-labels corresponding to ground-truth labels are added to the candidate set, leading to

$$\mathcal{S}^{(t+1)}_{\text{train}} = \mathfrak{R}^{(t)}_{\downarrow}(\mathcal{S}^{(t)}_{\text{top}}) \cup \mathfrak{R}^{(t+1)}(\mathbf{y})\,. \tag{3}$$

Consequently, a single training step in CascadeXML looks as given in algorithm 1.

Training Objective: At each level of the tree, the goal is to correctly identify the most likely meta-labels, which can be achieved through minimization of a one-vs-all loss

$$\mathcal{L}^{(t)}(\mathbf{x}, \mathbf{y}) = \frac{1}{|\mathcal{S}^{(t)}|} \sum_{l \in \mathcal{S}^{(t)}} \mathcal{L}^{(t)}_{\text{BC}}\left(\langle \mathbf{w}^{(t)}_{k_l}, \boldsymbol{\vartheta}^{(t)}(\mathbf{x}) \rangle, \mathfrak{R}^{(t)}(\mathbf{y})_l\right). \tag{4}$$

The overall objective is a weighted sum of the individual layer losses

$$\mathcal{L}(\mathbf{x}, \mathbf{y}) = \sum_{t=1}^{T+1} \alpha^{(t)} \mathcal{L}^{(t)}(\mathbf{x}, \mathbf{y}), \tag{5}$$

where level $T+1$ corresponds to the extreme classification task. To balance out different loss scales as a result of normalization by varied shortlist sizes, we re-scale losses from multiple resolutions using a simple re-scaling factor $\alpha^{(t)} = |\mathcal{S}^{(t)}| / \min_{t \in [T+1]}(|\mathcal{S}^{(t)}|)$.

Inference: The algorithm for inference remains the same as training, except we do not teacher-force true (meta-)labels $\mathfrak{R}^{(t)}(\text{y})$ during inference time (Eqn. 3). The inference consists of one forward pass through the transformer backbone, followed by successive shortlisting at all T resolutions, and final extreme classification. For a b-way branching tree with $\log_b L$ levels, each level t has b^t meta-labels, of which $|\mathcal{S}^{(t)}| \propto \sqrt{b^t}$ are selected for the shortlist. The time complexity is

$$\mathcal{O}(\mathcal{T}_{\text{dnn}} + (|\mathcal{S}^{(1)}| + |\mathcal{S}^{(2)}| + ... + |\mathcal{S}^{(T+1)}|)|d) = \mathcal{O}(\mathcal{T}_{\text{dnn}} + (\sum_{t=1}^{\log_b L} \sqrt{b^t})d) = \mathcal{O}(\mathcal{T}_{\text{dnn}} + d\sqrt{L})$$

where $\mathcal{T}_{\text{dnn}}$ denotes the time taken to compute transformer embeddings for an input.

4 Experiments

Benchmark datasets & Evaluation Metrics We benchmark the performance of CascadeXML on standard datasets and evaluation metrics. In all, we use 5 public XMC datasets [4, 29], the statistics of which are specified in appendix A.1. However, due to space considerations and as our main focus is on extremely large output spaces, we present results for the largest of the publicly available datasets

– Wiki-500K, Amazon-670K, and Amazon-3M, in the main paper. For all our experiments we use the same raw text input, train-test split and sparse feature representations as used in [49, 17, 51]. We use Precision@k (P@k) and Propensity-scored P@k (PSP@k) as the evaluation metrics. PSP@k focuses more on the model's performance on tail labels and the results are given in appendix A.3.3.

Baseline Methods: We compare CascadeXML with strong baselines: AnnexML [40], DiSMEC [2], Parabel [34], eXtremeText [42], Bonsai [20], XMLCNN [25], XR-Linear [50], AttentionXML, X-Transformer [8], LightXML and XR-Transformer. Results on Wiki-500K, Amazon-670K and Amazon-3M are shown in Table 2. The baseline results for other methods were obtained from [51, Table: 2]. It may be noted that, in order to incorporate the global statistical information of the dataset, XR-Transformer concatenates sparse *tf-idf* features to those learnt via deep networks (denoted Φ_{dnn})

$$\Phi_{\text{cat}}(\mathbf{x}) = \left[\frac{\Phi_{\text{dnn}}(\mathbf{x})}{\|\Phi_{\text{dnn}}(\mathbf{x})\|}, \frac{\Phi_{\text{tf-idf}}(\mathbf{x})}{\|\Phi_{\text{tf-idf}}(\mathbf{x})\|} \right]$$

For fair evaluation, we present separate comparison with methods which use (i) features from a DNN encoder only, and (ii) those obtained by combining DNN and tf-idf features.[1]

Metrics	P@1	P@3	P@5	P@1	P@3	P@5	P@1	P@3	P@5
Datasets	**Wiki-500K**			**Amazon-670K**			**Amazon-3M**		
AnnexML	64.22	43.15	32.79	42.09	36.61	32.75	49.30	45.55	43.11
DiSMEC	70.21	50.57	39.68	44.78	39.72	36.17	47.34	44.96	42.80
Parabel*	68.70	49.57	38.64	44.91	39.77	35.98	47.42	44.66	42.55
eXtremeText	65.17	46.32	36.15	42.54	37.93	34.63	42.20	39.28	37.24
Bonsai*	69.26	49.80	38.83	45.58	40.39	36.60	48.45	45.65	43.49
XR-Linear	65.59	46.72	36.46	43.38	38.40	34.77	47.40	44.15	41.87
	DNN Features								
XML-CNN	-	-	-	33.41	30.00	27.42	-	-	-
AttentionXML*	76.74	58.18	45.95	47.68	42.70	38.99	50.86	48.00	45.82
LightXML*	77.89	58.98	45.71	49.32	44.17	40.25	-	-	-
CascadeXML*	**78.39**	**59.86**	**46.49**	**50.22**	**45.20**	**41.45**	**53.10**	**50.64**	**48.49**
	[DNN $\bigoplus$ tf-idf] Features								
X-Transformer**	77.09	57.51	45.28	48.07	42.96	39.12	51.20	47.81	45.07
XR-Transformer*	79.40	59.02	46.25	50.11	44.56	40.64	**54.20**	50.81	48.26
CascadeXML*	**81.13**	**62.43**	**49.12**	**52.15**	**46.54**	**42.44**	54.11	**51.24**	**49.02**

Table 2: Comparison of CascadeXML to state-of-the-art methods on benchmark datasets. The best results for the respective feature settings are in **bold**. * implies ensemble result of 3 models, ** implies ensemble result of 9 models and '-' implies that the model does not scale for that dataset due to GPU memory constraints. Details about model ensembles can be found in the appendix.

Empirical Performance: As demonstrated in Table 2, CascadeXML, which leverages the multi-layered transformer architecture to learn label resolution specific feature representations, is empirically superior to all existing XMC approaches whether using: (i) tf-idf features, or (ii) DNN-based dense features (iii) or concatenated tf-idf and dense features. This is despite CascadeXML being a simple end-to-end trainable algorithm without using any bells or whistles like multi-stage procedures [49, 51], reclustering [51] or bootstrapped training [49, 51]. For example, on Amazon-3M dataset, CascadeXML significantly outperforms all DNN-based approaches while only taking 24 hours to train on a single GPU. In contrast, X-Transformer needs to ensemble 9 models and 23 days to train on 8 GPUs to reach the empirical performance (c.f. page 2, [51]), and LightXML remains unscalable.

On the concatenated feature space, CascadeXML also significantly outperforms XR-Transformer on 8 out of 9 dataset-metric combination with notable $\sim 6\%$ improvement on Wiki-500K dataset

[1]Details about leveraging tf-idf features along with DNN features are provided in Appendix A.4

for P@5 metric. As XR-Tranformer learns transformer features for label resolutions only upto 32K (2^{15}) clusters [51, Table 6], incorporating tf-idf features via a computation-intensive XR-Linear [50] for final extreme classification task becomes an integral part of their pipeline. On the other hand, CascadeXML does not have any such inherent limitation, and even without tf-idf features, it outperforms XR-Transformer on 6 out of 9 dataset-metric combinations.

Inference and Training Time: The comparison on inference times is shown in Table 3. CascadeXML clocks the fastest inference speed as compared to all previous DNN-based XMC models. Notably, it is $\sim 1.5\times$ and $\sim 2\times$ faster as compared to LightXML and XR-Transformer respectively. In terms of training time, CascadeXML is upto an order of magnitude faster than LightXML on a single GPU. When using multiple GPUs, its wall clock time on 4 GPUs is lower than both XR-Transformer and AttentionXML on 8 GPUs. Exact training times can be found in Section A.3.2 in the appendix.

Dataset	AttentionXML	X-Transformer	XR-Transformer	LightXML	CascadeXML
Wiki10-31K	20.0	48.1	39.1	27.1	**12.8**
AmazonCat-13K	14.4	47.6	26.1	24.1	**13.1**
Wiki-500K	80.1*	48.1	33.9	27.3	**16.0**
Amazon-670K	76.0*	48.0	30.9	23.3	**16.6**
Amazon-3M	130.5*	50.2	35.2	-	**16.9**

Table 3: Comparison of CascadeXML w.r.t. inference time (in milliseconds per sample) with SOTA XMC methods. Inference times were recorded on a single Nvidia V100 GPU and a single CPU with a batch size of 1. Superscript * implies that model parallel was used with 8 GPUs for inference.

Impact of label clusters size: Earlier works have argued that using fine-grained clusters leads to better model performance [17, 21, 31]. With increased number of label clusters at the penultimate level of HLT, the multi-resolution tasks tend to get more in-sync with each other [21] and hence enables transfer learning across different stages of learning pipelines [10, 31]. Our mutli-resolution training enables CascadeXML to use 2^{16} label clusters for penultimate label resolution, double that of XR-Transformer. As Table 4 shows, increasing the resolution at the last shortlisting step still results in non-trivial improvements, highlighting the need for architectures that can efficiently handle high-resolution meta-labels.

Dataset	Model Setting	P@1	P@3	P@5
Amazon-670K	Default	**48.5**	**43.7**	**40.0**
	2^{15} Clusters	47.6	42.9	39.3
	w/o Rescaling	47.8	42.8	39.2
Wiki-500K	Default	**76.9**	**58.4**	**45.2**
	2^{15} Clusters	76.6	58.1	44.9
	w/o Rescaling	76.5	57.9	44.7
Amazon-3M	Default	**51.3**	**49.0**	**46.9**
	2^{15} Clusters	50.8	48.7	46.5
	w/o Rescaling	50.9	48.6	46.6

Table 4: Ablation experiments showing different performance of a *single* CascadeXML model with different model settings. Default implies 2^{16} clusters in penultimate label resolution and per-resolution loss re-scaling.

Impact of re-scaling per-resolution loss: For extreme datasets, we witness more efficient training by having a larger shortlist size for finer label resolutions (see appendix A.2.2). This results in varied loss scales across label resolutions leading to one resolution dominating the training. Comparing the "Default" and "w/o Rescaling" rows for respective datasets in Table 4, indicates that rescaling as introduced in section 3 helps CascadeXML train more effectively.

5 Discussion

As shown above, CascadeXML substantially outperforms current state-of-the-art approaches both in terms of computational efficiency, and prediction performance. It achieves this while being simpler than XR-Transformer and more scalable than LightXML. Revisiting Table 1, the extraction of intermediate [CLS] representations enables us to perform end-to-end multi-resolution training with resolution-specific attention maps from a single transformer model. From Table 2, we show

improvements on both LightXML and XR-Transformer, which are transformer-based models sharing feature representations across multiple label resolutions. We attribute this increased performance to CascadeXML's end-to-end multi-resolution training pipeline and property of adapted feature embeddings. The presented architecture, therefore, combines the strength of different previous approaches, without inheriting their limitations.

We postulated that the (meta-)classification tasks at different resolutions need to attend to different tokens in the input sequence, a property lacking in recent approaches [8, 17, 51]. This intuition is corroborated by the fact that the attention maps learned by CascadeXML differ significantly between different resolutions. We provide graphs and supporting data in the appendix.

Obtaining label resolution specific feature embeddings in a pre-trained transformer model requires to make the compromise of extracting these at earlier transformer layers, where the token representations are not yet as refined as in the final layer [43]. In contrast to the extreme classification task, which requires placing true labels exactly at the top-5 positions, the shortlisting task only requires all true meta-labels be recalled within the shortlist, which is much longer than five elements. We observe that the meta-classifiers are able to achieve high-recall (as shown in appendix) at the shortlist length even with the less refined features. CascadeXML is, therefore, successful in efficiently leveraging intermediate transformer representations for the task of label shortlisting while keeping feature representations separate for each label resolution.

The structure of the model means that earlier layers in the transformer receive gradients from multiple resolutions, whereas the final layers' back propagation is limited to gradients from the extreme resolution (cf. Figure 1). This setup ensures that a portion of the representational capabilities is exclusively available for the extreme task and hence, the final transformer layers are increasingly more suitable for classification at the extreme resolution. At the same time, since the gradients from the finer tasks are also relevant for the coarser meta-label shortlisting task, so additional downstream task does not hurt the shortlisting performance at the intermediate transformer layers.

6 Other Related Works

The focus of a majority of initial works in XMC has been on using fixed features for learning the classifier with one of the following class of methods : (i) one-vs-rest [48, 2, 47, 3], (ii) label tree based [33, 20, 42, 16], (iii) decision tree methods [35, 39, 28], and (iv) label embedding based [5, 40, 12] classifiers. Beyond the above algorithmic categorization of approaches, computational considerations on scaling the training process via negative sampling [14], and sketching [30] have also been studied. Furthermore, the statistical consequences of negative sampling [37] and missing labels [15, 36] have led to novel insights in designing unbiased loss functions.

With advances in deep learning, joint learning of features and classifier has been the focus of most recent approaches. Building on the first steps for developing convolutional neural networks for text classification [22], deep extreme classification was introduced in XML-CNN [26]. This work preceded the recent developments on employing BiLSTM and transformer encoders discussed in detail the Introduction section. There has been a recent surge in works to tackle XMC for instances with short text inputs and those in which labels are endowed with textual descriptions [10, 9, 31, 32, 38]. Lately, the XMC setting has also been extended to predict unseen labels in zero-shot learning [13, 52, 44].

7 Conclusion

In this paper, we introduced a novel paradigm for fine-tuning transformer models in XMC settings. In contrast to the existing methods, which consider transformer as a blackbox encoder, we leverage its multi-layered architecture to learn data representation corresponding to different resolutions in the HLT as well as fine-grained labels at the level of the extreme classifier. The proposed instantiation of our framework in the form of CascadeXML not only yields state-of-the-art prediction performance, but is also end-to-end trainable (without intermediate reclustering steps), simpler to implement, and fast on training and inference. Beyond further research in extreme classification towards fully harnessing the representation capabilities of transformer encoders, we believe our approach can inspire future multi-resolution architectures in other domains as well which leverage label hierarchy.

References

[1] R. Agrawal, A. Gupta, Y. Prabhu, and M. Varma. Multi-label learning with millions of labels: recommending advertiser bid phrases for web pages. *Proceedings of the 22nd international conference on World Wide Web*, 2013.

[2] R. Babbar and B. Schölkopf. Dismec: Distributed sparse machines for extreme multi-label classification. *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, 2017.

[3] R. Babbar and B. Schölkopf. Data scarcity, robustness and extreme multi-label classification. *Machine Learning*, pages 1–23, 2019.

[4] K. Bhatia, K. Dahiya, H. Jain, P. Kar, A. Mittal, Y. Prabhu, and M. Varma. The extreme classification repository: Multi-label datasets and code, 2016.

[5] K. Bhatia, H. Jain, P. Kar, M. Varma, and P. Jain. Sparse local embeddings for extreme multi-label classification. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28, 2015.

[6] W. Chang, F. X. Yu, Y. Chang, Y. Yang, and S. Kumar. Pre-training tasks for embedding-based large-scale retrieval. In *8th International Conference on Learning Representations, ICLR 2020*.

[7] W.-C. Chang, D. Jiang, H.-F. Yu, C. H. Teo, J. Zhang, K. Zhong, K. Kolluri, Q. Hu, N. Shandilya, V. Ievgrafov, et al. Extreme multi-label learning for semantic matching in product search. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 2643–2651, 2021.

[8] W.-C. Chang, H.-F. Yu, K. Zhong, Y. Yang, and I. S. Dhillon. Taming pretrained transformers for extreme multi-label text classification. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020.

[9] K. Dahiya, A. Agarwal, D. Saini, G. K, J. Jiao, A. Singh, S. Agarwal, P. Kar, and M. Varma. Siamesexml: Siamese networks meet extreme classifiers with 100m labels. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 2330–2340. PMLR, 2021.

[10] K. Dahiya, D. Saini, A. Mittal, A. Shaw, K. Dave, A. Soni, H. Jain, S. Agarwal, and M. Varma. DeepXML: A Deep Extreme Multi-Label Learning Framework Applied to Short Text Documents. In *WSDM*, Mar. 2021.

[11] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, 2019.

[12] C. Guo, A. Mousavi, X. Wu, D. N. Holtmann-Rice, S. Kale, S. Reddi, and S. Kumar. Breaking the Glass Ceiling for Embedding-Based Classifiers for Large Output Spaces. In *NeurIPS*, 2019.

[13] N. Gupta, S. Bohra, Y. Prabhu, S. Purohit, and M. Varma. Generalized zero-shot extreme multi-label learning. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, August 2021.

[14] H. Jain, V. Balasubramanian, B. Chunduri, and M. Varma. Slice: Scalable Linear Extreme Classifiers trained on 100 Million Labels for Related Searches. In *WSDM*, 2019.

[15] H. Jain, Y. Prabhu, and M. Varma. Extreme multi-label loss functions for recommendation, tagging, ranking & other missing label applications. In *KDD*, pages 935–944, 2016.

[16] K. Jasinska, K. Dembczynski, R. Busa-Fekete, K. Pfannschmidt, T. Klerx, and E. Hullermeier. Extreme F-measure Maximization using Sparse Probability Estimates. In *ICML*, June 2016.

[17] T. Jiang, D. Wang, L. Sun, H. Yang, Z. Zhao, and F. Zhuang. Lightxml: Transformer with dynamic negative sampling for high-performance extreme multi-label text classification. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence*, pages 7987–7994, 2021.

[18] T. Karras, T. Aila, S. Laine, and J. Lehtinen. Progressive growing of gans for improved quality, stability, and variation. In *ICLR*, 2018.

[19] T. Karras, S. Laine, and T. Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4401–4410, 2019.

[20] S. Khandagale, H. Xiao, and R. Babbar. Bonsai: diverse and shallow trees for extreme multi-label classification. *Machine Learning*, 109(11):2099–2119, 2020.

[21] S. Kharbanda, A. Banerjee, A. Palrecha, and R. Babbar. Embedding convolutions for short text extreme classification with millions of labels. *arXiv preprint arXiv:2109.07319*, 2021.

[22] Y. Kim. Convolutional Neural Networks for Sentence Classification. In *EMNLP*, Oct. 2014.

[23] W.-S. Lai, J.-B. Huang, N. Ahuja, and M.-H. Yang. Deep laplacian pyramid networks for fast and accurate super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 624–632, 2017.

[24] K. Lee, M.-W. Chang, and K. Toutanova. Latent retrieval for weakly supervised open domain question answering. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6086–6096, Florence, Italy, July 2019. Association for Computational Linguistics.

[25] J. Liu, W. Chang, Y. Wu, and Y. Yang. Deep Learning for Extreme Multi-label Text Classification. In *SIGIR*, 2017.

[26] J. Liu, W.-C. Chang, Y. Wu, and Y. Yang. Deep learning for extreme multi-label text classification. In *Proceedings of the 40th international ACM SIGIR conference on research and development in information retrieval*, pages 115–124, 2017.

[27] 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*, abs/1907.11692, 2019.

[28] M. Majzoubi and A. Choromanska. Ldsm: Logarithm-depth streaming multi-label decision trees. In *International Conference on Artificial Intelligence and Statistics*, pages 4247–4257. PMLR, 2020.

[29] J. McAuley and J. Leskovec. Hidden factors and hidden topics: understanding rating dimensions with review text. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 165–172, 2013.

[30] T. K. R. Medini, Q. Huang, Y. Wang, V. Mohan, and A. Shrivastava. Extreme Classification in Log Memory using Count-Min Sketch: A Case Study of Amazon Search with 50M Products. In *NeurIPS*, 2019.

[31] A. Mittal, K. Dahiya, S. Agrawal, D. Saini, S. Agarwal, P. Kar, and M. Varma. Decaf: Deep extreme classification with label features. In *WSDM*, March 2021.

[32] A. Mittal, N. Sachdeva, S. Agrawal, S. Agarwal, P. Kar, and M. Varma. Eclare: Extreme classification with label graph correlations. In *Proceedings of The ACM International World Wide Web Conference*, April 2021.

[33] Y. Prabhu, A. Kag, S. Gopinath, K. Dahiya, S. Harsola, R. Agrawal, and M. Varma. Extreme multi-label learning with label features for warm-start tagging, ranking and recommendation. In *WSDM*, 2018.

[34] Y. Prabhu, A. Kag, S. Harsola, R. Agrawal, and M. Varma. Parabel: Partitioned label trees for extreme classification with application to dynamic search advertising. In *WWW*, 2018.

[35] Y. Prabhu and M. Varma. FastXML: A Fast, Accurate and Stable Tree-classifier for eXtreme Multi-label Learning. In *KDD*, Aug. 2014.

[36] M. Qaraei, E. Schultheis, P. Gupta, and R. Babbar. Convex surrogates for unbiased loss functions in extreme classification with missing labels. In *Proceedings of the Web Conference 2021*, pages 3711–3720, 2021.

[37] S. J. Reddi, S. Kale, F. Yu, D. N. H. Rice, J. Chen, and S. Kumar. Stochastic Negative Mining for Learning with Large Output Spaces. *CoRR*, 2018.

[38] D. Saini, A. K. Jain, K. Dave, J. Jiao, A. Singh, R. Zhang, and M. Varma. Galaxc: Graph neural networks with labelwise attention for extreme classification. In *Proceedings of The ACM International World Wide Web Conference*, April 2021.

[39] W. Siblini, P. Kuntz, and F. Meyer. Craftml, an efficient clustering-based random forest for extreme multi-label learning. In *International Conference on Machine Learning*, pages 4664–4673. PMLR, 2018.

[40] Y. Tagami. AnnexML: Approximate Nearest Neighbor Search for Extreme Multi-label Classification. In *KDD*, Aug. 2017.

[41] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[42] M. Wydmuch, K. Jasinska, M. Kuznetsov, R. Busa-Fekete, and K. Dembczynski. A no-regret generalization of hierarchical softmax to extreme multi-label classification. In *NIPS*, 2018.

[43] J. Xin, R. Tang, J. Lee, Y. Yu, and J. Lin. DeeBERT: Dynamic early exiting for accelerating BERT inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2246–2251, Online, July 2020. Association for Computational Linguistics.

[44] Y. Xiong, W.-C. Chang, C.-J. Hsieh, H.-F. Yu, and I. S. Dhillon. Extreme zero shot learning for extreme text classification. In *NAACL*, 2022.

[45] Z. Yang, Z. Dai, Y. Yang, J. G. Carbonell, R. Salakhutdinov, and Q. V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *NeurIPS*, 2019.

[46] H. Ye, Z. Chen, D.-H. Wang, and D. B. D. Pretrained Generalized Autoregressive Model with Adaptive Probabilistic Label Clusters for Extreme Multi-label Text Classification. In *ICML*, 2020.

[47] E. I. Yen, X. Huang, W. Dai, I. Ravikumar, P.and Dhillon, and E. Xing. PPDSparse: A Parallel Primal-Dual Sparse Method for Extreme Classification. In *KDD*, Aug. 2017.

[48] E. I. Yen, X. Huang, K. Zhong, P. Ravikumar, and I. S. Dhillon. PD-Sparse: A Primal and Dual Sparse Approach to Extreme Multiclass and Multilabel Classification. In *ICML*, June 2016.

[49] R. You, Z. Zhang, Z. Wang, S. Dai, H. Mamitsuka, and S. Zhu. AttentionXML: Label tree-based attention-aware deep model for high-performance extreme multi-label text classification. In *NeurIPS*, 2019.

[50] H.-F. Yu, K. Zhong, J. Zhang, W.-C. Chang, and I. S. Dhillon. Pecos: Prediction for enormous and correlated output spaces. *Journal of Machine Learning Research*, 2022.

[51] J. Zhang, W.-c. Chang, H.-f. Yu, and I. Dhillon. Fast multi-resolution transformer fine-tuning for extreme multi-label text classification. *Advances in Neural Information Processing Systems*, 34, 2021.

[52] Y. Zhang, Z. Shen, C.-H. Wu, B. Xie, J. Hao, Y.-Y. Wang, K. Wang, and J. Han. Metadata-induced contrastive learning for zero-shot multi-label text classification. In *Proceedings of the ACM Web Conference 2022*, pages 3162–3173, 2022.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See Section **??**.
   (b) Did you describe the limitations of your work? [Yes] In the appendix
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] In the appendix
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Code is in the supplementary material, the data is publically available.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] In the appendix A.2.2
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] It is not a standard practice in this domain.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] In the appendix A.2
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] In Section 4: Benchmark datasets
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]