# DeepXML: A Deep Extreme Multi-Label Learning Framework Applied to Short Text Documents

Kunal Dahiya
kunalsdahiya@gmail.com
IIT Delhi
India

Deepak Saini
desaini@microsoft.com
Microsoft Research
India

Anshul Mittal
Ankush Shaw
me@anshulmittal.org
shawank17198@gmail.com
IIT Delhi
India

Kushal Dave
Akshay Soni
kudave@microsoft.com
Akshay.Soni@microsoft.com
Microsoft
USA

Himanshu Jain
Sumeet Agarwal
himanshu.j689@gmail.com
sumeet@ee.iitd.ac.in
IIT Delhi
India

Manik Varma
manik@microsoft.com
Microsoft Research
IIT Delhi
India

## ABSTRACT

Scalability and accuracy are well recognized challenges in deep extreme multi-label learning where the objective is to train architectures for automatically annotating a data point with the most relevant subset of labels from an extremely large label set. This paper develops the DeepXML framework that addresses these challenges by decomposing the deep extreme multi-label task into four simpler sub-tasks each of which can be trained accurately and efficiently. Choosing different components for the four sub-tasks allows DeepXML to generate a family of algorithms with varying trade-offs between accuracy and scalability. In particular, DeepXML yields the Astec algorithm that could be 2-12% more accurate and 5-30× faster to train than leading deep extreme classifiers on publically available short text datasets. Astec could also efficiently train on Bing short text datasets containing up to 62 million labels while making predictions for billions of users and data points per day on commodity hardware. This allowed Astec to be deployed on the Bing search engine for a number of short text applications ranging from matching user queries to advertiser bid phrases to showing personalized ads where it yielded significant gains in click-through-rates, coverage, revenue and other online metrics over state-of-the-art techniques currently in production. DeepXML's code is available at https://github.com/Extreme-classification/deepxml.

## CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; *Supervised learning by classification.*

## KEYWORDS

Extreme multi-label learning, large-scale learning, short-text, bid-phrase recommendation, personalized ads

**ACM Reference Format:**
Kunal Dahiya, Deepak Saini, Anshul Mittal, Ankush Shaw, Kushal Dave, Akshay Soni, Himanshu Jain, Sumeet Agarwal, and Manik Varma. 2021. DeepXML: A Deep Extreme Multi-Label Learning Framework Applied to Short Text Documents. In *Proceedings of the Fourteenth ACM International Conference on Web Search and Data Mining (WSDM '21), March 8–12, 2021, Virtual Event, Israel.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3437963.3441810


*WSDM '21, March 8–12, 2021, Virtual Event, Israel*

ACM ISBN 978-1-4503-8297-7/21/03…$15.00
https://doi.org/10.1145/3437963.3441810

## 1 INTRODUCTION

**Objective**: This paper develops the DeepXML framework for deep extreme multi-label learning where the goal is to train architectures for automatically annotating a data point with the most relevant *subset* of labels from an extremely large label set. Note that multi-label learning generalizes multi-class classification which aims to predict a single mutually exclusive label. The objectives for developing DeepXML are threefold. First, DeepXML provides a framework for how to think about deep extreme multi-label learning that can not only be used to analyze seemingly disparate algorithms such as XML-CNN [28] and MACH [32] but which can also be used to derive significantly improved versions of such state-of-the-art deep extreme classifiers. Second, DeepXML can generate a family of new state-of-the-art algorithms obtained by combining various types of feature architectures with different classifiers in a scalable and accurate manner. In particular, DeepXML was used to derive Astec (standing for an Accelerated Short Text Extreme Classifier) which is specialized for extremely low-latency and high throughput short text applications as it can make billions of predictions per day and handle peak rates of up to a hundred and twenty thousand queries per second. Third, DeepXML provides an easy-to-use modular framework in which practitioners can design architectures for diverse applications by making minimal changes and simply plugging in the components of their choice rather than going back

to the drawing board and designing a specialized architecture for each application from scratch.

**Short text applications**: This paper focuses on the classification of short text documents, having just 3-10 words on average, into millions of labels. In particular, this paper considers a range of short text applications including matching user search engine queries to advertiser bid phrases, predicting Wikipedia tags from a document's title, predicting frequently bought together products from a given retail product's name and showing personalized ads based on the set of webpage titles in a user's browsing history. Such applications pose a number of additional challenges to deep extreme classifiers as compared to long text documents having up to 200 words on average. First, the deep extreme classifier is forced to make predictions on the basis of just 3-10 words on average which is a significantly harder task than long text document classification. Second, short text corpora have fewer occurrences of each word than long text corpora thereby leading to a paucity of training data. For instance, the Bing datasets have millions of words which occur at most twice in the training set and thus learning good quality embeddings for such rare words can be challenging. Third, low-latency and high-throughput short text applications with billions of users require predictions in milliseconds on a CPU to keep operating and energy costs low. While these challenges seem daunting at the extreme scale, it is nevertheless important to design solutions for short text applications as they can benefit billions of people.

**Challenges in deep extreme classification**: Deep extreme classifiers jointly learn a feature architecture with an extremely large classification layer leading to the following challenges. First, training and fine-tuning the feature architecture for millions of labels can be computationally expensive and can also lead to learning poor quality representations when training data is scarce. Second, both the forward prediction pass as well as gradient back-propagation become infeasible if the classification layer has costs that are linear in the number of labels (such as for a fully connected output layer). One might be tempted to address these challenges by replacing the classification layer with highly scalable non-deep learning based extreme classifiers [19, 41] which reduce the costs of the forward and backward pass to logarithmic in the number of labels. This is achieved by learning a sub-linear search data structure based on graphs [19], trees [20, 24, 40–42, 50], hashes [46, 47] or clustering [7] in a fixed feature space such as a bag-of-words representation or fixed pre-trained embeddings. Unfortunately, such algorithms cannot be directly used to replace the classification layer in the deep learning setting as the feature representation changes with every mini-batch update. This necessitates the frequent recomputation or updation of the sub-linear search data structure on the updated features which can be prohibitively expensive. As such, deep learning presents additional statistical and computational challenges to extreme classification.

**DeepXML**: DeepXML addresses these challenges by decomposing the deep extreme learning task into the following four sub-tasks or modules each of which can be constrained to be learnt in log-time while maintaining accuracy. In Module I, an intermediate feature representation is learnt using an application-appropriate feature architecture trained on a simpler surrogate task. The objective in Module I is to efficiently learn a near-final feature representation which can be fixed and used to eliminate all but a logarithmic number of the hardest negative labels for each data point. Then, in Module II, a graph, tree, hash or clustering based sub-linear search data structure is trained just once on the fixed intermediate features to shortlist the hardest negative labels for each data point in log-time. The motivation is to reduce the problem for any given data point from an extreme task with millions of labels to a traditional classification task with just hundreds of labels. Note that this can be achieved with minimal loss in accuracy as uninformative negative labels can be discarded since they don't contribute to the final solution [19]. Module III then transfers the intermediate features to learn final features for the given extreme task subject to the constraint that the final features don't lie very far away from the intermediate features. This is done so as to get all the accuracy gains of fine-tuning for the task at hand while ensuring that the hardest negatives continue to lie in the shortlisted label set. Module IV jointly learns an extreme classifier along with the final features in log-time using just the shortlisted labels. Varying the choices for the component including the feature architecture, the surrogate task, the sub-linear search data structure, the negative sampling procedure, the transfer mechanism and the extreme classifier leads to a family of state-of-the-art algorithms including Astec, DECAF [36], GalaXC [45], ECLARE [37], *etc.*

**Astec**: Astec was derived from DeepXML specifically for short text applications. It employed a low capacity feature architecture which could be learnt accurately from limited training data. This allowed Astec to be 2-12% more accurate than leading deep extreme classifiers on publicly available benchmark datasets while also being up to 20% more accurate than state-of-the-art techniques for matching user queries to advertiser bid phrases on Bing datasets. Furthermore, by leveraging the DeepXML framework, Astec could be 5-30× faster to train than leading deep extreme classifiers and could efficiently scale to problems involving 62 million labels. Finally, Astec could make predictions in milliseconds on a CPU and could therefore make billions of predictions per day, with peak rates of 120,000 queries per second, using commodity hardware. As a result, Astec increased the click-through-rate by 6.5% over state-of-the-art techniques in production for showing personalized ads based on the webpage titles and URLs in a user's browsing history. Similarly, Astec yielded an increase of 1.6% in revenue per thousand queries, a 2.9% increase in match quality and an 8.6% increase in query coverage over leading techniques in production for matching user queries to advertiser bid phrases. These represent just two of the multiple short text applications for which Astec resulted in a significant increase in key metrics on Bing.

## 2 RELATED WORK

**DeepXML and extreme Classification**: Much progress has been made in extreme classification [1–3, 7, 19–21, 24, 35, 40–42, 47, 50, 53, 54] for fixed representations such as bag-of-words features and pre-trained embeddings. Unfortunately, these algorithms cannot be used directly for deep extreme classification as the sub-linear search data structure they rely on for scalability needs to be frequently recomputed due to the change in features with every mini-batch update. As a result, specialized deep extreme classifiers [10, 22,

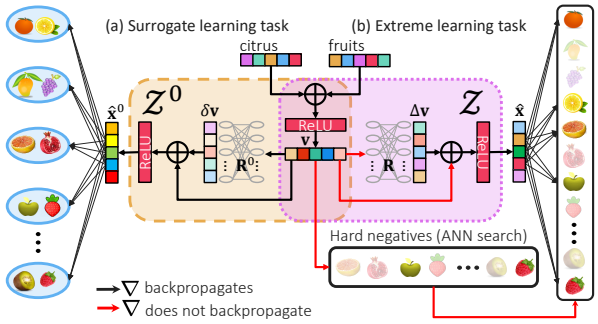


**Figure 1: Astec's architecture: Please refer to the text for details. Best viewed under magnification and in color.**

28, 32, 51, 56, 57] have been developed and many of these can be analysed and improved in the DeepXML framework.

**Astec and short text extreme classification**: Of particular relevance to Astec and this paper are specialized extreme classifiers that have been developed for short text applications including Slice [19] for recommending related queries on Bing and MACH [32] for searching Amazon retail products. Slice is a highly scalable classifier for fixed pre-trained embeddings and, unfortunately, cannot be used for deep extreme classification as already mentioned. Astec could therefore be up to 20% more accurate than Slice on pre-trained CDSSM [17], FastText [23] or BERT [13] embeddings (see Section 5). Also note that Slice could always be incorporated into DeepXML's second module if desired. Furthermore, Astec can be seen as a generalization of MACH when analyzed through the DeepXML framework. In particular, MACH stops training after the first DeepXML module and therefore has to learn a large ensemble of base classifiers to compensate. Astec could be up to 12% more accurate and orders of magnitude faster to train as it learnt a single base classifier by efficiently and accurately leveraging the DeepXML framework with all four modules.

**Short text applications**: Apart from extreme classification, four classes of techniques have been developed for matching user queries to advertiser bid phrases and the other short text applications considered in this paper. The first class of techniques leverage additional sources of information such as landing pages [52], web search results and other queries [9] and are therefore beyond the scope of this paper. The second class of techniques are based on generative models [14, 26, 27, 58] that synthesize bid phrases for a given query. Unfortunately, unconstrained synthesis can result in poor quality bid phrases being generated while constraining them through tries [27] or other approaches can limit bid phrase coverage. The third class of techniques use graph neural networks, random walks and other graph processing methods on the query-bid phrase, query-url, query-token or query-query graphs [18, 33]. Features based on such graphs can be readily incorporated into the DeepXML framework by leveraging graph neural networks in the first and third modules [37, 45]. The fourth class of techniques embed both queries and bid phrases into the same space using a Siamese network [4, 12, 17, 44] or two-tower model [25, 55], and make predictions for a novel query by retrieving the nearest embedded bid phrases with highest cosine similarity or other metrics. Section 5 demonstrates that Astec could yield significant improvements in online metrics for multiple short text applications over large ensembles of state-of-the-art methods for each class of techniques, that are currently in production in Bing.

## 3 THE DEEPXML FRAMEWORK

**Notation**: Let $L$ be the number of labels and $V$ be the vocabulary size if the input is a text document. Each of the $N$ training points is then represented as $(\mathbf{x}_i, \mathbf{y}_i)$, where $\mathbf{x}_i$ is a data-point represented either as a dense vector, sequence of tokens or sparse bag-of-tokens depending on the application and $\mathbf{y}_i \in \{-1, +1\}^L$ is the ground truth label vector with $y_{il} = +1$ if label $l \in [L]$ is relevant to data point $i$ and $y_{il} = -1$ otherwise.

**Components**: DeepXML has the following components. First, a feature architecture $\mathcal{Z}$ that maps a data point $\mathbf{x}_i$ onto a dense $D$ dimensional representation $\hat{\mathbf{x}}_i$, *i.e.,* $\mathcal{Z} : \mathbf{x}_i \rightarrow \hat{\mathbf{x}}_i \in \mathbb{R}^D$. Second, a surrogate objective to train intermediate feature representations. Third, a sub-linear search structure and negative sampling procedure. Fourth, a transfer mechanism to obtain final feature representations and, finally, parameters $\mathbf{W}$ of a classifier model to make final predictions. Section 4 includes efficient choices made by the Astec algorithm for all these components.

**Summary**: Given a task-specific loss function $\ell$ that measures the accuracy of a classifier model on a specific label, an ideal training strategy would train $\mathcal{Z}, \mathbf{W}$ jointly, taking into account all positive and all negative labels of all $N$ training points, *i.e.,* solve $\arg\min_{\mathcal{Z},\mathbf{W}} \mathcal{L}(\mathcal{Z}, \mathbf{W})$

$$\mathcal{L}(\mathcal{Z}, \mathbf{W}) = \sum_{i=1}^{N} \sum_{l=1}^{L} \ell(\mathbf{x}_i, y_{il}; \mathcal{Z}, \mathbf{W})$$

However, this strategy requires jointly learning $\Omega(VD + LD)$ parameters (to describe $\mathcal{Z}$ and $\mathbf{W}$) using the objective $\mathcal{L}$ for which computing even a single gradient takes $\Omega(NLD)$ time (as the objective has $NL$ terms) and is prohibitive for even moderate scale datasets. This remains true even if the objective uses a loss that does not decompose over the labels such as the F-measure, *etc.* To remedy this, DeepXML proposes a *modular* strategy that effectively scales to tasks with millions of labels and data points. The core idea is to solve a much cheaper "surrogate" task (in **Module I**) and use this solution to identify (in **Module II**) shortlists $\hat{\mathcal{N}}_i$ of say $O(\log L)$ negative labels for each data point $i \in [N]$ (*i.e.,* $y_{il} = -1$ for all $l \in \hat{\mathcal{N}}_i$) that offer a good approximation to the objective, *i.e.,* for all $i \in [N]$, it is the case that

$$\sum_{l:y_{il}=-1} \ell(\mathbf{x}_i, y_{il}; \mathcal{Z}, \mathbf{W}) \approx \sum_{l \in \hat{\mathcal{N}}_i} \ell(\mathbf{x}_i, y_{il}; \mathcal{Z}, \mathbf{W}).$$

For example, if using a margin loss function such as the hinge loss, a shortlist of all margin violator negative labels would suffice. More generally, since these labels are expected to incur high loss values and also be most likely to be confused for a positive label, they are commonly referred to as *hard negatives* [19]. After performing a feature transfer from the surrogate to the original task (in **Module III**), final training is done (in **Module IV**) to learn $\mathcal{Z}, \mathbf{W}$ by solving

a much less expensive optimization problem with the objective $\hat{\mathcal{L}}$

$$\hat{\mathcal{L}}(\mathcal{Z}, \mathbf{W}) = \sum_{i=1}^{N} \sum_{l \in \hat{\mathcal{N}}_i \cup \mathcal{P}_i} \ell(\mathbf{x}_i, y_{il}; \mathcal{Z}, \mathbf{W}),$$

where for any $i \in [N]$, $\mathcal{P}_i := \{l : y_{il} = +1\}$ denotes the set of positive labels of that data point. Minimizing $\hat{\mathcal{L}}(\mathcal{Z}, \mathbf{W})$ is expected to yield parameters that resemble those obtained by minimizing $\mathcal{L}(\mathcal{Z}, \mathbf{W})$ due to the way hard negatives are designed. However, each module in the DeepXML framework can be executed in no more than $O(ND \log L)$ time if careful choices are made. Finally, note that DeepXML supports learning an ensemble of multiple learners by training them from scratch or else simply training a re-ranker to cut down the training costs (See section 4).

**Flexibility**: DeepXML offers the flexibility of tackling a range of disparate applications with diverse inputs ranging from documents to images to graphs by letting practitioners plug in the components of their choice with minimal effort and without having to redesign the entire architecture from scratch for each application. Various feature architectures, ranging from convolutions to transformers, as well as classifier architectures, ranging from 1-vs-All to trees, can be plugged in depending on the accuracy and scalability requirements. Furthermore, DeepXML offers the flexibility of using diverse types of deep extreme classifiers, ranging from XML-CNN to MACH, by casting them in the proposed framework. Finally, DeepXML also offers the flexibility to incorporate metadata such as label features [36] or label correlations [37] into the various modules to obtain superior performance. This allowed DeepXML to be used for a number of applications including text ads, product ads, rich ads, native ads, retail product recommendation, news recommendation, personalized query recommendation, *etc.*

### 3.1 Module I: Intermediate representation

In this module, an intermediate feature architecture $\mathcal{Z}^0$ is trained using a *surrogate* task. Several considerations need to be kept in mind while choosing a feature architecture and surrogate task. The feature architecture $\mathcal{Z}^0$ should be trainable from available data especially with respect to rare tokens and rare labels. $\mathcal{Z}^0$ should also be able to efficiently embed data points $\mathbf{x}_i \mapsto \hat{\mathbf{x}}_i^0$, ideally in time $O(cND \log L)$ where $c$ is some constant depending on the architecture, to satisfy the requirements of low-latency and high-throughput applications. Simultaneously, the surrogate task should be solvable faster than solving $\arg\min_{\mathcal{Z},\mathbf{W}} \mathcal{L}(\mathcal{Z}, \mathbf{W})$, ideally offering gradient computations in time $O(cND \log L)$. It should also promote learning of a feature architecture $\mathcal{Z}^0$ whose data point representations, say $\hat{\mathbf{x}}_i^0 = \mathcal{Z}^0(\mathbf{x}_i)$, closely resemble those offered by $\mathcal{Z}$, say $\hat{\mathbf{x}}_i = \mathcal{Z}(\mathbf{x}_i)$. Recall that $\mathcal{Z}$ is the feature architecture that could have been learnt by directly solving $\arg\min_{\mathcal{Z},\mathbf{W}} \mathcal{L}(\mathcal{Z}, \mathbf{W})$. Doing so ensures that $\hat{\mathbf{x}}_i^0$ and $\hat{\mathbf{x}}_i$ have approximately the same nearest neighbors and label shortlists generated using $\hat{\mathbf{x}}_i^0$ (in Module II) are apt proxies for those that could have been generated using $\hat{\mathbf{x}}_i$.

The feature architecture $\mathcal{Z}^0$ may be learnt in several ways. Unsupervised surrogate tasks include skip-gram models [23, 34], next sentence prediction [13], masked language modeling [13], and multi-task learning [29]. While scalable, architectures learnt using unsupervised training may lie far away from, and were empirically found to be 4-5% less accurate than, those learnt using supervised training (see Table 6 in the supplementary material). Supervised learning techniques cut down the training cost by reducing the effective number of labels to $\hat{L} \ll L$. This can be implicitly done by sampling a small sub-set of labels in an online manner, say by mini-batch negative sampling [15], or explicitly by label selection, label projection or label clustering. Label selection methods [5, 8] select a subset of labels $\mathfrak{L}$ but were found to offer sub-optimal accuracies. These methods suffer from poor token coverage: if the set of data points tagged with at least one label in $\mathfrak{L}$, *i.e.,* $\{i : y_{ij} = +1 \text{ for any } j \in \mathfrak{L}\}$ do not cover all vocabulary tokens, then either pre-trained token embeddings have to be externally sourced or else performance may suffer. Ensuring perfect token coverage is usually challenging – nearly 1M labels had to be selected in order to cover 95% of the vocabulary on the Q2B-3M dataset which defeats the very purpose of creating a surrogate task. Low-rank projection methods [7, 32, 35, 47] project labels on to a low-dimensional space as $\hat{\mathbf{y}}_i = \mathbf{P}\mathbf{y}_i$, where $\mathbf{P} \in \mathbb{R}^{\hat{L} \times L}$ is a projection matrix. These are theoretically well understood but offered accuracies similar to label selection approaches in our experiments. Label clustering approaches [24, 36, 41, 46, 56] cluster labels and treat each cluster as a *meta label* using either explicit label features (using state-of-the-art encoders [13, 23]) or else using indirect label representations [19, 41]. Label clustering-based approaches were found to offer the best performance in the experiments reported in section 5 but other approaches might be more suitable for other applications.

### 3.2 Module II: Negative sampling

In this module, $\mathcal{Z}^0$ is used to obtain intermediate representations $\hat{\mathbf{x}}_i^0 = \mathcal{Z}^0(\mathbf{x}_i)$ for all data points which are then used to obtain shortlists $\hat{\mathcal{N}}_i \subset [L]$ of the $O(\log L)$ most confusing or "hardest" negative labels, for every data point $i \in [N]$. Such deliberate hard negative mining outperforms cheaper alternatives such as mini-batch sampling [15], or sampling negatives from the power law distribution [34] (see Fig. 2 in the supplementary material). This is because the probability of choosing the most confusing negative labels is negligible when the number of labels is in the millions. Two main considerations need to be kept in mind while creating these shortlists. First, the shortlist should contain negative labels most likely to be confused for positive labels to offer concise and directed signals while training the classifiers. Second, the shortlist should be computable for every data point $i \in [N]$ in time sub-linear in $L$. Several options exist including those based on graphs [19, 31], trees [24, 41, 46], clusters [36, 56] or hashing [7, 32, 43] that achieve sub-linear time negative sampling when working with fixed features. Note that it is possible to reuse the same data structure to shortlist $O(\log L)$ labels during prediction as well.

### 3.3 Module III: Transfer learning

In this module, the final form of the feature architecture $\mathcal{Z}$ is created by adapting the intermediate architecture $\mathcal{Z}^0$. A non-trivial transfer may be required since $\mathcal{Z}^0$ is tuned for the surrogate task and not the original task. If this transfer involves any re-parametrization, any free parameters thus introduced are either trained separately or else fine-tuned jointly in Module-IV (see below). Three main considerations need to be kept in mind while performing this feature

transfer. First, $\mathcal{Z}$ should not impose significant computational overhead as compared to $\mathcal{Z}^0$. Second, additional parameters introduced in $\mathcal{Z}$ should be trainable from available training data, especially with respect to rare tokens, in time $O(ND \log L)$. Finally, feature representations offered by $\mathcal{Z}$ should not lie too far away from those offered by $\mathcal{Z}^0$ so that hard negatives discovered in Module II continue to remain relevant for classifier training. Sophisticated transfer learning techniques [39, 45, 48, 49] could be deployed in this module and in particular, a method aiming for higher accuracies could choose to fine-tune $\mathcal{Z}$ in its entirety across Modules I-IV, albeit at greater computational expense.

### 3.4 Module IV: Classifier learning

In this module, the classifier's parameters $\mathbf{W}$ (and optionally, any free parameters in $\mathcal{Z}$) are (jointly) learnt using an approximate objective that considers only the positive labels of a data point, *i.e.,* $\mathcal{P}_i := \{l : y_{il} = +1\}$ and the shortlisted negative labels $\hat{\mathcal{N}}_i$.

$$\hat{\mathcal{L}}(\mathcal{Z}, \mathbf{W}) = \sum_{i=1}^{N} \sum_{l \in \hat{\mathcal{N}}_i \cup \mathcal{P}_i} \ell(\mathbf{x}_i, y_{il}; \mathcal{Z}, \mathbf{W})$$

Various classifiers including 1-vs-All [2, 3, 19, 24, 41, 56], tree [20, 22, 42, 50] or $k$-NN [7, 47] classifiers may serve as suitable choices. Note that for most commonly used loss functions, computing gradients $\nabla\hat{\mathcal{L}}$ takes only $O(ND \log L)$ time since data points typically contain only logarithmically many positive labels, *i.e.,* $|\mathcal{P}_i| \approx O(\log L)$ and $\left|\hat{\mathcal{N}}_i\right| \leq O(\log L)$ by design. Modules I and IV can use distinct loss functions aimed at promoting recall and precision respectively.

## 4 THE ASTEC ALGORITHM

**Module I**: Astec uses the following feature architecture that can be learnt from limited training data and be computed in 30 $\mu s$ on a CPU thereby meeting the accuracy and latency requirements of short text applications. In particular, Astec operates with sparse bag-of-words representations for documents, *i.e.* $\mathbf{x}_i \in \mathbb{R}^V$, and learns $D$ dimensional embeddings for each vocabulary token $\mathbf{e}_t \in \mathbb{R}^D : t \in [V]$. The intermediate features used by Astec are of the form $\hat{\mathbf{x}}^0 := \mathcal{Z}^0(\mathbf{x}) = \mathbf{v} + \delta\mathbf{v}$ where $\mathbf{v} := \text{ReLU}\left(\sum_{t=1}^{V} x_t \cdot \mathbf{e}_t\right)$, *i.e.* a ReLU non-linearity over the TF-IDF weighted linear combination of the learnt token embeddings, and $\delta\mathbf{v} := \text{ReLU}(\mathbf{R}^0\mathbf{v})$ where $\mathbf{R}^0 \in \mathbb{R}^{D \times D}$ is a residual matrix. The final features are of the form $\hat{\mathbf{x}} := \mathcal{Z}(\mathbf{x}) = \mathbf{v} + \Delta\mathbf{v}$ where $\Delta\mathbf{v} := \text{ReLU}(\mathbf{R}\mathbf{v})$ and $\mathbf{R} \in \mathbb{R}^{D \times D}$. Note that $\mathcal{Z}^0(\mathbf{x})$ and $\mathcal{Z}(\mathbf{x})$ share the component $\mathbf{v}$ and only differ in the residual component $\delta\mathbf{v}, \Delta\mathbf{v}$. Restricting the spectral norms of $\mathbf{R}^0$ and $\mathbf{R}$ encourages a high fidelity transfer in Module III, *i.e.,* $\hat{\mathbf{x}} \approx \hat{\mathbf{x}}^0$. Thus, Astec's feature architectures $\mathcal{Z}^0, \mathcal{Z}$ are parametrized using token embeddings $\mathbf{E} = [\mathbf{e}_1, \ldots, \mathbf{e}_V] \in \mathbb{R}^{D \times V}$ and $\mathbf{R}^0, \mathbf{R} \in \mathbb{R}^{D \times D}$.

Astec adopts label clustering for its surrogate task as this was empirically observed to lead to the highest classification accuracies while keeping training time to within a few hours on a single GPU across all datasets in Table 1. Label centroids, defined as $\boldsymbol{\mu}_l^s = \frac{\hat{\boldsymbol{\mu}}_l^s}{\|\hat{\boldsymbol{\mu}}_l^s\|_2}$, were used to cluster the labels, where, $\hat{\boldsymbol{\mu}}_l^s = \frac{1}{|\mathcal{P}_l|} \sum_{i \in \mathcal{P}_l} \mathbf{x}_i$, and $\mathcal{P}_l := \{i : y_{il} = +1\}$ is the set of documents for which label $l$ is relevant. The balanced 2-means++ algorithm [41] was used to recursively cluster the labels into balanced partitions until $\hat{L}$ clusters were obtained. These clusters were treated as meta-labels and new (meta) label vectors $\hat{\mathbf{y}}_i \in \{-1, +1\}^{\hat{L}}$ were created for each training document as $\hat{y}_{ik} = +1$ for documents $i$ tagged with at least one label in cluster $k$ and $\hat{y}_{ik} = -1$ otherwise. Using label correlations for improved clustering led to a 2% improvement in recall as compared to Parabel's clusters which ignored label correlations. Predicting the relevant clusters for a given document was taken as the surrogate task for Module I. 1-vs-All classifiers, parametrized as $\hat{\mathbf{W}} = [\hat{\mathbf{w}}_1, \ldots, \hat{\mathbf{w}}_{\hat{L}}] \in \mathbb{R}^{D \times \hat{L}}$ and $\mathcal{Z}^0$ were trained jointly by solving

$$\arg\min_{\mathcal{Z}^0, \hat{\mathbf{W}}} \sum_{i=1}^{N} \sum_{k=1}^{\hat{L}} \log\left(1 + \exp\left(-\hat{y}_{ik} \cdot \hat{\mathbf{w}}_k^\top \hat{\mathbf{x}}_i^0\right)\right) \tag{1}$$

subject to the constraint $\sup_{\mathbf{u} \neq \mathbf{0}} \left\|\mathbf{R}^0\mathbf{u}\right\|_2 / \|\mathbf{u}\|_2 \leq \lambda$, and with $\hat{\mathbf{x}}_i^0 = \mathcal{Z}^0(\mathbf{x}_i)$. Note that $\hat{\mathbf{W}}, \mathbf{R}^0$ are distinct from their counterparts $\mathbf{W}, \mathbf{R}$ used to solve the original task in Module IV. Indeed, $\hat{\mathbf{W}}, \mathbf{R}^0$ were summarily discarded after completion of Module I. However, $\mathbf{E}$ was shared by $\mathcal{Z}^0$ and $\mathcal{Z}$ and hence, retained. The power iteration method [38] was used to constrain the spectral norm of $\mathbf{R}^0$. Adding dropout after each ReLU layer was found to be as effective as using an explicit regularizer $\frac{1}{2}\sum_l \hat{\mathbf{w}}_l^\top \hat{\mathbf{w}}_l$. Using $\hat{L} \leq O(\log L)$, efficient parallelization on a GPU and the Adam optimizer controlled the complexity of the surrogate task. In practice, $\mathcal{Z}^0$ could be trained in a couple of hours on a single GPU across all datasets in Table 1.

**Module II**: The sampling cost of techniques discussed in Section 3.2 can be larger than $O(ND \log L)$ if document features $\hat{\mathbf{x}}_i^0$ keep changing during training due to $\mathcal{Z}^0$ getting updated with each mini-batch. Astec tackles this challenge by freezing the intermediate representations after completion of Module I before training a sub-linear search data structure to discover hard negatives using multiple label representations. For simplicity, Astec utilizes $\mathbf{v}$ to sample hard negatives which were found to be just as effective as those sampled using $\hat{\mathbf{x}}^0$. A four-step procedure was adopted to sample hard negatives. First, multiple representations were computed for each label (refer to section A.4 in the supplementary material). Then, two Approximate Nearest Neighbour Structures (ANNS) [31] were deployed: ANNS$^x$ over $\{\mathbf{v}_i : i \in [N]\}$ and ANNS$^\mu$ over label representations defined as $\boldsymbol{\mu}_l^0 = \frac{\hat{\boldsymbol{\mu}}_l^0}{\|\hat{\boldsymbol{\mu}}_l^0\|_2}$ where $\hat{\boldsymbol{\mu}}_l^0 = \frac{1}{|\mathcal{P}_l|}\sum_{i \in \mathcal{P}_l} \mathbf{v}_i$. Recall that $\mathbf{v}_i$ is the component shared by $\hat{\mathbf{x}}_i^0$ and $\hat{\mathbf{x}}_i$. The graphs were queried to generate $O(\log L) \leq 500$ negative labels for each $i \in [N]$ as: (a) $\hat{\mathcal{N}}_i^x = \left\{l : j \in \text{ANNS}^x(\mathbf{v}_i), y_{jl} = +1, y_{il} = -1\right\}$, and (b) $\hat{\mathcal{N}}_i^\mu = \left\{l : \boldsymbol{\mu}_l^0 \in \text{ANNS}^\mu(\mathbf{v}_i), y_{il} = -1\right\}$. Adding $O(\log L) \leq 50$ random negatives to the shortlist (to account for minor distortions between $\mathbf{v}_i$ and $\hat{\mathbf{x}}_i$) increased prediction accuracy.

Astec's shortlisting strategy could recall 5% more relevant labels and the overall predictions could be 13% more accurate than Slice. Theorem 1 indicates that it was justified to generate ANNS shortlists using $\mathbf{v}_i$ rather than the final features, i.e. $\hat{\mathbf{x}}_i$. This was also empirically validated as more than 87% overlap was observed between the label shortlists computed on $\mathbf{v}_i$ and those computed on $\hat{\mathbf{x}}_i$. Astec's negative sampling step incurs a cost of $O(ND \log L)$, assuming $N \leq L^{O(1)}$ and took only a few minutes on most datasets and at most 2 hours on the AmazonTitles and Q2B datasets with up to 3 million labels. Note that the shortlist could be further extended

with the labels selected based on explicit label features, however label features are beyond the scope of Astec.

THEOREM 1. *Let $\lambda$ be the spectral norm of $\mathbf{R}$, $\mathcal{P}_l = \{i | y_{il} = +1\}$ be the set of positive training points for the label $l$, $\epsilon_l = (1+\lambda\sqrt{|\mathcal{P}_l|})^2 - 1$, and $\boldsymbol{\mu}_l$ be the label representation that could have been computed using final features as $\boldsymbol{\mu}_l = \frac{\hat{\boldsymbol{\mu}}_l}{\|\hat{\boldsymbol{\mu}}_l\|_2}$ where $\hat{\boldsymbol{\mu}}_l = \frac{1}{|\mathcal{P}_l|}\sum_{i\in\mathcal{P}_l}\hat{\mathbf{x}}_i$ and $\hat{\mathbf{x}}_i = \mathcal{Z}(\mathbf{x})$. Then: (a) $\|\hat{\mathbf{x}}_i - \mathbf{v}_i\|_2 \leq \lambda\|\mathbf{v}_i\|_2 \ \forall i$, i.e. the final and intermediate features for any document can be brought arbitrarily close by restricting $\lambda$; and (b) $\frac{C(\mathbf{v}_i,\boldsymbol{\mu}_l^0)}{1+\epsilon_l} \leq C(\hat{\mathbf{x}}_i,\boldsymbol{\mu}_l) \leq C(\mathbf{v}_i,\boldsymbol{\mu}_l^0)+\epsilon_l$, i.e. the cosine similarity $C(\mathbf{a},\mathbf{b}) := \frac{\mathbf{a}^\top\mathbf{b}}{\|\mathbf{a}\|_2\|\mathbf{b}\|_2}$ used in the ANNS algorithm to determine the similarity between any document $i$ and label $l$ can also be made arbitrarily close between the intermediate and final representations by restricting $\lambda$, the spectral norm of $\mathbf{R}$. [Please refer to A.6 in the supplementary material for a proof.]*

**Module III**: Astec uses a final feature representation of the form $\hat{\mathbf{x}} := \mathcal{Z}(\mathbf{x}) = \mathbf{v}+\Delta\mathbf{v}$, where $\mathbf{v}$ is shared with $\mathcal{Z}^0$ and $\Delta\mathbf{v} = \text{ReLU}(\mathbf{R}\mathbf{v})$ for a residual matrix $\mathbf{R}$. Note that $\mathbf{R}$ is distinct from the residual matrix $\mathbf{R}^0$ used in $\mathcal{Z}^0$. Since $D^2 \ll VD$, this introduced a negligible increase in model parameters and computational complexity and $\Delta\mathbf{v}$ could be computed in under 10 $\mu$s on a CPU thereby meeting the low-latency constraints. The spectral norm of $\mathbf{R}$ was restricted to be no greater than $\lambda \leq 1$ to ensure that feature representations offered by $\mathcal{Z}$ did not lie far away from those offered by $\mathcal{Z}^0$ (see Theorem 1). Prediction accuracy was observed to degrade if the spectral norm was not restricted or if the residual block was replaced by a fully connected block [32] since such steps allowed the final features to drift away from the intermediate features.

**Module IV**: Astec adopts the high capacity 1-vs-All classifier model and learns a $D$-dimensional classifier per label. Thus, its classifier is parametrized as $\mathbf{W} = [\mathbf{w}_1, \ldots, \mathbf{w}_L] \in \mathbb{R}^{D\times L}$. The classifier and residual block $\mathbf{R}$ present in $\mathcal{Z}$ were jointly trained in $O(ND\log L)$ time by restricting training to the positive and shortlisted negative labels for each data point. Specifically, for any $i \in [N]$, let $\mathcal{P}_i := \{l : y_{il} = +1\}$ be the set of positive labels of the data point, $\hat{\mathcal{N}}_i = \hat{\mathcal{N}}_i^\mu \cup \hat{\mathcal{N}}_i^z$ be the negative labels shortlisted in Module II and let $\hat{\mathcal{S}}_i := \hat{\mathcal{N}}_i \cup \mathcal{P}_i$. Then the approximate objective $\arg\min_{\mathbf{R},\mathbf{W}} \hat{\mathcal{L}}(\mathbf{R},\mathbf{W})$ was solved where

$$\hat{\mathcal{L}}(\mathbf{R},\mathbf{W}) = \sum_{i=1}^{N}\sum_{l\in\hat{\mathcal{S}}_i}\log\left(1+\exp\left(-y_{il}\cdot\mathbf{w}_l^\top\hat{\mathbf{x}}_i\right)\right),$$

subject to the constraint $\sup_{\mathbf{u}\neq\mathbf{0}} \|\mathbf{R}\mathbf{u}\|_2/\|\mathbf{u}\|_2 \leq \lambda$, where $\mathbf{W} = [\mathbf{w}_1, \ldots, \mathbf{w}_L] \in \mathbb{R}^{D\times L}$ are the 1-vs-All classifiers and $\hat{\mathbf{x}}_i = \mathcal{Z}(\mathbf{x}_i)$. This formulation could be optimized efficiently using the Adam optimizer on a single P40 GPU in 0.25–5 hours on the datasets considered in this paper.

**Re-ranking the labels**: Astec improves its accuracy by learning a novel re-ranker as follows. Astec's predictions $\hat{\mathbf{y}}_i \in \{-1,+1\}^L$ were obtained for each training point $i$. Excluding the true positives yielded a shortlist of negative labels that Astec found the most confusing for each point. A re-ranker was then trained to eliminate these mis-predictions by optimizing $\arg\min_{\widetilde{\mathcal{Z}},\widetilde{\mathbf{W}}} \tilde{\mathcal{L}}(\widetilde{\mathcal{Z}},\widetilde{\mathbf{W}})$ where

$$\tilde{\mathcal{L}}(\widetilde{\mathcal{Z}},\widetilde{\mathbf{W}}) = \sum_{i=1}^{N}\sum_{l\in\widetilde{\mathcal{S}}_i}\log\left(1+\exp\left(-y_{il}\cdot\widetilde{\mathbf{w}}_l^\top\widetilde{\mathbf{x}}_i\right)\right),$$

where $\widetilde{\mathcal{S}}_i = \{l : y_{il} = +1\} \cup \{l : \hat{y}_{il} = +1\}$, $\widetilde{\mathbf{x}}_i = \widetilde{\mathcal{Z}}(\mathbf{x}_i)$ and $\widetilde{\mathcal{Z}}$ had an architecture similar to $\mathcal{Z}$ but with independent parameters $\widetilde{\mathbf{E}}, \widetilde{\mathbf{R}}$. This increased accuracy by 1–3% with comparatively larger gains on larger datasets, with only a 10–20% increase in training time.

**Log-time prediction**: Astec meets the latency requirements of online short text applications on even the largest datasets by making predictions in $O(D^2 + D\log L)$ time. Given a test document $\mathbf{x} \in \mathbb{R}^V$, the $O(\log L)$ most relevant labels $\hat{\mathcal{S}} := \mathcal{N}^\mu \cup \mathcal{N}^x$ along with their base similarity scores $s^l(\mathbf{v})$ were shortlisted using $\mathbf{v} := \text{ReLU}\left(\sum_{t=1}^{V} x_t \cdot \mathbf{e}_t\right) \in \mathbb{R}^D$. Next, the final features $\hat{\mathbf{x}} = \mathcal{Z}(\mathbf{x})$ and 1-vs-All classifiers were used to yield scores $\hat{y}_l = \alpha\sigma(\mathbf{w}_l^\top\hat{\mathbf{x}}) + (1-\alpha)\sigma(s^l(\mathbf{v}))$ for shortlisted labels $l \in \hat{\mathcal{S}}$ and $\hat{y}_l = 0$ otherwise where $\sigma$ is the sigmoid function and $\alpha \in [0,1]$ is a hyperparameter. Final predictions were given by linearly combining the re-ranker scores $\tilde{y}_l = \sigma(\widetilde{\mathbf{w}}_l^\top\widetilde{\mathbf{x}})$ with the base similarity scores as $\overline{y}_l = \beta\hat{y}_l + (1-\beta)\widetilde{y}_l$ if $l \in \hat{\mathcal{S}}$ and $\overline{y}_l = 0$ otherwise, using a hyper-parameter $\beta \in [0,1]$.

## 5 EXPERIMENTS

**Datasets**: Results are presented on publically available benchmark short text datasets with up to 3 million labels for predicting frequently bought together Amazon items based on just the product title (AmazonTitles-670K and AmazonTitles-3M) as well as using a Wikipedia article's title to predict it's Wikipedia tags (WikiTitles-500K) as well as its related Wikipedia articles (WikiSeeAlsoTitles-350K). All datasets are available on the Extreme Classification Repository [6]. Note that, even though the focus of this paper is on short text applications and Astec has been designed keeping their specific requirements in mind, results on benchmark long text datasets are presented in the supplementary material for completeness. Results are also presented on a proprietary Bing dataset with 3 million labels and 21 million training points for matching user queries to advertiser bid phrases (Q2B-3M). The dataset was created by mining Bing's click logs where each user's query was treated as a data point and clicked advertiser bid phrases became its labels. Table 3 in the supplementary material presents the data set statistics.

**XML baselines**: The focus of this paper is on comparing Astec to MACH [32] and Slice [19] as they have been specifically designed for short text applications. Astec was also compared to other leading deep extreme classifiers including XML-CNN [28], XTransformers [11] and AttentionXML [56]. Furthermore, for the sake of completeness, results are presented for non-deep extreme classifiers including XT [50], DiSMEC [2], PfastreXML [20], Parabel [41], Bonsai [24] and AnneXML [47]. Implementations of all the baseline algorithms were provided by their authors. The hyper-parameters of these algorithms were set as suggested by their authors wherever applicable and by fine-grained validation otherwise. Results are only presented for those datasets to which an implementation could scale. Results could therefore not be reported for XTransformers as it could not be trained on any of the datasets on a single GPU in a week. As is customary in extreme classification, results are

**Table 1: Astec could be significantly more accurate and scalable than leading deep extreme classifiers including MACH, XML-CNN and AttentionXML on publicly available benchmark datasets. Astec was also found to outperform leading methods designed to match user queries to bid phrases on the Q2B-3M dataset. Results for other methods and metrics are presented in the supplementary material.**

| Method | P@1 | P@3 | P@5 | PSP@3 | PSP@5 | Training Time (hr) |
|---|---|---|---|---|---|---|
| Q2B-3M | | | | | | |
| Astec | **73.37** | **33.91** | **21.67** | **85.13** | **90.42** | 15.59 |
| Parabel | 54.29 | 27.15 | 17.94 | 61.66 | 68.61 | 3.52 |
| Slice+CDSSM | 53.23 | 27.53 | 18.56 | 64.51 | 74.14 | 4.71 |
| Seq2Seq | 28.25 | 13.06 | 8.02 | 17.59 | 15.42 | - |
| Simrank++ | 52.70 | 29.69 | 19.33 | 41.67 | 39.36 | 25.00 |
| AmazonTitles-670K | | | | | | |
| Astec | 39.97 | 35.73 | 32.59 | 29.79 | 31.71 | 1.29 |
| Astec-3 | **40.63** | **36.22** | **33.00** | **30.17** | **32.07** | 3.85 |
| MACH | 34.92 | 31.18 | 28.56 | 23.14 | 25.79 | 6.41 |
| XML-CNN | 35.02 | 31.37 | 28.45 | 24.93 | 26.84 | 23.52 |
| Slice+fastText | 33.85 | 30.07 | 26.97 | 24.15 | 25.81 | 0.22 |
| AttentionXML | 37.92 | 33.73 | 30.57 | 26.43 | 28.39 | 37.50 |
| Parabel | 38.00 | 33.54 | 30.10 | 25.57 | 27.61 | 0.09 |
| Bonsai | 38.46 | 33.91 | 30.53 | 26.19 | 28.41 | 0.53 |
| DiSMEC | 38.12 | 34.03 | 31.15 | 25.46 | 28.67 | 11.74 |
| AmazonTitles-3M | | | | | | |
| Astec | 47.64 | 44.66 | 42.36 | 18.59 | 20.60 | 4.38 |
| Astec-3 | **48.74** | **45.70** | **43.31** | **18.89** | **20.94** | 13.04 |
| MACH | 37.10 | 33.57 | 31.33 | 8.61 | 9.46 | 40.48 |
| SLICE+fastText | 35.39 | 33.33 | 31.74 | 13.37 | 14.94 | 0.64 |
| Parabel | 46.42 | 43.81 | 41.71 | 15.58 | 17.55 | 1.54 |
| Bonsai | 46.89 | 44.38 | 42.30 | 16.66 | 18.75 | 9.90 |
| WikiSeeAlsoTitles-350K | | | | | | |
| Astec | 20.42 | 14.44 | 11.39 | 12.05 | 13.94 | 1.47 |
| Astec-3 | **20.61** | **14.58** | **11.49** | **12.16** | **14.04** | 4.36 |
| MACH | 14.79 | 9.57 | 7.13 | 7.02 | 7.54 | 7.44 |
| XML-CNN | 17.75 | 12.34 | 9.73 | 9.72 | 11.15 | 14.25 |
| Slice+fastText | 18.13 | 12.87 | 10.29 | 10.78 | 12.74 | 0.22 |
| AttentionXML | 15.86 | 10.43 | 8.01 | 7.20 | 8.15 | 30.44 |
| Parabel | 17.24 | 11.61 | 8.92 | 8.83 | 9.96 | 0.06 |
| Bonsai | 17.95 | 12.27 | 9.56 | 9.68 | 11.07 | 0.46 |
| DiSMEC | 16.61 | 11.57 | 9.14 | 9.19 | 10.74 | 6.62 |
| WikiTitles-500K | | | | | | |
| Astec | 46.01 | 25.62 | 18.18 | 18.59 | 18.95 | 4.45 |
| Astec-3 | **46.60** | **26.03** | **18.50** | **18.90** | **19.30** | 13.04 |
| MACH | 33.74 | 15.62 | 10.41 | 8.98 | 8.35 | 23.65 |
| XML-CNN | 43.45 | 23.24 | 16.53 | 14.74 | 14.98 | 55.21 |
| Slice+fastText | 28.07 | 16.78 | 12.28 | 14.69 | 15.33 | 0.54 |
| AttentionXML | 42.89 | 22.71 | 15.89 | 14.32 | 14.22 | 102.43 |
| Parabel | 42.50 | 23.04 | 16.21 | 16.12 | 16.16 | 0.34 |
| Bonsai | 42.60 | 23.08 | 16.25 | 16.85 | 16.90 | 2.94 |
| DiSMEC | 39.89 | 21.23 | 14.96 | 15.15 | 15.43 | 23.94 |

presented for not only Astec but also an ensemble with 3 learners referred to as Astec-3. Astec's hyper-parameters and their settings on various datasets are discussed in the supplementary material.

**Bing baselines**: The online flight results revealed the gains that Astec was able to achieve when added to a large ensemble of state-of-the-art techniques currently running in production on Bing including many leading techniques for query synthesis (constrained and unconstrained), graph based techniques (graph neural networks, random walks, sessions based methods, *etc.*), embedding methods (Siamese networks and two-tower models), extreme classifiers as well as techniques that leverage additional information (refer to Section 2). Offline results for some of these techniques such as Simrank++ [18] and a BERT based sequence-to-sequence constrained synthesis model [13, 27] are presented on the Q2B-3M dataset for matching queries to bid phrases.

**Evaluation metrics**: Performance was evaluated using precision@$k$ (P@$k$), and propensity scored precision@$k$ (PSP@$k$) which have been widely used in the extreme classification literature [2, 20]. Results on additional metrics such as nDCG@$k$ (N@$k$) and propensity scored nDCG@$k$ (PSN@$k$) have been included in supplementary material which also contains the definitions of all the metrics. All training times have been reported on a 24-core Intel Xeon 2.6 GHz machine with a single Nvidia P40 GPU unless stated otherwise.

**Table 1 - Offline results**: Astec's main competitors were Slice and MACH as they are extreme classifiers that have been developed for low-latency short text applications since their features could be extracted in milliseconds on a CPU while all other architectures required a GPU. Slice is not a deep learning method and was therefore trained on the same FastText embeddings that were used to initialize Astec. Nevertheless, Astec was 2.29-17.94% more accurate than Slice and 5.05-12.27% more accurate than MACH on the publically available Repository datasets and 20.14% more accurate than Slice on the Bing Q2B-3M dataset. Unfortunately, MACH could not be trained on the Q2B-3M dataset in a week on a single GPU whereas Astec could be trained in 13 hours. On the Repository datasets, Astec was 5-9× faster to train than MACH. Similarly, none of the other deep extreme classifiers could be trained on the AmazonTitles-3M and Q2B-3M datasets. On the smaller datasets, Astec was 2.05-4.56% more accurate and 9-29× faster to train than AttentionXML and XML-CNN respectively. Furthermore, Astec was at least 19% more accurate than all other methods on the Bing Q2B-3M dataset including SimRank++ and a BERT based sequence-to-sequence model that had been specifically designed for matching queries to bid phrases. These results demonstrate that Astec's features could be learnt accurately from limited training data and that the DeepXML framework enabled Astec to be significantly more scalable than all other deep extreme classifiers. Finally, for the sake of completeness, Astec was compared to non-deep learning extreme classifiers on the Repository datasets where it could be up to 6.12% more accurate and this increased marginally to 6.77% for the Astec-3 ensemble.

**Online results from Bing flights**: Astec was able to efficiently train in 20 hours on 4×P40 GPUs on various Bing internal datasets with up to 62 million labels that were far beyond the scaling capabilities of all other deep extreme classifiers. Furthermore, Astec's features could be extracted in microseconds on a CPU and its overall predictions made in a few milliseconds allowing it to make billions of predictions per day at peak rates of 120,000 queries per second on commodity hardware. This allowed Astec to be flighted for multiple short text applications on Bing with extremely low-latencies and high-throughputs including text ads, product ads, rich ads, native ads, retail product recommendation, news recommendation, personalized query recommendation, *etc.* Unfortunately, due to space

**Table 2: Astec's predicted bid phrases for the user query "what is diabetes type 2" are more accurate and diverse as compared to leading methods (M1–M3) in Bing. All mispredictions have been *italicized*.**

| Method | Predictions |
|---|---|
| Astec | definition diabetes type 2, what causes type 2 diabetes, do i have type 2 diabetes, what is type 2 diabetes mellitus what are the causes of diabetes type 2, type 2 diabetes |
| M1 | what is type ii diabetes, whats type 2 diabetes |
| M2 | type 2 diabetes |
| M3 | what is type 2 diabetes, what is type 1 and type 2 diabetes, type 2 diabetes, *what is email marketing*, *what is ptsd2* *what is anemia*, *what is radiation therapy* |

constraints, online results when Astec was added to the ensemble of state-of-the-art techniques currently in production can only be presented for just two of these applications.

**Personalized ads**: Billions of users were shown personalized ads as they surfed the web based on their browsing history in the first application. Each user's intent was represented by the set of queries that the user could have potentially asked on Bing to reach the last $K$ webpages that they had browsed. Astec was used to predict the set of Bing queries from each visited webpage's title in milliseconds. A GRU was used to model the user's last $K$ states in near real time. The GRU was then used to select a single predicted query that was passed through the Bing pipeline to show ads to the user. Human expert evaluation revealed that Astec increased the number of excellent predictions by more than 20% while reducing the number of fair and poor predictions by more than 10% as compared to the ensemble in production containing leading extreme classification, deep learning and IR techniques. Astec was also able to achieve 100% coverage when deployed online by making predictions for all visited webpages for all users within the latency and throughput constraints. This allowed Astec to increase the click-through-rate by 6.5% and revenue by more than 5%. Table 8 in the supplementary material shows examples comparing Astec's predictions to those of traditional approaches and demonstrates Astec's benefits over the state-of-the-art.

**Matching queries to bid phrases**: Astec treated each user's query as input and each advertiser bid phrase as a separate label in order to predict the set of bid phrases that could be matched to the user's query to show ads on Bing. Astec's offline accuracy was 19% higher than that of other approaches (see Table 1). This translated into an increase of 1.6% in revenue per thousand queries, 2.9% in match quality and 8.6% increase in query coverage over a large ensemble of state-of-the-art techniques when deployed online. Table 2 lists Astec's predicted bid phrases for the query "what is diabetes type". As can be seen, Astec's predictions could be more accurate and diverse than those of other methods which could fixate on the phrase "what is" and thereby make many mispredictions.

**Ablations**: Table 6 in the supplementary material presents the ablation results for each module validating Astec's design choices. First, much progress has been made in both designing and speeding up feature architectures such as BERT [13], Roberta [30], *etc.* that can easily be incorporated into the DeepXML framework, if desired. Unfortunately, doing so did not lead to accuracy gains on the short text extreme datasets and Astec's features were demonstrated to be at least 2% more accurate than those based on CNNs [28], MLPs [32] and BERT [13]. This indicates that Astec's features were more suitable for short text extreme classification and that more research is required on how to train and fine-tune transformer and other architectures when many features and labels have very limited training data. Second, training on Astec's surrogate task was demonstrated to be 1-4% more accurate than training using self-supervision, label selection or label low-rank projection indicating the suitability of label clustering and the importance of a well designed surrogate task. Third, it was demonstrated that the negative sampling strategy proposed in Astec could be 13% more accurate than the strategy in Slice. This further highlights the differences between Astec and Slice and indicates that, even if Slice could somehow have been trained jointly along with Astec's features, it would still have been significantly inferior to Astec. Finally, it was demonstrated that Astec's performance could not be further improved by replacing its classifier with alternatives such as DiSMEC or Parabel.

**State-of-the-art algorithms in the DeepXML framework**: DeepXML could be used to analyze and improve leading deep extreme classifiers thereby demonstrating its generality and usefulness. For instance, both XML-CNN and MACH could be recast into the DeepXML framework by replacing Astec's feature architecture by their CNN or MLP features. This led to accuracy gains of 1.8% and 2.4% and training speedups of 10× and 5× for XML-CNN and MACH respectively (see Table 4 in the supplementary material). Note that the original MACH algorithm stops training after the first DeepXML module, *i.e.,* after it has learnt its MLP features on the surrogate task, and therefore compensates by learning a large ensemble. Casting MACH into DeepXML allowed it to improve accuracy by fine-tuning its features for the extreme task at hand while speeding up training by learning just a single learner rather than an ensemble of 32 learners. Similarly, DeepXML increased XML-CNN's accuracy by fine-tuning its CNN features through the residual block in the third module and speeded up training by replacing its fully connected output layer with $O(L)$ complexity by a 1-vs-some classifier with $O(\log L)$ complexity in the fourth module. It should be reiterated that Astec's performance continued to remain superior to that of the improved MACH and XML-CNN.

**Vision task**: While the primary focus of this paper is on short text applications, DeepXML was also applied to a vision task to demonstrate its ease-of-use and generality. Table 9 in the supplementary material presents results for classifying a retail item's image into its product categories from the AmazonCat-13K dataset available on the Repository. DeepXML made only a minor modification in Astec by replacing its feature architecture by ResNet18 [16] while leaving everything else unchanged. Pre-trained ResNet18 features were also used to train Slice, MACH, Parabel and DiSMEC. Nevertheless, DeepXML was 4-29% more accurate than these methods thereby demonstrating its flexibility.

## 6 CONCLUSIONS

This paper developed the modular DeepXML framework that was used to: (a) derive the Astec algorithm for low-latency short text

applications; (b) analyse and improve leading deep extreme classifiers; and (c) provide a convenient and flexible tool for practitioners to plug in components of their choice with minimal effort for tackling diverse applications. Furthermore, Astec was demonstrated to be significantly more accurate and scalable than leading extreme classifiers for short text documents and could lead to significant gains in various online metrics for multiple applications on Bing.

## ACKNOWLEDGMENTS

The authors thank Purushottam Kar, Aditya Kusupati, Harsha Vardhan Simhadri, Yash Garg, and Risi Thonangi for helpful feedback.

## REFERENCES

[1] R. Agrawal, A. Gupta, Y. Prabhu, and M. Varma. 2013. Multi-label learning with millions of labels: Recommending advertiser bid phrases for web pages. In *WWW*.
[2] R. Babbar and B. Schölkopf. 2017. DiSMEC: Distributed Sparse Machines for Extreme Multi-label Classification. In *WSDM*.
[3] R. Babbar and B. Schölkopf. 2019. Data scarcity, robustness and extreme multi-label classification. *ML* (2019).
[4] X. Bai, E. Ordentlich, Y. Zhang, A. Feng, A. Ratnaparkhi, R. Somvanshi, and A. Tjahjadi. 2018. Scalable Query N-Gram Embedding for Improving Matching and Relevance in Sponsored Search. In *KDD*.
[5] E. J. Barezi, I. D. W., P. Fung, and H. R. Rabiee. 2019. A Submodular Feature-Aware Framework for Label Subset Selection in Extreme Classification Problems. In *NAACL*.
[6] K. Bhatia, K. Dahiya, H. Jain, A. Mittal, Y. Prabhu, and M. Varma. 2016. The Extreme Classification Repository: Multi-label Datasets & Code. http://manikvarma.org/downloads/XC/XMLRepository.html
[7] K. Bhatia, H. Jain, P. Kar, M. Varma, and P. Jain. 2015. Sparse Local Embeddings for Extreme Multi-label Classification. In *NeurIPS*.
[8] W. Bi and J. Kwok. 2013. Efficient multi-label classification with many labels. In *ICML*.
[9] A. Broder, P. Ciccolo, E. Gabrilovich, V. Josifovski, D. Metzler, L. Riedel, and J. Yuan. 2009. Online Expansion of Rare Queries for Sponsored Search. In *WWW*.
[10] I. Chalkidis, M. Fergadiotis, P. Malakasiotis, N. Aletras, and I. Androutsopoulos. 2019. Extreme Multi-Label Legal Text Classification: A case study in EU Legislation. In *NAACL*.
[11] W.-C. Chang, Yu H.-F., K. Zhong, Y. Yang, and I.-S. Dhillon. 2020. Taming Pretrained Transformers for Extreme Multi-label Text Classification. In *KDD*.
[12] W-C. Chang, F.-X. Yu, Y.-W. Chang, Y. Yang, and S. Kumar. 2020. Pre-training Tasks for Embedding-based Large-scale Retrieval. In *ICLR*.
[13] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. *NAACL* (2019).
[14] J. Gao, S. Xie, X. He, and A. Ali. 2012. Learning Lexicon Models from Search Logs for Query Expansion. In *EMNLP*.
[15] C. Guo, A. Mousavi, X. Wu, D.-N. Holtmann-Rice, S. Kale, S. Reddi, and S. Kumar. 2019. Breaking the Glass Ceiling for Embedding-Based Classifiers for Large Output Spaces. In *NeurIPS*.
[16] K. He, X. Zhang, S. Ren, and J. Sun. 2016. Deep Residual Learning for Image Recognition. In *CVPR*.
[17] P. S. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck. 2013. Learning Deep Structured Semantic Models for Web Search using Clickthrough Data. In *CIKM*.
[18] A. Ioannis, G. M. Hector, and C. C. Chi. 2008. Simrank++: Query Rewriting through Link Analysis of the Click Graph. In *WWW*.
[19] H. Jain, V. Balasubramanian, B. Chunduri, and M. Varma. 2019. Slice: Scalable Linear Extreme Classifiers trained on 100 Million Labels for Related Searches. In *WSDM*.
[20] H. Jain, Y. Prabhu, and M. Varma. 2016. Extreme Multi-label Loss Functions for Recommendation, Tagging, Ranking and Other Missing Label Applications. In *KDD*.
[21] K. Jasinska, K. Dembczynski, R. Busa-Fekete, K. Pfannschmidt, T. Klerx, and E. Hullermeier. 2016. Extreme F-measure Maximization using Sparse Probability Estimates. In *ICML*.
[22] Y. Jernite, A. Choromanska, and D. Sontag. 2017. Simultaneous Learning of Trees and Representations for Extreme Classification and Density Estimation. In *ICML*.
[23] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov. 2017. Bag of Tricks for Efficient Text Classification. In *EACL*.
[24] S. Khandagale, H. Xiao, and R. Babbar. 2019. Bonsai - Diverse and Shallow Trees for Extreme Multi-label Classification. *Machine Learning* (2019).
[25] W. Krichene, N. Mayoraz, S. Rendle, L. Zhang, X. Yi, L. Hong, E. Chi, and J. Anderson. 2019. Efficient training on very large corpora via gramian estimation. In *ICLR*.
[26] M. C. Lee, B. Gao, and R. Zhang. 2018. Rare Query Expansion Through Generative Adversarial Networks in Search Advertising. In *KDD*.
[27] Y. Lian, Z. Chen, J. Hu, K. Zhang, C. Yan, M. Tong, W. Han, H. Guan, Y. Li, Y. Cao, Y. Yu, Z. Li, X. Liu, and Y. Wang. 2019. An end-to-end Generative Retrieval Method for Sponsored Search Engine -Decoding Efficiently into a Closed Target Domain. *CoRR* (2019).
[28] J. Liu, W. Chang, Y. Wu, and Y. Yang. 2017. Deep Learning for Extreme Multi-label Text Classification. In *SIGIR*.
[29] X. Liu, P. He, W. Chen, and J. Gao. 2019. Multi-Task Deep Neural Networks for Natural Language Understanding. In *ACL*.
[30] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *CoRR* (2019).
[31] A. Y. Malkov and D. A. Yashunin. 2016. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. *CoRR* (2016).
[32] T. K. R. Medini, Q. Huang, Y. Wang, V. Mohan, and A. Shrivastava. 2019. Extreme Classification in Log Memory using Count-Min Sketch: A Case Study of Amazon Search with 50M Products. In *NeurIPS*.
[33] Q. Mei, D. Zhou, and K. Church. 2008. Query Suggestion Using Hitting Time. In *CIKM*.
[34] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. 2013. Distributed Representations of Words and Phrases and Their Compositionality. In *NeurIPS*.
[35] P. Mineiro and N. Karampatziakis. 2015. Fast Label Embeddings via Randomized Linear Algebra. In *ECML/PKDD*.
[36] A. Mittal, K. Dahiya, S. Agrawal, D. Saini, S. Agarwal, P. Kar, and M. Varma. 2021. DECAF: Deep Extreme Classification with Label Features. In *WSDM*.
[37] A. Mittal, N. Sachdeva, S. Agrawal, S. Agarwal, P. Kar, and M. Varma. 2021. ECLARE: Extreme Classification with Label Graph Correlations. In *TheWebConf*.
[38] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. 2018. Spectral Normalization for Generative Adversarial Networks. *CoRR* (2018).
[39] S. J. Pan and Q. Yang. 2010. A Survey on Transfer Learning. *TKDE* (2010).
[40] Y. Prabhu, A. Kag, S. Gopinath, K. Dahiya, S. Harsola, R. Agrawal, and M. Varma. 2018. Extreme multi-label learning with label features for warm-start tagging, ranking and recommendation. In *WSDM*.
[41] Y. Prabhu, A. Kag, S. Harsola, R. Agrawal, and M. Varma. 2018. Parabel: Partitioned label trees for extreme classification with application to dynamic search advertising. In *WWW*.
[42] Y. Prabhu and M. Varma. 2014. FastXML: A Fast, Accurate and Stable Tree-classifier for eXtreme Multi-label Learning. In *KDD*.
[43] A .S Rawat, J. J. Chen, F. Yu, Suresh A. .T, and S. Kumar. 2019. Sampled softmax with random fourier features. In *NeurIPS*.
[44] N. Reimers and I. Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *EMNLP*.
[45] D. Saini, A.K. Jain, Kushal. Dave, J. Jiao, A. Singh, R. Zhang, and M. Varma. 2021. GalaXC: Graph neural networks with labelwise attention for extreme classification. In *TheWebConf*.
[46] W. Siblini, P. Kuntz, and F. Meyer. 2018. CRAFTML, an Efficient Clustering-based Random Forest for Extreme Multi-label Learning. In *ICML*.
[47] Y. Tagami. 2017. AnnexML: Approximate Nearest Neighbor Search for Extreme Multi-label Classification. In *KDD*.
[48] Y. X. Wang, D. Ramanan, and M. Hebert. 2017. Learning to Model the Tail. In *NeurIPS*.
[49] T. Wei, W. W. Tu, and Y. F. Li. 2019. Learning for Tail Label Data: A Label-Specific Feature Approach. In *IJCAI*.
[50] M. Wydmuch, K. Jasinska, M. Kuznetsov, R. Busa-Fekete, and K. Dembczynski. 2018. A no-regret generalization of hierarchical softmax to extreme multi-label classification. In *NeurIPS*.
[51] H. Ye, Z. Chen, D.-H. Wang, and B. D. Davison. 2020. Pretrained Generalized Autoregressive Model with Adaptive Probabilistic Label Clusters for Extreme Multi-label Text Classification. In *ICML*.
[52] C. Yejin, F. Marcus, G. Evgeniy, Vanja. J., M. Mauricio, and P. Bo. 2010. Using Landing Pages for Sponsored Search Ad Selection. In *WWW*.
[53] E.H. I. Yen, X. Huang, K. Zhong, P. Ravikumar, and I. S. Dhillon. 2016. PD-Sparse: A Primal and Dual Sparse Approach to Extreme Multiclass and Multilabel Classification. In *ICML*.
[54] I. Yen, S. Kale, F. Yu, D. Holtmann R., S. Kumar, and P. Ravikumar. 2018. Loss Decomposition for Fast Learning in Large Output Spaces. In *ICML*.
[55] X. Yi, J. Yang, L. Hong, D. Z. Cheng, L. Heldt, A. Kumthekar, Z. Zhao, L. Wei, and E. Chi. 2019. Sampling-Bias-Corrected Neural Modeling for Large Corpus Item Recommendations. In *RecSys*.
[56] R. You, S. Dai, Z. Zhang, H. Mamitsuka, and S. Zhu. 2019. AttentionXML: Extreme Multi-Label Text Classification with Multi-Label Attention Based Recurrent Neural Networks. In *NeurIPS*.
[57] Z. Yuan, Z. Guo, Yu X., X. Wang, and T. Yang. 2020. Accelerating Deep Learning with Millions of Classes. In *ECCV*.
[58] H. Zhou, M. Huang, Y. Mao, C. Zhu, P. Shu, and X. Zhu. 2019. Domain-Constrained Advertising Keyword Generation. In *WWW*.