SST: Semantic and Structural Transformers for Hierarchy-aware Language Models in E-commerce

Karan Samel* — *Georgia Tech* — ksamel@gatech.edu
Houyu Zhang — *Amazon* — zhanhouy@amazon.com
Jun Ma+ — *Walgreens AI Lab* — jun.ma@walgreens.com
Haoming Jiang — *Amazon* — jhaoming@amazon.com
Qing Ping — *Amazon* — pingqing@amazon.com
Sheng Wang — *University of Washington, Amazon* — swanguw@amazon.com
Yi Xu — *Amazon* — yxaamzn@amazon.com
Belinda Zeng — *Amazon* — zengb@amazon.com
Trishul Chilimbi — *Amazon* — trishulc@amazon.com

***Abstract*—Hierarchies are common structures used to organize data, such as e-commerce hierarchies associated with product data. With these product hierarchies, we aim to learn hierarchy-aware product text embeddings to improve fine-tuning performance on a variety of downstream e-commerce tasks. Existing methods leverage hierarchies by either aligning the text embeddings to separate hierarchical embeddings or by aligning the hierarchical information implicitly within a unified text Transformer. Although these models optimize to predict hierarchy information, performing further fine-tuning on new tasks is non-trivial. To bridge this gap, we propose a pre-training architecture to implicitly encode the hierarchy within the product text and then directly leverage a sub-set of the pre-training model during fine-tuning. Pre-training is done through Semantic and Structural Transformers (SST) where the Semantic-Transformer first encodes the product text into a contextual embedding, which is then used by the Structural-Transformer to infer the product's path in the hierarchy. Fine-tuning is done using only the initial Semantic-Transformer, now that hierarchy-aware text embeddings are learned. With this design, we eliminate the need of linking each fine-tuning dataset with corresponding hierarchies. This leads to fine-tuning performance improvements on critical e-commerce downstream tasks over the existing state-of-the-art hierarchy models, even when hierarchy data *is* available during fine-tuning. Moreover, this improvement is consistent even after augmenting our baseline models to support fine-tuning. We conclude by discussing how such implicit structural encodings can be leveraged beyond the e-commerce domain.**

***Index Terms*—hierarchical models, language modeling, e-commerce, product classification**

I. Introduction

Hierarchies are used to effectively represent large-scale text data in a structured manner [1]–[4]. In e-commerce, there are large-scale fine-grained hierarchies constructed to organize products. These are used by customers to easily browse and navigate between different categories of products [5]–[7]. While products are organized within hierarchies, these structures are not typically used during language model pre-training. The most common pre-training technique is masked language modeling (MLM) [8] which learns the semantics of token-level relations within a product's description. In addition, we would also want the representation of products to be similar within the nodes in a taxonomy.

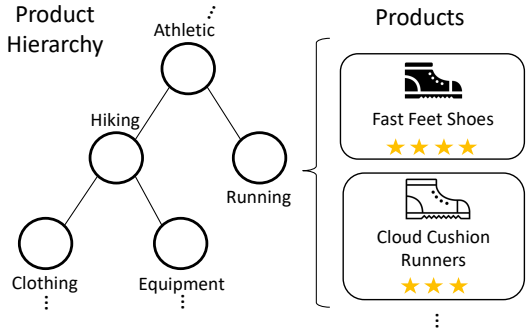

Fig. 1. A sample e-commerce hierarchy and the corresponding products within a hierarchy node. The hierarchy is used to pre-train structurally aware product embeddings to improve their downstream fine-tuning capabilities.

For example in Figure 1, we see two products within the *Running* node. During MLM pre-training we can infer the token "Feet", given the tokens "Fast" and "Shoes". Similarly given "Cloud Cushion" we can associate the term "Runners". However, we cannot directly capture the similarity between "Fast Feet" and "Cloud Cushion", which are both running shoes. On the other hand, such similarity can be captured by associating both products to the *Running* node in the hierarchy. Therefore, we propose to embed the hierarchy implicitly within product representations during pre-training. Such pre-training is useful in many downstream e-commerce tasks that rely on this implicit hierarchical knowledge to categorize related products or match products to user queries.

There are also practical fine-tuning constraints when leveraging a hierarchy-aware pre-trained model. First, some downstream tasks may not have the hierarchy information associated with the text, such as an incoming user query [9], [10]. Second, the hierarchy does not always cover all products. This occurs when the hierarchy information is noisy, the

*Work completed during an internship at Amazon.
+Work done while at Amazon.

979-8-3503-2445-7/23/$31.00 ©2023 IEEE

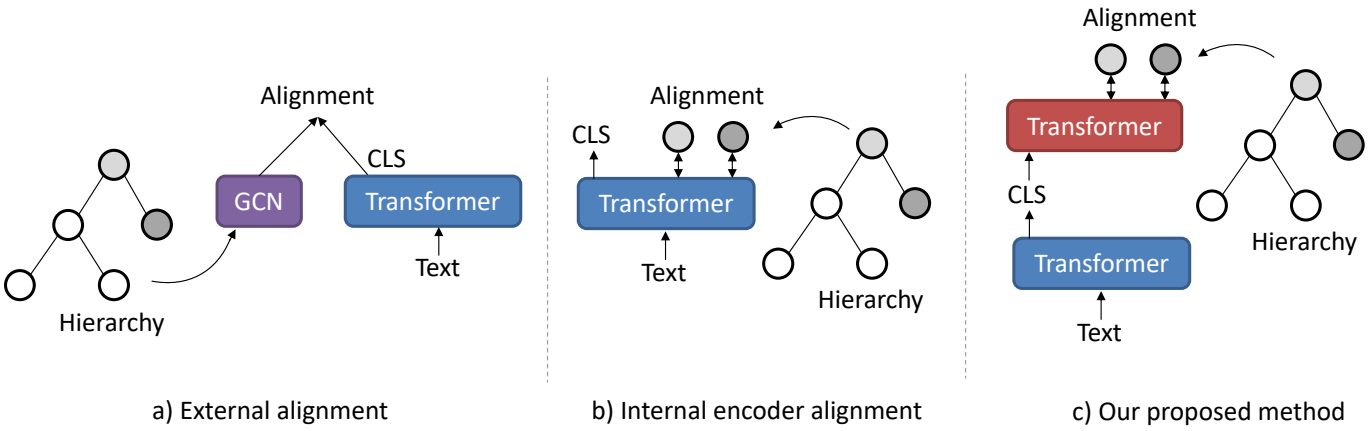

Fig. 2. We depict the different strategies used in hierarchy prediction. The first strategy on the left in a) focuses on aligning a graph encoding of the hierarchy to the text representation. The second strategy in the middle b) aligns the node embeddings internally within the text encoder. To train on downstream tasks our proposed method on the right c) forces the model to learn the hierarchical alignment through the compressed text representation. Then the bottom Transformer encoder, which captures hierarchy-aware embeddings, can be used directly for fine-tuning.

product is new and not manually assigned to the hierarchy [11]–[13], or when using product data from external sources [14]–[16]. Finally, there are additional considerations when it comes to storing the hierarchy metadata and any hierarchy-specific model architectures. This leads to additional overhead and complexity for development. For these reasons, we are motivated *not* to require the hierarchy information during fine-tuning, and only rely on the available fine-tuning input data.

Since we restrict hierarchy data in fine-tuning, we leverage it to learn hierarchy-aware product representations *at scale* during the pre-training stage instead. Moreover, any model parameters learned from this pre-training stage should be transferable to a standard Transformer [8], [17]–[19] for fine-tuning. This is crucial for collaborative ML teams in the industry to quickly onboard and evaluate new models with minimal friction. It is also beneficial for the general research community, which can easily adapt a standard Transformer model with hierarchy-aware embeddings. Such models can be leveraged as a backbone for datasets with inherent hierarchical structures or further architecture modification.

a) Methodological contribution.: To explore this hierarchy pre-training and hierarchy-agnostic fine-tuning paradigm, we propose a class of Semantic and Structural Transformers. During pre-training, the Semantic-Transformer first provides a contextualized text embedding of the product and then the Structural-Transformer predicts the product hierarchy path from this embedding. During fine-tuning, the Structural-Transformer can be used directly as a standard backbone model. It is pre-trained with hierarchical data but does not need the associated hierarchies for fine-tuning datasets.

b) Experimental contribution.: We test hierarchy-aware baselines along with our proposed work on four e-commerce fine-tuning tasks. We improve over the state-of-the-art hierarchy pre-training method on average by 0.5% and by 2.3% in long-tail settings, while using a simpler architecture. This improvement is consistent even after augmenting our baselines to improve their fine-tuning capabilities. We also obtain further performance improvements using our method if hierarchical data is available during fine-tuning.

II. Related Works

Given a hierarchy and text documents, most works focus on hierarchical text classification (HTC). For HTC there are two primary strategies. The first strategy is to align the representations from the outputs of the text encoder and hierarchy encoder models, as depicted in Figure 2 a). The second strategy, shown in Figure 2 b), aligns text and the hierarchy within a Transformer to provide a single hierarchy-aware text encoder.

Within the first strategy, HiAGM [20] uses a graph convolution network (GCN) to first generate hierarchy embeddings. Then it computes soft attention between the text and hierarchy features to infer the hierarchy labels. HiMatch [21] uses a contrastive loss to align a GCN embedding of the hierarchy node to the text representation. For negative samples, it uses a margin-based loss, where the margin varies based on the path distance from the positive node. The alignment and negative sampling loss are also computed jointly with the downstream fine-tuning task losses end-to-end. These methods differ from our use case where we want to perform this alignment during pre-training and then performing fine-tuning disjointly. Without joint fine-tuning, we explore how such graph alignment methods perform on downstream tasks.

Within the second strategy, HGCLR [22] uses a Graphormer [23] to encode the hierarchy within the same text Transformer encoder. It uses this graph encoder to generate synthetic positive samples which are also used to train its node-text contrastive objective. HPT [24] uses a graph attention model [25] to encode each level of the hierarchy and appends it into Transformer input text as prompt tokens to encode the global structure. For each level-wise input prompt, it also appends a masked node embedding. It then uses the output

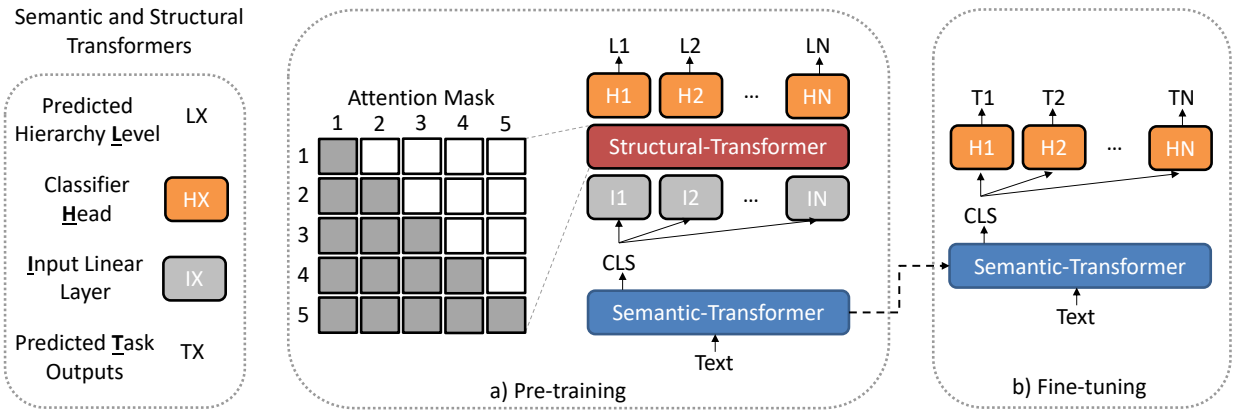

Fig. 3. Our proposed method: a) During pre-training, our architecture first encodes documents using a Semantic-Transformer into the CLS representation, which is then used to predict the document's hierarchy path at every level using a Structural-Transformer. Here we detail the attention mask used to prevent information leakage from deeper levels to shallower ones. b) During fine-tuning we load the pre-trained Semantic-Transformer and train classifier heads for each downstream task.

hidden states from these masked inputs to align with the ground truth hierarchy path embeddings. HBGL [26] first pre-trains the individual node representations by predicting masked nodes given their surrounding context nodes. During training, it concatenates the text, ground truth path embedding, and masked node embeddings. The objective is to recover the ground truth path from the masked nodes, where self-attention between the ground truth and masked nodes is blocked. During inference, the hierarchy path is directly inferred from masked input nodes.

It is unclear if these hierarchy-aligned Transformers can be used for further task-agnostic fine-tuning. To enable explicit fine-tuning, we split hierarchy prediction into two Transformer levels. In the first level, we capture the *semantics* of the text through the output CLS embedding. In the second level, we use this CLS embedding to infer the hierarchy *structure*, as shown in 2 c). This enables us to use the first Semantic-Transformer as a general backbone for downstream fine-tuning, while the full Semantic and Structural Transformers model is only leveraged during pre-training.

III. Problem Definition

We start with a tree structure hierarchy H containing nodes $n_i \in H$. These hierarchy nodes n_i are associated with a single parent node and children node(s). At the first level, the hierarchy contains root node(s), which have no parents. Each level may also contain leaf nodes that have parents, but no subsequent children. During pre-training are also given text documents D where each document is linked to a corresponding hierarchy leaf node n_d^l at level $l \in [1, L]$ in the hierarchy. Here we define L as the max depth of the hierarchy. Correspondingly we can also define a path of nodes $P = \{n_d^1, n_d^2, \ldots, n_d^l\}$ for the associated text document. This starts with the highest level node at the root level n_d^1 and goes down to the leaf node n_d^l directly associated with the document. During pre-training, we are provided with documents as well as their corresponding paths $\{(D_i, P_i)\}_{i=1}^N$.

For each fine-tuning task, the input data format may differ from the document data used in pre-training. Unlike pre-training, we don't assume that hierarchical information is associated with each fine-tuning input sample. Each fine-tuning input is defined as X and has the corresponding labels y. The model obtained from our hierarchy pre-training is fine-tuned on each dataset $\{(X_i^t, y_i^t)\}_{i=1}^{N^t}$ separately for each task $t \in T$. We evaluate our methods based on their performance on these fine-tuning tasks.

IV. Methodology

To motivate our methodology, we observe that previous Transformer-based HTC methods use full-text self-attention to predict the hierarchy. It is unclear if these methods lead to hierarchy-aware CLS output embeddings since they use all tokens to predict the hierarchy and not just the CLS token. This is important since during downstream classification tasks we will be only using the CLS output hidden state. Therefore the hierarchical context of a document must be present within the CLS output itself. We illustrate how to best utilize the CLS representation for hierarchical pre-training in order to have a hierarchy-aware CLS representation for fine-tuning.

A. Architecture

For our pre-training model we first encode the document semantics through a standard Transformer, which we call a Semantic-Transformer, to obtain the output CLS hidden representation $h_{[CLS]}$. Then we can use this representation to predict the hierarchy path structure P through a Structural-Transformer.

The intuition is that during pre-training the Semantic-Transformer becomes hierarchy-aware since the Structural-Transformer is forced to tune the $h_{[CLS]}$ output to predict the hierarchy path. This provides the flexibility of encoding the document text while guaranteeing that the hierarchy is encoded within the same CLS embedding. With the Semantic-Transformer implicitly encoding the product's hierarchy, it

doesn't require explicit hierarchy data during downstream tasks. Therefore the Structural-Transformer is discarded during fine-tuning. These Semantic and Structural Transformers (SST) are illustrated in Figure 3.

The Structural-Transformer only inputs a single $h_{[CLS]}$ embedding, whereas typically a sequence of embeddings is input into a Transformer. To distinguish between the input spaces, we first project the $h_{[CLS]}$ embedding using linear layers to create inputs for each level in our hierarchy path P: $I_l = h_{[CLS]}W_l^\top + b_l$ for each $l \in L$. Here $W_l \in \mathbb{R}^{k \times k}$ and $b_l \in R_k$, where k is the dimension of the hidden state. Then the full sequence input to our Structural-Transformer is the input for each level $I = [I_1, I_2, I_3, \ldots, I_L] \in \mathbb{R}^{L \times k}$.

Interactions from the deeper levels to the shallower levels of hierarchy inputs I are blocked by an attention mask M to prevent information leakage:

$$M_{i,j} = \begin{cases} 0 & \text{if } i \geq j \\ -\infty & \text{otherwise} \end{cases} \in \mathbb{R}^{L \times L} \quad (1)$$

This masking is necessary since providing a deeper level node can automatically imply which shallower nodes to predict along the hierarchy path P. The opposite interactions between shallower to deeper level nodes remain intact to let the model leverage ancestor node information when predicting descendants. This mask M is added in the self-attention step to enforce these interactions:

$$\text{Attention}(Q, K, V, M) = \text{softmax}(\frac{QK^\top + M}{\sqrt{k}})V \quad (2)$$

We provide a motivating example for this masking strategy using our Figure 1 example. We have a hierarchy node for *Running*, which is a child of the node *Athletic*. We don't want the level input for *Running* to be visible to earlier levels, since we want to infer the *Athletic* node from context and not because *Running* is always a descendant of *Athletic*. In contrast, knowing that we are in the *Athletic* branch is a good prior to predicting *Running*. Even with the prior, we still need to disambiguate between the different siblings for *Running* in the same level, such as *Hiking*.

We can then obtain the output hidden states $O = \text{Structural-Transformer}(I, M)$ where each output hidden state $o_l \in O$ corresponds to a level $l \in L$. Then we take each o_l output and predict the corresponding node $n_d^l \in P$ at that level using a linear classifier head: $\hat{n}_d^l = o_l A_l^\top + c_l$. At each level, we only predict between the nodes at the same level. We can define the set of nodes in each level as $H_l = \{n \in H \mid \text{level}(n) = l\}$. Then the parameter dimension of our heads are $A_l \in \mathbb{R}^{|H_l| \times k}$ and $c_l \in \mathbb{R}^{|H_l|}$ for each level l.

B. Training

During hierarchy pre-training, we optimize the node predicted at each level of the Structural-Transformer to align with the path of the document. For each level, we optimize a standard binary cross entropy (BCE) loss between the predicted node and provided node and then sum the losses at each level: $\mathcal{L} = \sum_{l=1}^{L} \text{BCE}(\hat{n}_d^l, n_d^l)$. Using BCE allows for predicting multiple nodes at each level. Therefore enabling us to associate multiple paths to each document. It also suppresses predictions for deeper hierarchy levels if the path P for a document stops at a shallower level than the max depth $l < L$. In contrast, the softmax function in a cross entropy (CE) objective will force a prediction at each hierarchy level, even if the ground truth path does not go to the max depth.

During fine tuning we simply take the Semantic-transformer and add a linear head to fine-tune on top of the CLS output $\hat{y} = h_{[CLS]}V_t^\top + z_t$. Again, the CLS representation from the Semantic-Transformer is a hierarchy-aware document representation, forgoing the need for the hierarchy or the second Transformer during fine-tuning. Then we can use this output to compute the loss for each task $L_t = \text{loss}(\hat{y}, y)$ where loss is BCE for binary/multi-label tasks or CE for multi-class tasks.

V. Experimental Setup

A. Pre-training Hierarchy Data

We test our setup on real-world e-commerce data with raw product hierarchies and product document data. First, we obtain the product hierarchy H from a sampled subset of the full hierarchy. Here even our subsampled hierarchy $|H| \geq 30k$ is much larger than public hierarchies evaluated for previous HTC papers [22], [24], [26] which only contain a few hundred nodes.

For the product document data D, we obtain $N = 25M$ samples of products linked to our mined hierarchy. Products can also be linked to more than one node in the hierarchy. For example, an athletic shoe can belong to "running shoe" and "women's apparel" to provide a broader classification of products. Therefore multiple paths can be associated with each product.

B. Downstream Fine-tuning Data

TABLE I
DATA STATISTICS FOR OUR FINE-TUNING DATASETS.

Dataset	Samples (Train / Val / Test)	Num Labels
Product Cat.	723k / 90k / 90k	1000+
Query-Product	1M / 125k / 125k	4
Local Financial	410k / 51k / 51k	100-1000
Global Financial	25k / 3k / 3k	<100

To evaluate our hierarchy pre-training methods, we test four downstream e-commerce application tasks. These are sub-sampled datasets regarding query-product quality, product, local financial, and global financial categorization.

Product categorization (PC) data involves predicting high-level categories of the products, which are more general than our hierarchy categories. The dataset contains over a thousand unique labels, but the samples are highly skewed towards common categories, such as "shirt" or "shoes". This means that the most common 100 labels cover 80-90% of the

data. This provides opportunities to leverage hierarchical pre-training to learn the inherent structure across this large number of long-tailed labels.

For the query-product (QP) dataset, the goal of this task is to predict how relevant a product is given a customer query. Instead of a ranking task, this dataset is discretized into a multi-class classification problem. If the product is a perfect match to the query, then it is labeled "exact". If the product is functionally similar, but not exact it is labeled "substitute". A product that is related to the general query category is assigned as "complement". Finally, if the recommended product is unrelated then it is labeled as "irrelevant". Similar to PT data, the distribution of labels is also skewed. Here 68%, 20%, 2%, and 8% of data correspond to exact, substitute, complement, and irrelevant respectively. These PC and QP datasets provide opportunities to test models in the long tail setting.

We can also test our method for financial classification purposes. One category of data contains local financial classification (LF) tasks to group products based on local reporting regulations. Similarly we categorize products based on their global financial (GF) categorization requirements.

For each dataset we obtain, we split the data into 80%, 10%, and 10% for train, validation, and testing respectively. The number of samples and labels for each dataset are summarized in Table I. A publicly prepared version of the pre-training data[1] consisting of hierarchy paths associated to products and a subset of the QP fine-tuning dataset[2] is also available.

C. Training Setup

We test HTC baselines and proposed pre-training methods on downstream tasks. We use the base 12 layer ALBERT model [19] as the specific Transformer backbone for all baselines and proposed methods for consistency. For all experiments, we use an AdamW [27] optimizer with a learning rate of $1e^{-5}$. Since we test contrastive learning methods like HiMatch, we also accumulate the gradients over 5 batches, which is also done for all of our pre-training and fine-tuning experiments.

For initial MLM pre-training, we only pre-train on the sampled product data D till convergence. Hierarchical and MLM pre-training can be done jointly as in the original HPT paper, but is computationally expensive given the number of hierarchy pre-training methods we test. In our experiments, we perform multi-stage pre-training to reduce the cost of MLM pretraining to a single pre-training run. Once we pre-train using MLM we then load this ALBERT encoder for all subsequent hierarchical pre-training experiments.

Similarly the hierarchy pre-training methods are also conducted on the product D and hierarchy H data till convergence. All subsequent fine-tuning experiments were run for a max of 10 epochs, except for GF, for which we run up to 50 epochs due to its small sub-sampled dataset size. During fine-tuning, we checkpoint the best model based on its validation micro F1 score, which usually occurred before the max epoch is reached. This checkpoint model is evaluated on our final test datasets, which are the numbers reported.

[1]Products and associated hierarchies: https://tinyurl.com/57rajwew

[2]Public query-product data subset: https://tinyurl.com/ymavyxs8

D. Baselines

For our evaluation, we compare against recent HTC baselines. With HiMatch, we use an ALBERT encoder as similarly done with previous Transformer-HiMatch variations. We also use a Relational-GCN [28] to model the different parent, child, and sibling relations in the hierarchy. This is in contrast to a standard GCN used only between parent and child relations used in the original work.

The Transformer HTC baselines also required modifications to work at the scale of our hierarchies. HPT runs level-wise GAT to generate prompt features to feed into the transformer. However, our intermediate hierarchy levels contain a large number of nodes, which makes convolution over all nodes infeasible. To address this we perform random sampling of the nodes selected for convolution at each level, where we sample up to 500 nodes.

In contrast, HBGL doesn't use graph convolution. In its node pre-training stage, the original paper loads in the entire hierarchy to perform masked pre-training. Instead, we split the entire hierarchy into subtrees. Then these subtrees can be used in a mini-batch fashion for pre-training during the first HBGL stage. Each subtree contains as many nodes as the max length of our ALBERT model, which is 512 tokens. We also lower bound the size of the subtree to 256. This prevents us from sampling subtrees that are only leaves since most of the hierarchy nodes are leaf nodes. This also provides more variety in the subtree structures sampled.

For all methods, we only use the product documents during the fine-tuning tasks. This means that no additional hierarchical data is used by any method during fine-tuning. The only difference is the way the model is optimized given hierarchical data during pre-training. For methods like HPT and HBGL, this means we input the product documents and use the CLS embedding for fine-tuning classification, without any node inputs. For the document inputs, we concatenate the product's title, description, and any bullet point information provided. In the query-product dataset, we additionally concatenate the search query before the product document.

VI. Results

We test our downstream fine-tuning tasks given our pre-training baselines and proposed SST architecture. Immediately we see that the best performing methods are HBGL and SST in Table II, which shows relative improvements over baselines. SST improves *micro* metrics on average by 0.5% over HBGL and by 0.6% over the standard MLM pre-training method. Using SST we also see larger gains in *macro* scores (2.3% over HBGL and 3.8% over MLM), which indicates that SST generalizes better for long-tail distributed data. These are significant improvements since we replicate them across multiple datasets and in the practical deployment sense since we don't change the model architecture in fine-tuning. In addition, SST uses a

TABLE II

FINE-TUNING RESULTS GIVEN OUR PRE-TRAINING METHODS. WE REPORT BOTH THE MICRO AND MACRO F1 SCORES COMPARISON FOR PRODUCT CATEGORY (PC), QUERY-PRODUCT (QP), LOCAL FINANCIAL (LF), AND GLOBAL FINANCIAL (GF) DATASETS. THE SCORES ARE RELATIVE IMPROVEMENTS VERSUS THE STATED BASELINES IN THE TOP ROW. THE BEST RESULTS ARE IN BOLD AS WELL AS RESULTS WITHIN A MARGIN OF ERROR.

Pre-training	PC Micro F1	PC Macro F1	QP Micro F1	QP Macro F1	LF Micro F1	LF Macro F1	GF Micro F1	GF Macro F1
MLM only	0	0	**0**	0	0	0	0	0
HiMatch	-11.2%	-68.6%	-3.7%	-21.2%	-26.4%	-86.7%	-28.3%	-75.2%
HPT	+0.4%	+6%	-1.4%	-1.5%	+0.1%	+4.6%	-0.2%	-0.7%
HBGL	-0.5%	-0.5%	**0.1%**	-1.8%	**+0.8%**	**+7.9%**	-0.1%	**+0.4%**
SST	**+1.7%**	**+13.4%**	**0**	**0.3%**	+0.5%	1.9%	**+0.2%**	-0.2%

TABLE III

ABLATION RESULTS WITHOUT THE INPUT LEVEL PROJECTION OR THE MASKING STRATEGY FOR THE STRUCTURAL-TRANSFORMER.

Pre-training	PC Micro	PC Macro	QP Micro	QP Macro
SST	**0**	0	**0**	0
w.o. Mask	-0.4%	**+1.2%**	-0.8%	-3.1%
w.o. Proj	-0.4%	-3.6%	-0.3%	**+0.3%**

TABLE IV

TEST RESULTS COMPARISON FROM SIMPLER HIERARCHY PATH PREDICTION MODELS USING LINEAR CLASSIFIER HEADS.

Pre-training	PC Micro	PC Macro	QP Micro	QP Macro
Node Only	0	0	0	0
Multi-Label	-0.5%	+5.5%	-0.5%	-1.8%
Multi-Task	+0.5%	**+15%**	-0.1%	-0.2%
SST	**+1.3%**	+9.5%	**+0.1%**	**+0.2%**

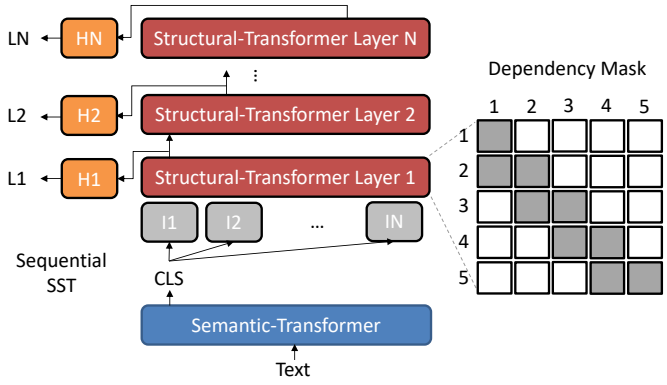

Fig. 4. Sequential variation of our SST where each hierarchy level is predicted after each Transformer layer. On the right, we show an alternative dependency mask strategy.

simple pre-training procedure and uses standard Transformer embeddings. In contrast, HBGL has two stages for hierarchy pre-training and uses a more complicated architecture, which increases overhead in production deployment.

We also observe the impact of not being able to train hierarchy alignment and fine-tune jointly, where methods like HiMatch suffer the most. This may be due to embedding distribution mismatches for GNN nodes and the ALBERT CLS representation, while other methods align the document to the hierarchy within the Transformer itself. While we see broader gains from our method, we aim to further understand which components lead to these gains and test alternative setups for the SST architecture.

A. SST Ablations

1) Masking and Projection: We start our ablations by observing the design choices around our Structural-Transformer. We first observe what happens if we don't add our attention masking. Second, we observe the results if we don't project the CLS embedding into different Transformer input levels. In this case, the same CLS embedding is used for all L inputs levels and Structural-Transformer would rely only on the position embedding to differentiate between the levels. We evaluate these variations on the product categorization and query-product datasets.

Looking at the results in Table III, we see that masking and projection are useful for SST. Not masking the inputs leads the model to learn spurious correlations between the levels, which degrades fine-tuning performance. The input projection provides an additional degree of freedom to differentiate the different hierarchy levels, which also provides an edge in fine-tuning.

2) Structural-Transformer Design: We further explore how to best use the CLS embedding output from our Semantic-Transformer for hierarchy pre-training. We first test if the Structural-Transformer architecture is needed by testing simpler linear classifier heads to predict the hierarchy nodes:

- Node Only: Fine-tune a single classifier head to predict the corresponding product hierarchy node $n_d^l \in P$ only.
- Multi-Label: Use a single head to predict all the nodes in the path P as a multi-label objective.
- Multi-Task: Use multiple classifier heads to predict the nodes at each level in the path.

These results are presented in Table IV and we observe that using the original SST setup works the best overall, while the multi-task method runs a close second. This demonstrates that the Structural-Transformer is a stronger hierarchical decoder for CLS embeddings, leading to better Semantic-Transformer pre-training.

To induce better biases in our hierarchical prediction, we test predicting each level of the path after each subsequent Transformer layer in our Structural-Transformer. Such an

TABLE V
COMPARISONS OF MODIFICATIONS ON TOP OF HTC ARCHITECTURES FOR FINE-TUNING (FT). WE DESCRIBE THE INPUTS USED IN THE FT INPUTS COLUMN, AS WELL AS THE CLASSIFIER HEAD USED GIVEN THE INPUTS IN FT MODEL.

Pre-training	FT Inputs	FT Model	PC Micro F1	PC Macro F1	QP Micro F1	QP Macro F1
HPT	Text	CLS Head	0	0	0	0
HPT	Text, Prompts, Mask Nodes	CLS Head	+0.3%	-2.8%	+0.4%	-0.8%
HPT	CLS, Mask Node Outputs	Transformer	+0.9%	+2.1%	+0.5%	-0.3%
HBGL	Text	CLS Head	-0.9%	-6.2%	**+1.5%**	-0.3%
HBGL	Text, Mask Nodes	CLS Head	+0.3%	+5.2%	+1.2%	+0.3%
HBGL	CLS, Mask Node Outputs	Transformer	+0.3%	-2.1%	+0.9%	0
SST	Text	CLS Head	**+1.3%**	**+7%**	**+1.4%**	**+1.8%**

TABLE VI
SEQUENTIAL SST MODEL RESULTS COMPARISON, WHERE THE DEPENDENCY (DEP) MASKING STRATEGY IS ALSO TESTED.

Pre-training	PC Micro	PC Macro	QP Micro	QP Macro
Seq. SST	0	0	**0**	0
+ Dep Mask	+0.3%	**+4.3%**	-0.3%	**+0.6%**
SST	**+0.8%**	-0.2%	**0**	**+0.8%**

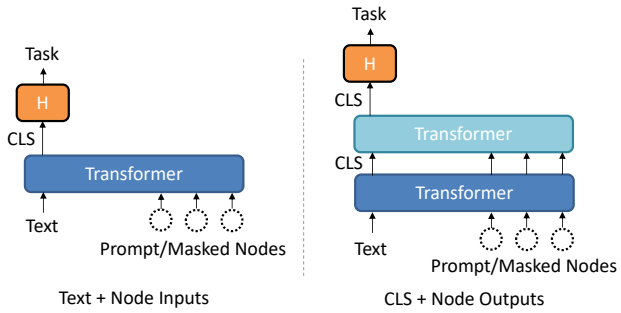

Fig. 5. Two variations of our HTC Transformer models for fine-tuning. On the left, we add the node inputs and allow attention between the nodes and the text. On the right, we use the CLS and node outputs from the original Transformer as inputs to the second Transformer.

architecture can mimic the hierarchy dependency where it forces deeper levels to directly utilize the embeddings from shallower levels, as shown in Figure 4.

In this setup, we also test another variation of the attention mask which we call dependency masking. For each input level, it only allows the model to use the hidden states from the *previous* input level as shown on the right of Figure 4. In combination with the sequential version of our SST model, it effectively forces the model to only use the previous layer's hidden states to predict the node corresponding to the current level, similar to a seq-to-seq decoding fashion [29]. In contrast, our standard attention mask will allow each layer to use *all* outputs from the previous layers.

Within the sequential results in Table VI, the original SST model still works better with a simpler architecture. This indicates that hierarchical decoding is too restrictive during pre-training. Adding the dependency mask better leverages the hierarchical model structure and provides comparable performance to SST.

B. HTC Fine-tuning Variants

In our experiments, we only input product documents and use the CLS embedding for our classifier heads. In HPT and HBGL they also input prompts and masked nodes respectively during pre-training. Therefore these node inputs can also provide extra when contextualizing the documents. We can test adding these prompts and masked nodes to the inputs as shown on the left of Figure 5. Additionally, these methods output the level-wise hidden states corresponding to the masked input nodes, which they use for hierarchy path classification during the pre-training. We can use these output hidden states as additional features during fine-tuning. In this setting, we add a second transformer on top of the original models, which takes in the document CLS output as well as the node outputs. Then the output of this second Transformer is used for fine-tuning as shown on the right of Figure 5.

From the results in Table V we see that for HPT, there are no significant improvements when using just adding additional inputs into the original model. However, when incorporating the node features into the secondary Transformer, the overall performance improves. This indicates that latent output node embedding contains useful hierarchical feature data, even when the ground truth hierarchical information is not present.

For the HBGL results, we see a much larger improvement when adding the masked nodes as inputs during fine-tuning. This may be related to the fact that the node representations are pre-trained within the same Transformer model, thus better contextualizing the CLS embedding. Using the second Transformer model to fine-tune on top of the base model reduces performance which indicates better contextualization with the document tokens in the base HBGL model itself. Regardless, our proposed SST approach provides the best fine-tuning performance over these HTC variations.

C. Leveraging Hierarchies During Fine-tuning

In our evaluation we fine-tune with product document inputs only, but how do our methods perform if the hierarchy information is available? This can be useful when there are downstream tasks that can incorporate this hierarchy data to further improve performance. To test this, we use the PC and QP datasets and link the hierarchy paths to the corresponding

TABLE VII
FINE-TUNING RESULTS COMPARISON WHEN WE ALSO INCLUDE THE HIERARCHY NODE NAME CORRESPONDING TO THE DOCUMENT.

Pre-training	FT Inputs	PC Micro F1	QP Micro F1
MLM only	Text	0	0
HBGL	Text	-0.5%	0
SST	Text	+1.7%	**+0.1%**
MLM only	Node, Text	+2.9%	-0.5%
HBGL	Node, Text	+3.3%	-0.4%
SST	Node, Text	**+3.7%**	**+0.1%**

TABLE VIII
HIERARCHY PREDICTION PERFORMANCE (MICRO F1) IMPROVEMENT OVER THE HPT BASELINE.

Pre-training	L1	L2	L3	L4	L5	L6
HPT	0	0	0	0	0	0
HBGL	+2.6%	**+4.5%**	**+46%**	**+46%**	**+52%**	**+24%**
Multi-Task	+2.6%	+1.2%	+22%	+10%	+9.3%	+2.3%
SST	+2.3%	+1.5%	+21%	+9.4%	+6.7%	+0.0%
Seq. SST	**+2.9%**	+1.8%	+22%	+10%	+8.0%	+1.1%
+ Dep Mask	+2.5%	+1.2%	+21%	+8.8%	+6.2%	-2.3%

products. Then the input during fine-tuning is the name of the hierarchy node concatenated to the original document inputs.

We can observe the results in Table VII, where all methods improve for PC but vary for QP results. For QP, this may result from not having the query and the node name present in pre-training, but appearing during fine-tuning, causing input distribution mismatch. From the overall results with the hierarchy, HBGL does better than the MLM baseline and SST does better than HBGL. This shows that hierarchy pre-training is still valuable even if the hierarchy inputs are added during fine-tuning.

D. Verification for Hierarchy Prediction

Our hierarchy pre-training is an intermediate step before fine-tuning, but we don't know how well this intermediate hierarchy prediction performs. Therefore we also test each method's performance in predicting the nodes at each level of the path. To accomplish this we create a separate pre-training test set with 166k product documents assigned to nodes in our hierarchy. We evaluate models that predict the entire path and compute the micro F1 score improvements over the HPT model baseline. Here we include the multi-task model from our previous testing, which uses the CLS embedding to train classifier heads for node prediction at each level. We only test down to $l = 6$ levels deep since the number of samples to level $L = 10$ is small, resulting in a large variance.

From the results in Table VIII, HBGL has the strongest performance. This is reasonable since these are HTC methods that optimize for hierarchy prediction itself. Furthermore, HBGL uses attention between the entire input document, which provides a more nuanced representation to derive the path from.

In comparison, our SST model only uses the compressed CLS output versus the entire document's tokens. Therefore it cannot pick out individual salient tokens to use for hierarchy path prediction. In doing so we enable a better alignment focused on downstream fine-tuning over pure hierarchy prediction performance. Interestingly even our CLS-based methods perform better than HPT for hierarchy prediction. This indicates the benefits of aligning hierarchies purely within Transformer models, rather than aligning Transformers to graph-based models.

VII. FUTURE WORK

a) E-commerce Extensions: Beyond product document information, we are also interested in how our proposed structural learning can incorporate other features. Given product images, we can also use a Structural-Transformer to pre-train image embeddings, where images in the same node hierarchy should visually similar. This can be used for improving image search or can augment the proposed document embeddings for downstream tasks. Another feature is the customer's query, where we can train a better query representation to narrow down the category of products they intend to search in a hierarchy.

b) Beyond E-commerce: We want to also explore the generalization of our training paradigm in broader research tasks. This can be done by leveraging hierarchies from Wikipedia [30], [31] to pre-train a general model. Whereas the current standard is to pre-train using MLM on a corpus such as Wikipedia, we can investigate if our method can improve the association between articles in neighboring nodes. This can improve common NLP tasks that involve few shot learning, relation extraction, or named entity recognition.

We can also explore adapting our pre-training to general knowledge graphs (KGs). Many tasks require domain-specific knowledge to obtain the correct inputs for a problem, where current methods embed KG inputs into the model [32]–[34]. Instead of explicitly requiring KG data during training, we can explore how to improve implicit KG representations during pre-training [35]–[37]. One extension is in the Structural-Transformer, where we can predict the document's associated KG node in the first level, and the nodes in its k-hop neighborhood in each of the next k levels. This opens up having a simple knowledge-aware Transformer that can be applied directly to KG-dependent fine-tuning tasks.

VIII. CONCLUSION

Hierarchical data is key to organizing information in a structured manner, such as a taxonomy of product texts in the e-commerce setting. We explore encoding these hierarchical structures implicitly into document representations. These hierarchy-aware document embeddings should be useful in downstream fine-tuning tasks which contain latent hierarchical structures, even when the explicit hierarchical information is not present. We propose Semantic and Structural Transformers that pre-train on available products linked to hierarchies, and then only use the base Semantic-Transformer to fine-tune on downstream tasks. We compare this approach to existing

hierarchy classification models and show that our model is the best approach for downstream fine-tuning.

REFERENCES

[1] D. D. Lewis, Y. Yang, T. Russell-Rose, and F. Li, "Rcv1: A new benchmark collection for text categorization research," *Journal of machine learning research*, vol. 5, no. Apr, pp. 361–397, 2004.

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

[3] I. Chalkidis, M. Fergadiotis, P. Malakasiotis, and I. Androutsopoulos, "Large-scale multi-label text classification on eu legislation," *arXiv preprint arXiv:1906.02192*, 2019.

[4] E. Sandhaus, "The new york times annotated corpus," *Linguistic Data Consortium, Philadelphia*, vol. 6, no. 12, p. e26752, 2008.

[5] Y. H. Cho and J. K. Kim, "Application of web usage mining and product taxonomy to collaborative recommendations in e-commerce," *Expert systems with Applications*, vol. 26, no. 2, pp. 233–246, 2004.

[6] Y. S. Kim, "Recommender system based on product taxonomy in e-commerce sites." *Journal of Information Science & Engineering*, vol. 29, no. 1, 2013.

[7] M. Skinner and S. Kallumadi, "E-commerce query classification using product taxonomy mapping: A transfer learning approach." in *eCOM@ SIGIR*, 2019.

[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[9] D. Shen, J.-T. Sun, Q. Yang, and Z. Chen, "Building bridges for web query classification," in *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, 2006, pp. 131–138.

[10] H. Cao, D. H. Hu, D. Shen, D. Jiang, J.-T. Sun, E. Chen, and Q. Yang, "Context-aware query classification," in *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, 2009, pp. 3–10.

[11] T. Zahavy, A. Magnani, A. Krishnan, and S. Mannor, "Is a picture worth a thousand words? a deep multi-modal fusion architecture for product classification in e-commerce," *arXiv preprint arXiv:1611.09534*, 2016.

[12] Â. Cardoso, F. Daolio, and S. Vargas, "Product characterisation towards personalisation: learning attributes from unstructured data to recommend fashion products," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 80–89.

[13] V. Gupta, H. Karnick, A. Bansal, and P. Jhala, "Product classification in e-commerce using distributional semantics," *arXiv preprint arXiv:1606.06083*, 2016.

[14] Z. Kozareva, "Everyone likes shopping! multi-class product categorization for e-commerce," in *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2015, pp. 1329–1333.

[15] P. Ristoski, P. Petrovski, P. Mika, and H. Paulheim, "A machine learning approach for product matching and categorization," *Semantic web*, vol. 9, no. 5, pp. 707–728, 2018.

[16] H. Xu, B. Liu, L. Shu, and P. Yu, "Open-world learning and application to product classification," in *The World Wide Web Conference*, 2019, pp. 3413–3419.

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

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

[19] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," *arXiv preprint arXiv:1909.11942*, 2019.

[20] J. Zhou, C. Ma, D. Long, G. Xu, N. Ding, H. Zhang, P. Xie, and G. Liu, "Hierarchy-aware global model for hierarchical text classification," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020, pp. 1106–1117.

[21] H. Chen, Q. Ma, Z. Lin, and J. Yan, "Hierarchy-aware label semantics matching network for hierarchical text classification," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 2021, pp. 4370–4379.

[22] Z. Wang, P. Wang, L. Huang, X. Sun, and H. Wang, "Incorporating hierarchy into text encoder: a contrastive learning approach for hierarchical text classification," *arXiv preprint arXiv:2203.03825*, 2022.

[23] C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T.-Y. Liu, "Do transformers really perform badly for graph representation?" *Advances in Neural Information Processing Systems*, vol. 34, pp. 28 877–28 888, 2021.

[24] Z. Wang, P. Wang, T. Liu, Y. Cao, Z. Sui, and H. Wang, "Hpt: Hierarchy-aware prompt tuning for hierarchical text classification," *arXiv preprint arXiv:2204.13413*, 2022.

[25] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[26] T. Jiang, D. Wang, L. Sun, Z. Chen, F. Zhuang, and Q. Yang, "Exploiting global and local hierarchies for hierarchical text classification," *arXiv preprint arXiv:2205.02613*, 2022.

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

[28] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. v. d. Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *European semantic web conference*. Springer, 2018, pp. 593–607.

[29] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Advances in neural information processing systems*, vol. 27, 2014.

[30] L. Muchnik, R. Itzhack, S. Solomon, and Y. Louzoun, "Self-emergence of knowledge trees: Extraction of the wikipedia hierarchies," *Physical review E*, vol. 76, no. 1, p. 016106, 2007.

[31] T. Flati, D. Vannella, T. Pasini, and R. Navigli, "Two is bigger (and better) than one: the wikipedia bitaxonomy project," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2014, pp. 945–955.

[32] W. Liu, P. Zhou, Z. Zhao, Z. Wang, Q. Ju, H. Deng, and P. Wang, "K-bert: Enabling language representation with knowledge graph," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 03, 2020, pp. 2901–2908.

[33] M. E. Peters, M. Neumann, R. L. Logan IV, R. Schwartz, V. Joshi, S. Singh, and N. A. Smith, "Knowledge enhanced contextual word representations," *arXiv preprint arXiv:1909.04164*, 2019.

[34] Z. Zhang, X. Han, Z. Liu, X. Jiang, M. Sun, and Q. Liu, "Ernie: Enhanced language representation with informative entities," *arXiv preprint arXiv:1905.07129*, 2019.

[35] X. Wang, T. Gao, Z. Zhu, Z. Zhang, Z. Liu, J. Li, and J. Tang, "Kepler: A unified model for knowledge embedding and pre-trained language representation," *Transactions of the Association for Computational Linguistics*, vol. 9, pp. 176–194, 2021.

[36] Y. Sun, S. Wang, Y. Li, S. Feng, X. Chen, H. Zhang, X. Tian, D. Zhu, H. Tian, and H. Wu, "Ernie: Enhanced representation through knowledge integration," *arXiv preprint arXiv:1904.09223*, 2019.

[37] Y. Sun, S. Wang, S. Feng, S. Ding, C. Pang, J. Shang, J. Liu, X. Chen, Y. Zhao, Y. Lu *et al.*, "Ernie 3.0: Large-scale knowledge enhanced pre-training for language understanding and generation," *arXiv preprint arXiv:2107.02137*, 2021.