An AST Structure Enhanced Decoder for Code Generation

Hui Jiang, Linfeng Song, Yubin Ge, Fandong Meng, Junfeng Yao, and Jinsong Su

***Abstract*—Currently, the most dominant neural code generation modelsare often equipped with a tree-structured LSTM decoder, which outputs a sequence of actions to construct an Abstract Syntax Tree (AST) via pre-order traversal. However, such a decoder has two obvious drawbacks. First, except for the parent action, other faraway and important history actions rarely contribute to the current decision. Second, it also neglects future actions, which may be crucial for the prediction of the current action. To deal with these issues, in this paper, we propose a novel AST structure enhanced decoder for code generation, which significantly extends the decoder with respect to the above two aspects. First, we introduce an AST information enhanced attention mechanism to fully exploit history actions, of which impacts are further distinguished according to their syntactic distances, action types and relative positions; Second, we jointly model the predictions of current action and its important future action via multi-task learning, where the learned hidden state of the latter can be further leveraged to improve the former. Experimental results on commonly-used datasets demonstrate the effectiveness of our proposed decoder.**[1]

***Index Terms*—Code generation, abstract syntax tree, attention mechanism, future action prediction.**

I. Introduction

CODE generation is an important application of conditional text generation in the community of software engineering, which still remains a great challenge. Given a natural language (NL) description, it aims to automatically output an executable code. Because of its potential in improving the working efficiency of programmers, code generation has attracted great attention recently and become one of the popular research topics in conditional text generation.

To build effective code generation models, many previous efforts have been initiated on exploring hand-engineered features that are highly tailored to specific programming languages [1]–[3]. Inspired by the successful applications of deep learning in natural language processing tasks, such as machine translation [4], researchers have turned to exploring neural models for code generation [5]–[17]. Among these studies, the most typical models consider code generation as a sequence-to-Abstract Syntax Tree (Seq2AST) modeling problem [7], [8], [10], [13], [14]. Specifically, an encoder is first used to learn the semantic representations of input NL descriptions. Then, under the semantic guidance of the encoder, a decoder outputs a sequence of grammar actions, with the corresponding AST it can be generated through pre-order traversal. Finally, deterministic generation tools are applied to convert the generated AST into surface codes. Compared with previous models, the utilization of AST not only shrinks search space but also naturally reflects the recursive structure of programming languages, leading to the generation of well-form code.

Despite of these benefits, utilizing AST introduces a tree generation problem that is generally more difficult than standard sequence prediction. Thus, how to effectively leverage rich features of the partially generated AST is important to ensure accurate predictions of subsequent actions. However, this may be beyond the capability of a standard RNN decoder. First, except for the parent action, other faraway and important history actions can greatly affect the current action prediction. Second, standard RNNs are unable to exploit important future actions, such as, the right-side sibling actions, which is often crucial for the prediction of current action as well. Therefore, it is worth exploring the syntactic information of partial AST used in these Seq2AST models.

In this paper, we propose an AST structure enhanced decoder to effectively exploit rich features of partial AST for action predictions. It is a significant extension of the conventional tree-structured decoder [10], with two aspects of improvements. First, we equip the decoder with an AST information enhanced attention mechanism, which distinguishes impacts of different history actions on the prediction of current action according to their AST-based syntactic distances, action types, and relative positions. Second, we choose one future action that provides important context for the prediction of current action and jointly

Manuscript received May 28, 2021; revised October 17, 2021; accepted November 28, 2021. Date of publication December 28, 2021; date of current version January 21, 2022. This work was supported in part by the National Key Research and Development Program of China under Grant 2020AAA0108004, in part by the National Natural Science Foundation of China under Grant 61672440, in part by the Natural Science Foundation of Fujian Province of China under Grant 2020J06001, in part by the Youth Innovation Fund of Xiamen under Grant 3502Z20206059, and in part by the Fundamental Research Funds for the central universities under Grant 20720200077. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Eric Fosler-Lussier. *(Corresponding author: Jinsong Su.)*

Hui Jiang and Junfeng Yao are with the School of Informatics, Xiamen University, Xiamen, Fujian 361005, China (e-mail: hjiang@stu.xmu.edu.cn; yao0010@xmu.edu.cn).

Jinsong Su is with the School of Informatics, Xiamen University, Xiamen, Fujian 361005, China, and also with the Pengcheng Lab, Shenzhen, Guangdong Province 518066, China (e-mail: jssu@xmu.edu.cn).

Linfeng Song is with Tencent Company, Shenzhen 518000, China (e-mail: freesunshine0316@gmail.com).

Yubin Ge is with the University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA (e-mail: yug37@pitt.edu).

Fandong Meng is with Tecent Company, Shenzhen, Beijing 100080, China (e-mail: fandongmeng@tencent.com).

Digital Object Identifier 10.1109/TASLP.2021.3138717

[1]We release our code at https://github.com/DeepLearnXMU/CG-ASED.

model the predictions of current action and this future action via multi-task learning, where the learned hidden state of the latter can be then leveraged to benefit the former. It should be noted that the future actions are predicted on the fly during the inference. Compared with the conventional decoder, ours is able to better exploit both beneficial history and future actions for the prediction of current action.

Overall, major contributions of our work are three-fold:

1) We study how to better exploit history actions for predicting actions via an AST information enhanced attention.
2) We propose to make use of the prediction of future action to benefit predicting current action. This is in contrast with most decoding process where causal decoding is normally adopted.
3) Experimental results on commonly-used datasets demonstrate the effectiveness of our proposed decoder.

II. Related Work

With the rise of deep learning, neural network based models have gradually become prevalent in code generation. Typically, Ling *et al.* [5] treated code generation as a sequence-to-sequence modeling task. However, this model does not consider the fact that code has to be well-defined programs in the target syntax. To address this issue, Yin and Neubig [7] proposed a neural Seq2AST model to generate tree-structured meaning representations (MRs) using a series of tree-construction actions. Rabinovich *et al.* [8] explored abstract syntax networks, where domain-specific MRs are represented by ASTs specified under the abstract syntax description language framework. Besides, Yin and Neubig [10] applied a transition-based method to generate ASTs. Unlike the above work, Yin and Neubig [12] adopted reranking to rescore an N-best list of predicted MRs using complicated features. And Xu *et al.* [17] explored how to exploit external knowledge for neural code generation via pre-training. Inspired by the studies [18]–[21] exploiting future information to refine conditional text generation models with autoregressive decoder, Xie *et al.* [22] introduced a mutual learning framework to jointly train two Seq2Tree models with different traversals (pre-order traversal vs. breath-first traversal) based decodings, both of which are expected to benefit each other. Moreover, recently, Jiang *et al.* [23] proposed to equip the Seq2Tree model with a context-based Branch Selector, which is able to dynamically determine optimal expansion orders of branches for multi-branch nodes.

In this work, we mainly focus on exploiting the AST information for code generation. In this regard, Sun *et al.* [13] proposed a grammar-based structural CNN decoder. Then, Sun *et al.* [14] adapted Transformer into code generation, with an additional CNN layer to handle AST structure. Nevertheless, they not only ignore the fact that impacts of different history actions on the prediction of current action vary significantly, but also neglect the context contained in future actions, which may be crucial for the prediction of current action. By contrast, our proposed decoder is able to solve these two defects, leading to better performance of neural models for code generation.

III. Our Model

In this section, we give a detailed description to the code generation model with our proposed decoder. Our model is based on TRANX [10], that has been widely used due to its competitive performance [12], [17]. Note that our strategies can also be applied to other Seq2AST models.

In the following subsections, we first describe how to model code generation using abstract syntax description language (ASDL) grammars, which are the bases of both TRANX and our model, then illustrate the encoder-decoder framework of our model, in details.

A. Modeling Code Generation Using ASDL Grammar

As implemented in TRANX [10], we adopt a three-step strategy to model the NL description-to-code conversion, where ASTs are introduced as intermediate representations. As shown in Fig. 1, for each input NL description $\boldsymbol{x}$, we first apply our model to produce a series of ASDL grammar based actions $\boldsymbol{a} = a_1, a_2, \ldots, a_T$, which then construct an AST $\boldsymbol{z}$ in the manner of pre-order traversal. Finally, we directly calls the user specified function AST_to_MR($*$) to convert $\boldsymbol{z}$ into the target code $\boldsymbol{y}$.

Formally, each ASDL grammar consists of two components: *type* and *constructors*, where the type is defined as *composite* or *primitive*, and each constructor specifies a language component of a particular type using its *fields*. Here, each field specifies the type of value it can hold and contains a cardinality indicating the number of values it holds. In each field with composite type, there exist several constructors under its type. In constrast, the field with primitive type directly stores value. Back to Fig. 1, in the ASDL grammar "*expr→Tuple(expr* elts)*" of a_2, the leftmost composite type "*expr*" indicates that its constructor "*Tuple(expr* elts)*" can only be applied to form an AST node with type "*expr*". When specific to this constructor, the field "*elts*" with the type "*expr*" and the cardinality "*," will be further expanded to form its child AST nodes with type "*expr*". In addition, the ASDL grammar "*expr→Name(identifier id)*" of a_3 indicates the type of the field "*id*" is primitive, where we use a_4: "*GenToken[result]*" to form an AST node, directly storing the field "*id*" with the value "*result*".

Following Yin and Neubig [10], at each time step, we consider three kinds of actions based on ASDL grammar to construct an AST node:

- *ApplyConstr*[c]. Using this action, a constructor c is applied to the opening composite field of the parent node with the same type as c, populating a node using the fields of c. Here the composite field of the parent node is also called as *frontier field*;
- *Reduce*. It denotes the completion of expanding a field with optional or multiple cardinalities;
- *GenToken*[v]. This action populates a primitive frontier field with a token v.

Formally, the generation probability of the code $\boldsymbol{y}$ can be factorized as the probabilities of the action sequence $\boldsymbol{a}$ corresponding to the AST $\boldsymbol{z}$ of $\boldsymbol{y}$: $p(\boldsymbol{z}|\boldsymbol{x}) = \prod_{t=1}^{T} p(a_t|a_{<t}, \boldsymbol{x})$, where a_t is the t-th action, and $a_{<t}$ is the sequence of actions before

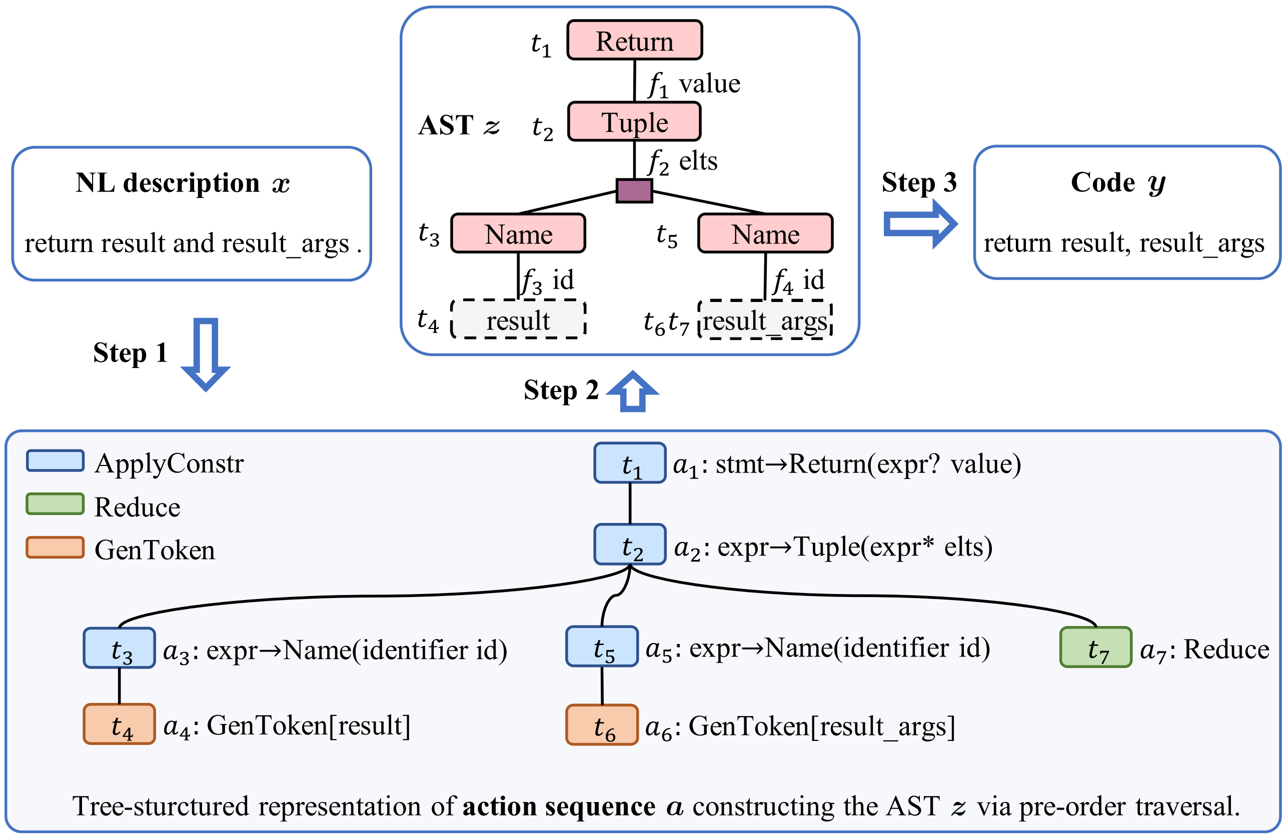

NL description x
return result and result_args .
AST z
t_1 Return
f_1 value
t_2 Tuple
f_2 elts
t_3 Name
t_5 Name
f_3 id
f_4 id
t_4 result
$t_6 t_7$ result_args
Step 1
Step 2
Step 3
Code y
return result, result_args
ApplyConstr
Reduce
GenToken
t_1 a_1: stmt→Return(expr? value)
t_2 a_2: expr→Tuple(expr* elts)
t_3 a_3: expr→Name(identifier id)
t_5 a_5: expr→Name(identifier id)
t_7 a_7: Reduce
t_4 a_4: GenToken[result]
t_6 a_6: GenToken[result_args]
Tree-sturctured representation of action sequence a constructing the AST z via pre-order traversal.

Fig. 1. An example of converting an input NL description into code.

the time step t and we parameterize the prediction probability $p(\boldsymbol{z}|\boldsymbol{x})$ using a neural encoder-decoder network.

B. The Framework of Our Model

Our encoder is same as that of TRANX. For an NL description $\boldsymbol{x} = x_1, x_2, \ldots, x_N$, we employ a BiLSTM encoder to learn word-level hidden states $\{\mathbf{h}_i\}$ where a forward LSTM reads the NL description from left to right while a backward LSTM operates in an opposite direction,

$$\overrightarrow{\mathbf{h}}_i = \text{LSTM}\left(\overrightarrow{\mathbf{h}}_{i-1}, \mathbf{E}(x_i)\right), \tag{1}$$

$$\overleftarrow{\mathbf{h}}_i = \text{LSTM}\left(\overleftarrow{\mathbf{h}}_{i+1}, \mathbf{E}(x_i)\right), \tag{2}$$

where $\mathbf{E}(x_i) \in \mathbb{R}^{d_w}$ is the embedding of the NL word x_i, and $\overrightarrow{\mathbf{h}}_i$, $\overleftarrow{\mathbf{h}}_i \in \mathbf{R}^{d_w}$ denote the hidden states generated in two directions, respectively. Finally, the word-level representations are obtained by concatenating the forward and backward hidden states, $\mathbf{h}_i = [\overrightarrow{\mathbf{h}}_i : \overleftarrow{\mathbf{h}}_i]$, where $1 \leqslant i \leqslant N$, $[:]$ denotes the operation of vector concatenation.

As mentioned above, our decoder is an enhanced version of the TRANX decoder, which is also an LSTM network. At each time step t, its hidden state is given by

$$\mathbf{s}_t = f_{\text{LSTM}}\left([\mathbf{E}(a_{t-1}) : \tilde{\mathbf{s}}_{t-1} : \mathbf{p}_t], \mathbf{s}_{t-1}\right), \tag{3}$$

where $\mathbf{E}(a_{t-1})$ is the embedding of the previous action a_{t-1}, and $\mathbf{p}_t$ is a concatenation of two vectors: the embedding of the frontier field and the decoder hidden state for the parent action. Besides, the temporary hidden state $\tilde{\mathbf{s}}_t$ is defined as

$$\tilde{\mathbf{s}}_t = \tanh\left(\mathbf{W}_1 \left[\mathbf{s}_t : \mathbf{c}_t : \mathbf{hc}_t : \mathbf{fc}_t\right]\right), \tag{4}$$

where $\mathbf{c}_t$ is the attentional context vector induced from the encoder hidden states $\{\mathbf{h}_i\}_{i=1}^N$, $\mathbf{hc}_t$ represents the context information from history actions through our attention mechanism, $\mathbf{fc}_t$ denotes the context induced from our prediction module for future action, and $\mathbf{W}_1$ is a parameter matrix.[2]

Next, as implemented in TRANX, we calculate the probability of action a_t according to the type of its frontier field:

- *Composite*. We adopt an APPLYCONSTR action to expand the field or a REDUCE action to close the field. Formally, we define the probability of using APPLYCONSTR$[c]$/REDUCE action as

$$\begin{aligned} & p\left(a_t = \text{APPLYCONSTR}[c]/\text{REDUCE} | a_{<t}, \boldsymbol{x}\right) \\ & = \text{softmax}\left(\mathbf{E}(c)^\top \mathbf{W}_2 \tilde{\mathbf{s}}_t\right). \end{aligned} \tag{5}$$

  Here, $\mathbf{E}(c)$ denotes the embedding of the constructor c.
- *Primitive*. We apply a GENTOKEN action to produce a token v, which is either generated from the vocabulary or copied from the input NL description. Thus, the probability of

[2]In this paper, all $\mathbf{W}_*$ denote parameter matrices.

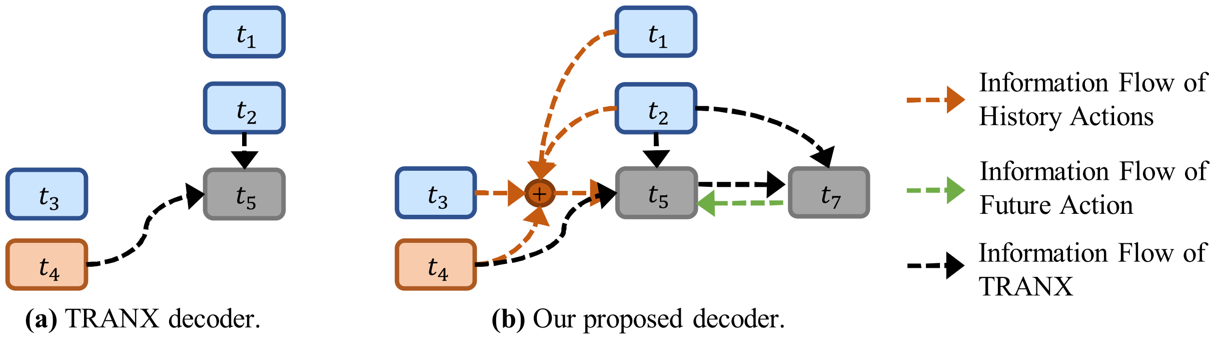

t_1
t_2
t_3
t_4
t_5
(a) TRANX decoder.
t_1
t_2
t_3
t_4
t_5
t_7
(b) Our proposed decoder.
Information Flow of History Actions
Information Flow of Future Action
Information Flow of TRANX

Fig. 2. Information flow of the TRANX decoder and our proposed decoder at the time step t_5.

using GenToken[v] is given by

$$\begin{aligned} & p\left(a_t = \text{GenToken}[v] | a_{<t}, \boldsymbol{x}\right) \\ = \; & p\left(\text{gen}\, | a_{<t}, \boldsymbol{x}\right) p_{gen}\left(v | a_{<t}, \boldsymbol{x}\right) + \\ & (1 - p\left(\text{gen}\, | a_{<t}, \boldsymbol{x}\right)) p_{copy}\left(v | a_{<t}, \boldsymbol{x}\right), \end{aligned} \tag{6}$$

$$\begin{aligned} \text{where } p\left(\text{gen}\, | a_{<t}, \boldsymbol{x}\right) &= \text{sigmoid}\left(\mathbf{W}_3 \tilde{\mathbf{s}}_t\right), \\ p_{gen}(v | a_{<t}, \boldsymbol{x}) &= \text{softmax}\left(\mathbf{E}(v)^\top \mathbf{W}_4 \tilde{\mathbf{s}}_t\right), \\ p_{copy}(v = x_i | a_{<t}, \boldsymbol{x}) &= \text{softmax}\left(\mathbf{h}_i^\top \mathbf{W}_5 \tilde{\mathbf{s}}_t\right). \end{aligned}$$

We will show in detail about how to get $\mathbf{hc}_t$ and $\mathbf{fc}_t$ in the following subsections.

1) AST Information Enhanced Attention Mechanism: Different from the conventional TRANX decoder that only considers parent and previous actions, our decoder is equipped with an attention mechanism to fully exploit all history actions. As shown in Fig. 2, when predicting the action at the time step t_5, the TRANX decoder only focuses on the history actions at t_2 and t_4. By contrast, our decoder is able to leverage history actions from t_1 to t_4.

Extend from the standard attention mechanism, our attention effectively distinguishes impacts of history actions on the prediction of current action from the following aspects:

- *Syntactic distance* d_{tj}. Intuitively, the impact of the history action a_j mainly depends on their syntactic distance d_{tj} on AST. By considering syntactic distance, we expect that our proposed attention mechanism is able to better qualify the effects of different history actions.
- *Action type* p_j. Intuitively, if the history action a_j is an ApplyConstr or Reduce action, it contains the structural information of AST. Otherwise, it mainly contains semantic details of AST. For instance, in Fig. 1, the action a_3: "*ApplyConstr[expr→Name(identifier id)]*" indicates that it has only one child action, while the action a_4: "*GenToken[result]*" means that a specific word will be generated here. Through above analysis, we refine our proposed attention mechanism by introducing two types of labels to further distinguish the impacts of history actions: ApplyConstr/Reduce and GenToken, where Reduce action is considered as a special ApplyConstr action.
- *Relative position* r_{tj}. The motivation behind stems from the fact that if the history action a_j is an ancestor or sibling one, it often possesses greater impact on the prediction of current action. To this end, in our attention mechanism, we classify history actions into two categories: closely-related actions and the other actions. Back to Fig. 1, the closely-related actions of a_5 mainly include a_1, a_2, and a_3.

Formally, we induce the context vector $\mathbf{hc}_t$ from history actions as follows:

$$\mathbf{hc}_t = \sum_{j=1}^{t-1} \frac{\exp\left(e_{tj}\right) \cdot \mathrm{M}_{tj}}{\sum_{j'=1}^{t-1} \exp\left(e_{tj'}\right) \cdot \mathrm{M}_{tj'}} \mathbf{s}_j, \tag{7}$$

$$e_{tj} = \left(\mathbf{W}_7 \mathbf{s}_j + \mathbf{E}(p_j) + \mathbf{E}(r_{tj}) + \mathbf{E}(d_{tj})\right) \mathbf{s}_t^\top, \tag{8}$$

where $\mathbf{s}_j$ is the j-th decoder hidden state, and M_{tj} is a non-increasing function converting the the syntactic distance d_{tj} into a value in [0, 1]. More specifically, M_{tj} is defined as

$$\mathrm{M}_{tj} = \min\left[\max\left[\frac{1}{m}(m + z_t - d_{tj}), 0\right], 1\right], \tag{9}$$

$$z_t = S_m \sigma\left(\mathbf{v}^T \mathbf{s}_t + b\right), \tag{10}$$

where m is a hyper-parameter controlling its softness and S_m is the maximal scope. Note that our soft masking function is inspired by Sainbayar *et al.* [24]. By doing so, we can dynamically control the scope of attention operation to reduce the negative effect of noisy context. For example, the scope of GenToken action does not require the context of large span, so the z_t of GenToken action is usually small. By contrast, the z_t of ApplyConstr is often large, since it requires structural information. We verify this hypothesis in the case study of the subsequent experiment section.

2) Prediction Module for Important Future Action: As mentioned above, the TRANX decoder ignores future actions that may be crucial for the predication of current action. As shown in Fig. 2(a), for the action prediction at t_5, if the decoder can predict in advance its right-side sibling action a_7 is a Reduce action that closes the frontier field, it is likely to be more confident that the current action should be an ApplyConstr action. To model this, we jointly model the predictions of the current action and its important future action via multi-task learning, and then leverage the learned hidden state for predicting important future action to refine the prediction of current action, as depicted in Fig. 2(b).

In this case, one key question is how to identify the important future action a_t^f at the t-th time step. Here, we select a_t^f according to the following criterions. For the predicted action at each time step, if it has a right-side sibling action, we set this sibling action as its important future action. Otherwise, we set its important future action the same as that of its parent action. In Fig. 1, the important future action of a_5 is a_7, while that of a_6 is also a_7 since it has no right-side sibling action. The only two exceptions include:

1) The action a_1 working at the root node of AST.
2) REDUCE action used to close the frontier field with multiple cardinality.

Both of their important future actions are represented with a special tag $\langle \text{none} \rangle$.

Then, we predict the important future action a_t^f as follows:

$$\mathbf{fs}_t = \tanh\left(\mathbf{W}_8\left[\mathbf{s}_t : \mathbf{fp}_t\right]\right), \tag{11}$$

$$\tilde{\mathbf{fs}}_t = \tanh\left(\mathbf{W}_9\left[\mathbf{c}'_t : \mathbf{fs}_t\right]\right), \tag{12}$$

where $\mathbf{fp}_t$ encodes the frontier field information of the future action and is also a concatenation of two vectors like $\mathbf{p}_t$ (See (3)), and the $\mathbf{c}'_t$ is an attention context vector. Finally, we use $\tilde{\mathbf{fs}}_t$ to predict the important future action, of which definitions are similar to (5) and (6).

Furthermore, we leverage $\tilde{\mathbf{fs}}_t$ to refine the prediction of current action. Concretely, we employ an element-wise gating operation $\odot$ to produce the vector representation $\mathbf{fc}_t$ of the important future action:

$$\mathbf{g}_t = \text{sigmoid}\left(\mathbf{W}_{10}\left[\tilde{\mathbf{fs}}_t : \mathbf{s}_t\right]\right), \tag{13}$$

$$\mathbf{fc}_t = \tanh\left(\mathbf{W}_{11}(\mathbf{g}_t \odot \tilde{\mathbf{fs}}_t)\right), \tag{14}$$

where $\mathbf{fc}_t$ will be exploited as described in (4).

C. Model Training

Given the training dataset $D = \{(\boldsymbol{x}, \boldsymbol{a})\}$, We train our model using the following objective function:

$$J(D;\theta) = \sum_{(\boldsymbol{x},\boldsymbol{a})\in D} \frac{1}{T}\sum_{t=1}^{T}\big(\log p(a_t|a_{<t}, \boldsymbol{x}) + \lambda \log p(a_t^f|a_{<t}, \boldsymbol{x})\big), \tag{15}$$

where a_t^f denotes the important future action at the t-th time step, and λ is a hyper-parameter that balances the preference between two items of the loss function.

IV. EXPERIMENTS

A. Setup

We conduct experiments using four benchmark datasets:

1) DJANGO [25] totally contains 18,805 lines of Python source code, which are extracted from the Django Web framework, with each line paired with an NL description. In this dataset, the code mainly covers various real-world use cases of Python, including I/O operation, string manipulation, exception handling, etc.
2) CONALA [26] consists of 2,879 examples of manually annotated NL questions and their Python solution on STACK OVERFLOW. Compared with DJANGO, examples in CONALA cover real-world NL queries issued by programmers with diverse intents, and therefore are significantly more difficult due to its broad coverage and high compositionality of target meaning representations.
3) ATIS has been widely used in semantic parsing, it is a set of 5,410 inquiries of flight information.
4) GEO consists of a collection of 880 U.S. geographical questions. In each example of this dataset, its input is a NL description and output is a short piece of code in lambda logical forms like ATIS.

TABLE I
STATISTICS OF OUR EXPERIMENTAL DATASETS

Statistics	DJANGO	CONALA	ATIS	GEO
# Train	16,000	2,179	4,434	600
# Dev	1,000	200	491	–
# Test	1,805	500	448	280

We employ NLT[3] to tokenize all NL descriptions. In particular, we perform simple canonicalization for DJANGO, such as replacing input quoted strings with placeholders. Table I gives the detailed statistics of our datasets.

In addition to TRANX, we compare our model (**ASED**) with the following baselines:

- *SEQ2TREE* [6]. It is the first Seq2Tree model in code generation, which generates logical forms by conditioning the output trees on the encoded vectors of input NL utterances.
- *LPN* [5]. Typically, this model allows both the choice of conditioning context and granularity of generation, for example, characters or tokens.
- *ASN* [8]. It employs a modular architecture, generating an AST using specifically designed neural networks for each construct in the ASDL grammar.
- *YN17* [7]. It generates AST using a series of tree-construction actions based on the task-specific context free grammar.
- *COARSE2FINE* [9]. This model first generates a rough sketch of its meaning, then fills in missing details by taking the NL input and the sketch itself into account.
- *SZM19* [13]. This model contains several CNN modules, including the tree-based convolution and pre-order convolution, to predict the grammar rules of the programming language.
- *TREEGEN* [14]. It introduces the attention mechanism of Transformer [27], and a novel AST reader to incorporate grammar rules and AST structures into the network.

We use the same configure as TRANX [10]. Concretely, we set the size of action and field embeddings to 128. The dimension of hidden states is 256. We set the dropout probabilities of all

[3]https://github.com/nltk/nltk

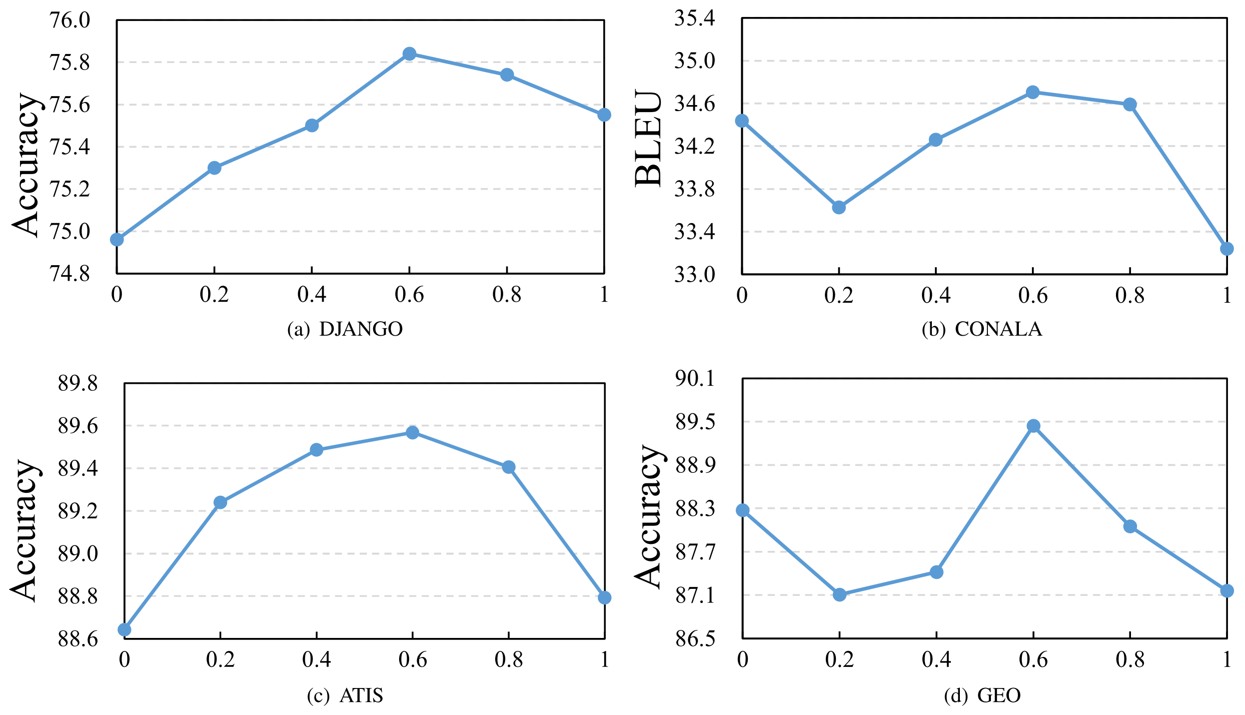


Fig. 3. Effect of λ (X-axis) on the validation sets.

TABLE II
PERFORMANCE OF OUR MODEL IN COMPARISON WITH VARIOUS BASELINES. † INDICATES PREVIOUSLY REPORTED SCORES. HERE WE REPORT THE MEAN AND STANDARD DEVIATION OVER FIVE INDEPENDENT RUNS

Model	**DJANGO** Acc.	**CONALA** BLEU / Acc.	**ATIS** Acc.	**GEO** Acc.
SEQ2TREE [6]†	–	–	84.6	86.1
LPN [5]†	62.3	–	–	–
ASN [8]†	–	–	87.1	85.9
YN17 [7]†	71.6	–	–	–
COARSE2FINE [9]†	–	–	87.7	88.2
SZM19 [13]†	–	–	85.0	–
TRANX [12]†	77.3 $\pm$0.4	24.35 $\pm$0.4 / 2.5 $\pm$0.7	87.6 $\pm$0.1	88.8 $\pm$1.0
TREEGEN [14]	–	–	88.0 $\pm$0.6	89.6
TRANX	77.2 $\pm$0.6	24.47 $\pm$0.6 / 2.3 $\pm$0.7	87.6 $\pm$0.7	88.7 $\pm$0.9
ASED	**79.2** $\pm$0.5	**25.56** $\pm$0.6 / **2.8** $\pm$0.7	**88.9** $\pm$0.7	**89.8** $\pm$1.1

datasets to 0.5, except for the probability of CONALA, which is 0.3. For decoding, we use a beam size 5 for GEO and ATIS, and 15 for DJANGO and CONALA. Following previous studies [10], [12]–[14], we use the exact matching accuracy as the metric for all datasets. Particularly, the corpus-level BLEU is used as a complementary metric for CONALA.

B. *Effect of* λ

As described in (15), λ is a crucial hyper-parameter that balances the preference between the conventional loss and future prediction loss. To investigate its effect, we conduct experiments using different λs on the validations sets. Fig. 3 shows the experimental results. Our model achieves the best performance with λ as 0.6 on all validation sets. Therefore, we use $\lambda = 0.6$ for our model in the following experiments.

C. *Main Results*

Table II provides the experimental results. Our reimplemented TRANX is comparable to that of its original paper, demonstrating that our baseline is competitive in performance. Overall, our proposed model significantly outperforms all baselines. Particularly, our model surpasses TRANX that is our most related baseline by 2.0, 1.09, 1.3 and 1.1 percents on four datasets. Even compared with the recently proposed model

TABLE III
ABLATION STUDY OF DIFFERENT COMPONENTS ON OUR MODEL

Model	DJANGO	ATIS
TRANX	77.2 ±0.6	87.6 ±0.7
ASED	**79.2** ±0.5	**88.9** ±0.7
ASED(AHA)	78.7 ±0.2	88.7 ±0.8
ASED(AHA-M)	78.2 ±0.5	88.4 ±0.7
ASED(AHA-SD)	78.3 ±0.4	88.2 ±0.9
ASED(AHA-AT)	78.3 ±0.1	88.4 ±0.9
ASED(AHA-RP)	77.9 ±0.3	88.1 ±0.7
ASED(PFA)	78.1 ±0.3	88.4 ±0.7
ASED(PFAML)	77.7 ±0.4	88.0 ±0.6

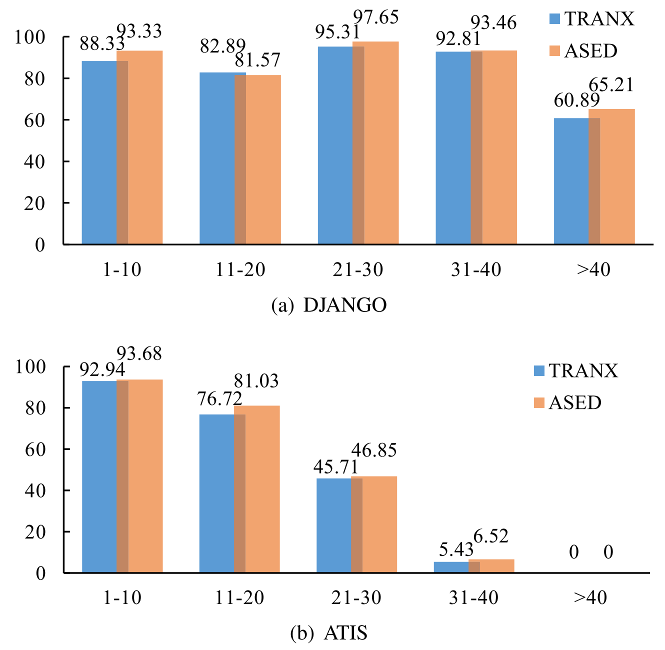


Fig. 4. Accuracy on different data groups divided according to the length of action sequence.

TREEGEN, our model also exhibits better performance. We also conduct 1,000 bootstrap tests and find that our model is significantly better than baseline with $p < 0.01$, $p < 0.05$, $p < 0.01$ and $p < 0.01$ on four datasets, respectively.

D. Performance by the Lengths of Action Sequences

Following Yin and Neubig [7], we individually split two relatively large datasets (DJANGO and ATIS) into different groups according to the lengths of action sequences, and show the average performance of models in Fig. 4(a) and (b), respectively. We notice that ASED always achieves better performance in most groups. These results further demonstrate the generality of our model.

E. Ablation Study

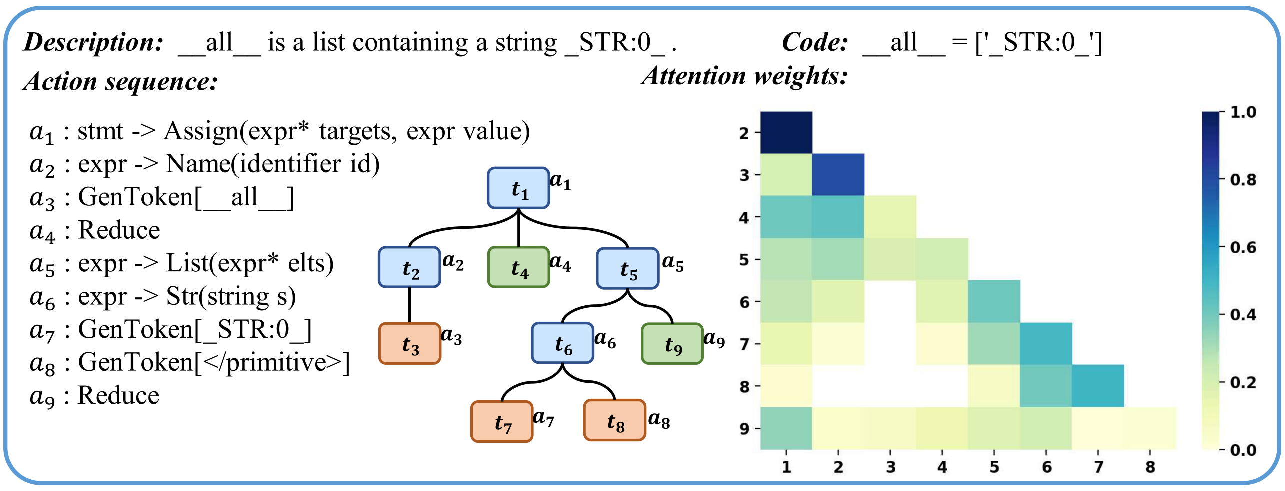
To investigate effects of different components on our model, we also provide the performance of the following ablated versions of our model on DJANGO and ATIS:

- *ASED(AHA)*. It is only equipped with our proposed **a**ttention mechanism acting on **h**istory **a**ctions. Furthermore, we investigate the performance of its four variants: **ASED(AHA-M)**, **ASED(AHA-SD)**, **ASED(AHA-AT)** and **ASED(AHA-RP)**, where M_{tj} function, **s**yntactic **d**istances, **a**ction **t**ypes and **r**elative **p**ositions of history actions are removed from our proposed attention mechanism in turn.
- *ASED(PFA)*. This simplified model only contains our proposed **p**rediction module for important **f**uture **a**ctions. It also has one variant: **ASED(PFAML)**, where $\tilde{\mathbf{fs}}_t$ is removed from (13).

From Table III , we find that removing any of our proposed modules leads to the performance degradation of our model. These results demonstrate the effectiveness of these components on the usage of AST structure information. Moreover, four variants of ASED(AHA) show further performance degradation, particularly for ASED(AHA-RP) which drops the most, showing that relative position is the most important feature in our proposed attention mechanism. Finally, ASED(PFAML) exhibits inferior performance to ASED(PFA), indicating that compared with the conventional multi-task learning, our approach is more effective to exploit future context.

F. Case Study

Figs. 5 and 6 show two examples of actions produced from DJANGO dataset. In the first example, we visualized the attention weights of our AST information enhanced attention mechanism. We can observe that our model always focuses more on the parent or sibling history actions for the action prediction at each time step. Particularly, some remote history actions are directly ignored under influence of our dynamic modeling of attention scope. In the second example, we compare the 1-best action sequences produced by different models. We find that



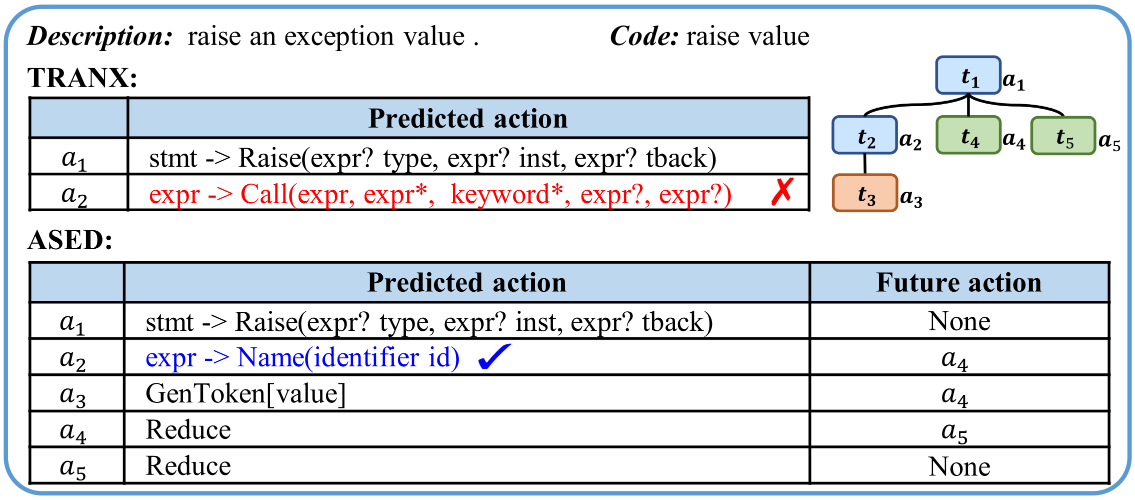
Description: __all__ is a list containing a string _STR:0_ .

Code: __all__ = ['_STR:0_']

Action sequence:

a_1 : stmt -> Assign(expr* targets, expr value)
a_2 : expr -> Name(identifier id)
a_3 : GenToken[__all__]
a_4 : Reduce
a_5 : expr -> List(expr* elts)
a_6 : expr -> Str(string s)
a_7 : GenToken[_STR:0_]
a_8 : GenToken[</primitive>]
a_9 : Reduce

Fig. 5. The example illustrating our proposed attention mechanism.



Description: raise an exception value .

Code: raise value

TRANX:

	Predicted action
a_1	stmt -> Raise(expr? type, expr? inst, expr? tback)
a_2	expr -> Call(expr, expr*, keyword*, expr?, expr?) ✗

ASED:

	Predicted action	Future action
a_1	stmt -> Raise(expr? type, expr? inst, expr? tback)	None
a_2	expr -> Name(identifier id) ✓	a_4
a_3	GenToken[value]	a_4
a_4	Reduce	a_5
a_5	Reduce	None

t_1 a_1
t_2 a_2
t_4 a_4
t_5 a_5
t_3 a_3

Fig. 6. The example of our module for predicting future action.

TRANX makes an incorrect action prediction at the 2-nd time step, while our model produces correct action. This is because TRANX correctly predicts the action at the 4-th time step firstly, which helps model to predict the value in field "*type*".

V. Conclusion

In this work, we have proposed an AST structure enhanced decoder for code generation, which significantly extends the TRANX decoder in two aspects. First, we introduce an AST information enhanced attention to fully exploit history actions. Second, via multi-task learning, we equip our decoder with a module predicting important future action, which enables our decoder to leverage future action during the prediction of current action. Compared with previous models, our enhanced model is able to better exploit history and future actions for code generation. Experimental results on several datasets verify the effectiveness of our decoder.

In the future, we plan to investigate the generality of our decoder on other sequence-to-tree NLP tasks, such as neural machine translation with tree-structured output [28]. Besides, we will explore how to improve our framework by introducing variational networks, which can be used to capture global context and have been widely used in many NLP tasks [29], [30].

References

[1] R. Balzer, "A 15 year perspective on automatic programming," *IEEE Trans. Softw. Eng.*, vol. SE-11, no. 11, pp. 1257–1268, Nov. 1985.

[2] G. Little and R. C. Miller, "Keyword programming in java," *Autom. Softw. Eng.*, vol. 16, no. 1, pp. 37–71, 2009.

[3] T. Gvero and V. Kuncak, "Interactive synthesis using free-form queries," in *Proc. Int. Conf. Soft. Eng.*, 2015, pp. 689–692.

[4] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Proc. Int. Conf. Learn. Representations*, 2015, pp. 1–15.

[5] W. Ling, P. Blunsom, E. Grefenstette, K. M. Hermann, and A. Senior, "Latent predictor networks for code generation," in *Proc. Annu. Meeting Assoc. Comput. Linguistics*, 2016, pp. 599–609.

[6] L. Dong and M. Lapata, "Language to logical form with neural attention," in *Proc. Annu. Meeting Assoc. Comput. Linguistics*, 2016, pp. 33–43.

[7] P. Yin and G. Neubig, "A syntactic neural model for general-purpose code generation," in *Proc. Annu. Meeting Assoc. Comput. Linguistics*, 2017, pp. 440–450.

[8] M. Rabinovich, M. Stern, and D. Klein, "Abstract syntax networks for code generation and semantic parsing," in *Proc. Annu. Meeting Assoc. Comput. Linguistics*, 2017, pp. 1139–1149.

[9] L. Dong and M. Lapata, "Coarse-to-fine decoding for neural semantic parsing," in *Proc. Annu. Meeting Assoc. Comput. Linguistics*, 2018, pp. 6559–6569.
[10] P. Yin and G. Neubig, "Tranx: A transition-based neural abstract syntax parser for semantic parsing and code generation," in *Proc. Empir. Methods Natural Lang. Process.*, 2018, pp. 7–12.
[11] S. A. Hayati, R. Olivier, P. Avvaru, P. Yin, A. Tomasic, and G. Neubig, "Retrieval-based neural code generation," in *Proc. Empir. Methods Natural Lang. Process.*, 2018, pp. 925–930.
[12] P. Yin and G. Neubig, "Reranking for neural semantic parsing," in *Proc. Annu. Meeting Assoc. Comput. Linguistics*, 2019, pp. 4553–4559.
[13] Z. Sun, Q. Zhu, L. Mou, Y. Xiong, G. Li, and L. Zhang, "A grammar-based structural cnn decoder for code generation," in *Proc. Assoc. Advance. Artif. Intell.*, 2019, pp. 7055–7062.
[14] Z. Sun, Q. Zhu, Y. Xiong, Y. Sun, L. Mou, and L. Zhang, "Treegen: A tree-based transformer architecture for code generation," in *Proc. Assoc. Advance. Artif. Intell.*, 2020, pp. 8984–8991.
[15] B. Wei, G. Li, X. Xia, Z. Fu, and Z. Jin, "Code generation as a dual task of code summarization," in *Proc. Neural Inf. Process. Syst.*, 2019, pp. 6563–6573.
[16] E. C. Shin, M. Allamanis, M. Brockschmidt, and A. Polozov, "Program synthesis and semantic parsing with learned code idioms," in *Proc. Neural Inf. Process. Syst.*, 2019, pp. 10825–10835.
[17] F. F. Xu, Z. Jiang, P. Yin, B. Vasilescu, and G. Neubig, "Incorporating external knowledge through pre-training for natural language to code generation," in *Proc. Annu. Meeting Assoc. Comput. Linguistics*, 2020, pp. 6045–6052.
[18] J. Su, S. Wu, D. Xiong, Y. Lu, X. Han, and B. Zhang, "Variational recurrent neural machine translation," in *Proc. Assoc. Advance. Artif. Intell.*, 2018, pp. 5488–5495.
[19] X. Zhang, J. Su, Y. Qin, Y. Liu, R. Ji, and H. Wang, "Asynchronous bidirectional decoding for neural machine translation," in *Proc. Assoc. Advance. Artif. Intell.*, 2018, pp. 5698–5705.
[20] J. Su, X. Zhang, Q. Lin, Y. Qin, J. Yao, and Y. Liu, "Exploiting reverse target-side contexts for neural machine translation via asynchronous bidirectional decoding," *Artif. Intell.*, vol. 277, 2019, Art no. 103168.
[21] B. Zhang, D. Xiong, J. Su, and J. Luo, "Future-aware knowledge distillation for neural machine translation," *IEEE ACM Trans. Audio Speech Lang. Process.*, pp. 2278–2287, Dec. 2019.
[22] B. Xie *et al.*, "Improving tree-structured decoder training for code generation via mutual learning," in *Proc. Assoc. Advance. Artif. Intell.*, 2021, pp. 14121–14128.
[23] H. Jiang *et al.*, "Exploring dynamic selection of branch expansion orders for code generation," in *Proc. Annu. Meeting Assoc. Comput. Linguistics*, 2021, pp. 5076–5085.
[24] S. Sukhbaatar, E. Grave, P. Bojanowski, and A. Joulin, "Adaptive attention span in transformers," in *Proc. Annu. Meeting Assoc. Comput. Linguistics*, 2019, pp. 331–335.
[25] Y. Oda, H. Fudaba, G. Neubig, H. Hata, and S. Nakamura, "Learning to generate pseudo-code from source code using statistical machine translation (t)," in *Proc. Automated Soft. Eng.*, 2015, pp. 574–584.
[26] P. Yin, B. Deng, E. Chen, B. Vasilescu, and G. Neubig, "Learning to mine aligned code and natural language pairs from stack overflow," in *Proc. Mining Soft. Repositories*, 2018, pp. 476–486.
[27] A. Vaswani *et al.*, "Attention is all you need," in *Proc. Neural Inf. Process. Syst.*, 2017, pp. 5998–6008.
[28] J. Gū, H. S. Shavarani, and A. Sarkar, "Top-down tree structured decoding with syntactic connections for neural machine translation and parsing," in *Proc. Empir. Methods Natural Lang. Process.*, 2018, pp. 401–413.
[29] B. Zhang, D. Xiong, and J. Su, "Variational neural machine translation," in *Proc. Empir. Methods Natural Lang. Process.*, 2016, pp. 521–530.
[30] S. Semeniuta, A. Severyn, and E. Barth, "A hybrid convolutional variational autoencoder for text generation," in *Proc. Empir. Methods Natural Lang. Process.*, 2017, pp. 627–637.

Hui Jiang was born in 1998. He received the bachelor's degree from the Information School, Xiamen University, Xiamen, China. He is currently a Graduate Student with the Information School, Xiamen University. He is supervised by Prof. Jinsong Su. His research interests mainly include natural language processing and code generation.

Linfeng Song received the M.S. degree in computer science from the Institute of Computing Technology, Chinese Academy of Science, Beijing, China, in 2014, and the Ph.D. degree from the Computer Science Department, University of Rochester, Rochester, NY, USA. He is currently a Senior Researcher with Tencent AI Lab, with research interests including dialogue understanding, semantic parsing, and question answering.

Yubin Ge was born in 1994. He received the master's degree from the Information Science University of Pittsburgh, Pittsburgh, PA, USA. He is currently working toward the Ph.D. degree with the University of Illinois Urbana-Champaign, Champaign, IL, USA. His main research interests include natural language processing and human-computer interaction.

Fandong Meng was born in 1988. He received the Ph.D. degree from the Chinese Academy of Sciences, Beijing, China. He is currently a Senior Researcher with Pattern Recognition Center, WeChat AI, Tencent Inc. His research interests include natural language processing and machine translation.

Junfeng Yao received the Ph.D. degree in thermal engineering, specialized in computer simulation from the Central South University, Changsha, China, in 2001. He is currently the Vice Dean of the School of Informatics, Xiamen University, Xiamen, China. He is also a Professor with the Center for Digital Media Computing. His primary research research interests include 3D digital human being, development and application of virtual reality system, intelligent algorithm research, and simulation of the industrial process. From September 2001 to December 2003, he conducted his Postdoctoral research work with Tsinghua University, Beijing, China, in the area of Electrical Simulation and Controlling.

Jinsong Su was born in 1982. He received the Ph.D. degree from the Chinese Academy of Sciences, Beijing, China. He is currently a Professor with Xiamen University, Xiamen, China. His research interests include natural language processing and machine translation.