Stacked AMR Parsing with Silver Data

Qingrong Xia[1], **Zhenghua Li**[1]*, **Rui Wang**[2], **Min Zhang**[1]
[1]Institute of Artificial Intelligence, School of Computer Science and Technology,
Soochow University, China
[2]Vipshop (China) Co., Ltd.
[1]kirosummer.nlp@gmail.com,{zhli13,minzhang}@suda.edu.cn
[2]mars198356@hotmail.com

Abstract

Lacking sufficient human-annotated data is one main challenge for abstract meaning representation (AMR) parsing. To alleviate this problem, previous works usually make use of silver data or pre-trained language models. In particular, one recent seq-to-seq work directly fine-tunes AMR graph sequences on the encoder-decoder pre-trained language model and achieves new state-of-the-art results, outperforming previous works by a large margin. However, it makes the decoding relatively slower. In this work, we investigate alternative approaches to achieve competitive performance at faster speeds. We propose a simplified AMR parser and a pre-training technique for the effective usage of silver data. We conduct extensive experiments on the widely used AMR2.0 dataset and the results demonstrate that our Transformer-based AMR parser achieves the best performance among the seq2graph-based models. Furthermore, with silver data, our model achieves competitive results with the SOTA model, and the speed is an order of magnitude faster. Detailed analyses are conducted to gain more insights into our proposed model and the effectiveness of the pre-training technique.

1 Introduction

Abstract meaning representation (AMR) parsing aims to abstract semantics from a natural language sentence into a rooted, directed, and labeled graph, where the nodes represent concepts and edges represent semantic relations (Banarescu et al., 2013). Figure 1 gives an example.

One main challenge of AMR parsing is the lack of large-scale annotated data, which limits the model representative ability. To alleviate the problem and boost the performance, early works propose to use silver (pseudo) data that are generated from some released AMR parsing models

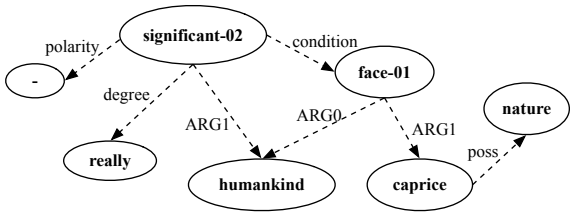

Figure 1: AMR example of the sentence "*Facing the caprice of nature, humankind is really insignificant.*"

(van Noord and Bos, 2017; Konstas et al., 2017). Apart from AMR silver data, Xu et al. (2020) try to use other kinds of large-scale silver data to train a pre-trained model, such as constituent parsing data and machine translation data. With the development of pre-trained language models, recent works try to use pre-trained language models to enhance the model input representative ability (Cai and Lam, 2019; Zhou et al., 2021). Most of the them use pre-trained models in the model encoder side since it naturally provides powerful contextualized representations for sentences. Recently, Bevilacqua et al. (2021) propose a seq2seq AMR parser based on BART (Lewis et al., 2020), which is one encoder-decoder fashion pre-trained language model. They first convert the AMR graph into a text sequence with symbols indicating the concepts' graph positions. Then, they propose to fine-tune the sentence sequence and AMR graph sequence on BART, achieving large improvements compared with previous works, including those with BERT. However, it makes the model relatively slower, which parses 31 tokens per second. We think there are two main reasons: 1) the 12-layer Transformer decoder and 2) the longer converted graph sequences that include the added symbols.

In this work, we investigate alternative approaches to achieve competitive performance at faster speeds. We propose a simplified AMR parser and a pre-training technique for effective use of silver data. First, we propose a simple Transformer-

*Corresponding author.

Findings of the Association for Computational Linguistics: EMNLP 2021, pages 4729–4738
November 7–11, 2021.

based seq2graph AMR parser, denoted as TAMR, that only needs one external bi-affine scorer (Dozat and Manning, 2017) for the relation classification. The remaining question is how we do concept generation and edge classification with Transformer, which is usually used for encoding sequences. Our answer is *giving the Transformer attention mechanisms more meanings*. In detail, we try to demonstrate that the *self-attention* in the decoder captures the semantic relation that can guide establishing the connections for the concepts and the *cross-attention* implicitly links the concept with its surface word, which is similar to the core of attention-based machine translation. Based on the inspirations, we use the copy mechanism (Zhang et al., 2019b; Cai and Lam, 2020) to copy words or lemmas as candidate concepts for concept prediction, in which we treat the *cross-attention* between the encoder and decoder as the probability. Another source of candidate concepts is the extracted concept vocabulary from the training data. For edge classification, we directly treat part of the decoder *self-attention* values as the edge scores between concept nodes.

Second, to achieve competitive performance with the current SOTA model, we seek to use silver data to enhance the model representative ability. Specifically, we employ three different performance AMR models (denoted as "father" models) to generate three different performance silver data and try to investigate several questions which are seldom discussed in previous works: 1) What are the best learning schedules to build pre-trained models with silver data and later fine-tune with the gold-standard data, respectively? 2) Are all the different performance silver data beneficial for our model, even its father model lags behind our model? and 3) Whether using multiple different performance silver data can provide more information than the best performance one or not, i.e., can the higher performance silver data benefits from lower performance silver data? Based on the answers to these questions, which are shown in Section 6.2, we propose a stack pre-training technique for effectively using silver data.

We conduct extensive experiments on the commonly used AMR2.0 dataset. The experimental results show that our proposed model achieves the best results among the seq2graph-based models. Utilizing the silver data, our final model achieves comparable results with the current SOTA model, and the speed is an order of magnitude faster. Our contributions are threefold: (I) We propose a simple Transformer-based AMR parser, which only needs to add one external bi-affine scorer for the relation classification. (II) We investigate how to ensemble different models via the proposed stack pre-training method. (III) Detailed analyses show more insights into our model and several interesting findings of utilizing the silver data.

2 Related Work

AMR parsing approaches can mostly be categorized into four classes: pipeline-based, transition-based, seq2seq-based, and seq2graph-based approaches.

Pipeline-based approaches mainly consist of two steps: 1) concept identification and 2) relation identification. Flanigan et al. (2014) is the first AMR parsing work (JAMR) that treats concept identification as a sequence labeling problem and relation identification as a maximum-scoring connected graph searching problem, in which they also propose an influential rule-based aligner for aligning the concepts and words. Lyu and Titov (2018) treat the alignment as latent variables and propose a joint model for AMR parsing. Zhang et al. (2019a) first use the attention-based copy mechanism to predict concepts in a BiLSTM encoder-decoder framework and then use the bi-affine scorer for edge and relation prediction based on the predicted concepts.

Transition-based methods aim to design a series of actions to generate the AMR graph. Wang et al. (2016) propose to transform the sentence's dependency tree into its AMR graph. Ballesteros and Al-Onaizan (2017); Naseem et al. (2019) use Stack-LSTM transition-based AMR parser that transforms the sentence into the AMR graph, which is different from Wang et al. (2016). With the rise of Transformer, Astudillo et al. (2020); Zhou et al. (2021) propose to use Stack-Transformer for transition-based AMR parsing.

Seq2seq-based approaches convert the AMR graph generation problem into a symbolic sequence generation problem, where the hierarchy structure is converted into human-defined symbols. Konstas et al. (2017) propose a seq2seq-based AMR parser, which uses millions of unlabeled data with self-training. van Noord and Bos (2017) leverage a character level seq2seq-based model and silver data, achieving promising improvements. Recently, Bevilacqua et al. (2021) fine-tune the the gold-

standard data on BART (Lewis et al., 2020), which achieves new SOTA performance.

Seq2graph-based methods generate a new concept node and its connections with previously generated concepts at one time step, thus are relatively faster than seq2seq-based methods. Zhang et al. (2019b) propose a BiLSTM encoder-decoder-based model for several semantic tasks, including AMR. Cai and Lam (2019) present a top-down AMR parser that generates the concept nodes in a root-to-leaf way. Cai and Lam (2020) introduce an iterative inference for the decoding process on the Transformer encoder-decoder architecture. Motivated by the seq2graph methods' generation process and the Transformer encoder-decoder framework, we propose to adapt AMR parsing into the Transformer architecture. The main difference between our model and previous seq2graph models is that our model mostly relies on Transformer, only added one biaffine scorer for relation classification.

3 Methodology

3.1 Task Formulation.

Given one sentence $s = w_1, w_2, ..., w_n$, AMR parsing aims to parse the sentence into an AMR graph $\mathcal{G} = \{\mathcal{N}, \mathcal{E}\}$, where $\mathcal{N} = \{c_1, c_2, ..., c_m\}$ is the set of concept nodes in the AMR graph[1] and $\mathcal{E} = \{(c_i, c_j, r)|1 \leq i \leq m, 1 \leq j \leq m, r \in \mathcal{R}\}$ is the set of edges in the graph. $\mathcal{R}$ is the set of AMR relations.

Overall, our Transformer-based model consists of the following modules, i.e., input layer, encoder layer, decoder layer, concept generator, edge generator, and relation classifier. We will describe the model architecture in detail and show how to adapt the AMR parsing process into Transformer in the following sections.

3.2 Input Layer.

Encoder Input. The model input of each word w_i in the sentence s is composed of its character representation which is generated by a convolutional neural network (CNN) (Kalchbrenner et al., 2014), randomly initialized lemma, part-of-speech tag, named entity tag, and dependency label embeddings (Xia et al., 2019), which is denoted as $\mathbf{f}_i = \mathbf{rep}^{char}_{w_i} \oplus \mathbf{emb}^{lem}_{w_i} \oplus \mathbf{emb}^{PoS}_{w_i} \oplus \mathbf{emb}^{NE}_{w_i} \oplus \mathbf{emb}^{DL}_{w_i}$, where $\oplus$ means the concatenation operation. We also use BERT (Devlin et al., 2019) to enhance the word representation. To get the word-based representations, we make average pooling to sub-word-based representations. And due to the GPU limitation, we fix the BERT model parameters as Zhang et al. (2019b). The final model input representation for w_i is computed as $\mathbf{x}^w_i = \sqrt{dim} * (\text{MLP}(\mathbf{f}_i) + \text{MLP}(\mathbf{rep}^{w_i|s}_{BERT})) + \mathbf{emb}^p_i$, where dim is the embedding dimension and $\mathbf{emb}^p_i$ is the i-th sinusoidal position embedding.

Decoder Input. In the decoder, we use the concatenation of the concept character representation and the randomly initialized concept embedding as the concept representation, denoted as $\mathbf{x}^c_j = \sqrt{dim} * (\text{MLP}(\mathbf{rep}^{char}_{c_j} \oplus \mathbf{emb}^{con}_{c_j})) + \mathbf{emb}^p_j$.

3.3 Encoder Layer.

Given the encoder input representations, we use the Transformer encoder (Vaswani et al., 2017) to encode the sentence. Formally,

$$\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_n = \text{TF}_{enc}(\mathbf{x}^w_1, \mathbf{x}^w_2, ..., \mathbf{x}^w_n), \quad (1)$$

where TF_{enc} means the multi-layer Transformer encoder. As the Transformer has been widely used, we refer our readers to their original paper for the details. The left part of Figure 2 shows the process.

3.4 Decoder.

We will describe the decoder layer, concept generator, edge generator, and relation classifier together in this part for better understanding. In general, given the sentence representation and a start concept node `START`, the decoder needs to generate the AMR graph one concept by one concept. Figure 2 shows the process at the second step.

Decoder Layer. Given the concept node input representations of the decoder, we compute the output representations as $\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_t = \text{TF}_{dec}(\mathbf{x}^c_1, \mathbf{x}^c_2, ..., \mathbf{x}^c_t | \mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_n)$, where TF_{dec} is the multi-layer Transformer decoder.

Concept Generator. Following previous works (Zhang et al., 2019a; Cai and Lam, 2020), our model generates the concept node from two sources, i.e., the concept vocabulary and the source words (or lemmas) in the sentence. First, given the t-th decoder output representation $\mathbf{h}_t$, we employ a MLP to re-encode it for dislodging the irrelevant information (Dozat and Manning, 2017), denoted as $\mathbf{c}_t = \text{MLP}(\mathbf{h}_t)$. Next, we compute the candidate concept probability distribution over the concept vocabulary as:

$$P^{c\_voc} = Softmax(\mathbf{W}^{c\_voc}\mathbf{c}_t + \mathbf{b}^{c\_voc}), \quad (2)$$

[1] We follow the breadth-first-traversal order to determine the index of a concept node in the graph.

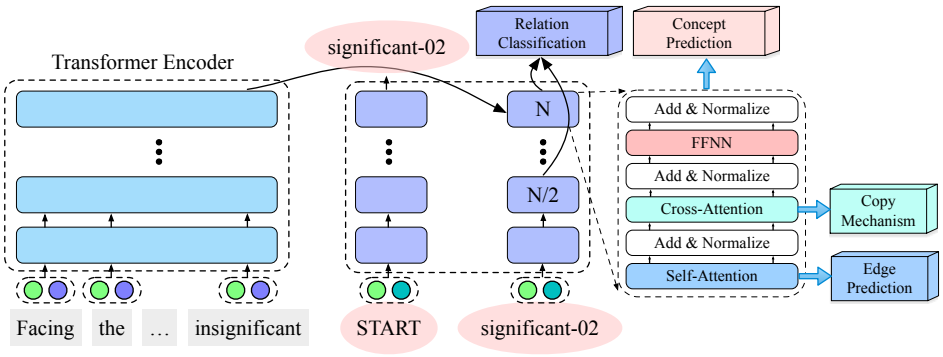

Figure 2: The architecture of our proposed model, which shows the second step of the generation process.

where $\mathbf{W}^{c\_voc}$ and $\mathbf{b}^{c\_voc}$ are linear projection parameters.

Second, we treat the cross-attention (Vaswani et al., 2017) α_t^c as the latent alignment between the currently predicting concept node n_t and the surface words (or lemmas) in the sentence. Based on this assumption, we compute the probability of copying tokens and lemmas of the sentence as follows:

$$P_{tok}^{copy} = \sum_{i:w_i=n_t}^{n} \alpha_t^c[i], \; P_{lem}^{copy} = \sum_{i:l_i=n_t}^{n} \alpha_t^c[i], \quad (3)$$

where w_i means the i-th word, l_i means the i-th lemma, and $[i]$ means indexing the i-th token. The final probability of predicting concept n_t is:

$$P_t^c = \lambda_1 * P^{\,c\_voc} + \lambda_2 * P_{tok}^{copy} + \lambda_3 * P_{lem}^{copy}, \quad (4)$$

where λ_1, λ_2, and λ_3 are normalized weights, which are computed by a single MLP and the softmax function on $\mathbf{c}_t$.

Edge Generator. To connect the current predicted concept node n_t and previously concepts $n_1, n_2, ..., n_{t-1}$, we directly use the self-attention α^s in the decoder. Intuitively, we can treat the self-attention α^s as the relevancy between current node n_t and previous concept nodes. The edge prediction module of Figure 2 shows the workflow. Specifically, we use the self-attention of the upmost decoder layer, which is computed as:

$$\alpha^s = \text{softmax}(\frac{\mathbf{W}^Q\mathbf{h}_t(\mathbf{W}^K\mathbf{h}_{1:t})^\top}{\sqrt{d_k}}). \quad (5)$$

Finally, to determine the edges for the current node and previously generated nodes, we use half of the attention and compute the edge scores as:

$$p_t^e[i] = max_{h=1}^{H/2}\{\alpha_h^s[i]\}, \quad (6)$$

where H is the number of attention heads. If $p_t^e[i] > 0.5$, we connect concept n_t and n_i. Specifically, this strategy allows the current node to attend to all previous nodes with multi connections, which is a crafty way to handle the reentrancy problem.

Relation Classifier. After the current concept node and related edges are generated, the left process is to assign an appropriate label for each edge. In this work, we directly use the bi-affine scorer (Dozat and Manning, 2017) to classify the semantic relation for each concept node pair. Formally, given the predicted concept n_t and any previous concept n_j, we compute the relation scores as,

$$\text{bi-affine}(\mathbf{c}_t, \mathbf{c}_j) = \mathbf{c}_t^\top\mathbf{W}\mathbf{c}_j + \mathbf{U}[\mathbf{c}_t^\top \oplus \mathbf{c}_j] + \mathbf{b}. \quad (7)$$

where $\mathbf{c}_t = \mathbf{h}_t^N$, $\mathbf{c}_j = \mathbf{h}_j^{N/2}$. N is the number of decoder layers. $\mathbf{W}$, $\mathbf{U}$ and $\mathbf{b}$ are learnable parameters. Intuitively, at the i-th step in the decoder, the lower half of decoder layers are used to represent the i-th concept, and the upper half decoder layers are used to represent and predict the $i+1$-th concept. Therefore, we use the $N/2$-th layer representation to represent the other concepts that participate in relation classification.

With the above-described model architecture, TAMR generates the AMR graph one node by one node, until a special ending node `END` generated.

Training & Testing. In training, we employ masked self-attention in the decoder, which ensures each node in the concept sequence can attend to all preceding nodes. For the training objective, our model aims to maximize the log-likelihood of the gold-standard AMR graph given one sentence, which is the sum of the decomposed log-likelihood of each model component.

During testing, we use the beam search method to search the highest-scoring AMR graph.

4 Stack Pre-training with Silver Data

Reviewing the progress of AMR parsing, we can find that the performance boosting with the development of deep learning in the NLP community, especially the usage of silver data and evolution of pre-trained language models like BERT. Injecting BERT representations (Zhang et al., 2019b; Cai and Lam, 2020) into AMR parsing models brings significant improvements compared with previous BERT-free models (Lyu and Titov, 2018; Cai and Lam, 2019), which is an effective way to potentially alleviate the data sparsity problem and enhance model representative ability. However, BERT can only provide powerful representations for the sentence, which can not directly bring benefits to the decoder module in the encoder-decoder framework.

Recently, Bevilacqua et al. (2021) propose a BART-based (Lewis et al., 2020) seq2seq AMR parsing model (SPRING), where the hierarchy structure of AMR graph is represented by human-defined symbols. BART is a pre-trained seq2seq model that provides powerful representations for both encoder and decoder. Thus, Bevilacqua et al. (2021) achieve new SOTA performance on AMR benchmarks by simply fine-tuning BART with gold-standard AMR data. We think the powerful representations of encoder and decoder contribute to the success of Bevilacqua et al. (2021). However, SPRING runs relatively slower than previous seq2graph-based methods because of the auto-regression generation process for the hierarchy symbol-based graph sequence, which needs 26 minutes to test the AMR2.0 test data (about 0.88 sentences/second). Unfortunately, training seq2graph models based on the seq2seq fashion pre-trained models is tricky because BART uses sub-word representations, which is difficult to establish edges for concept nodes. So, can we train a model that has the competitive performance with SPRING and runs as fast as the classic seq2graph models? The answer is yes and we can utilize large-scale silver data (Konstas et al., 2017). In this work, we first employ large-scale silver data to train pre-trained TAMR models for simulating the role of pre-trained language models. Then we fine-tune the pre-trained model with the gold-standard AMR data.

To better understand the effect of silver data and investigate some valid questions that are seldom discussed before, we conduct experiments with silver data that are generated from three different performance models, i.e., JAMR, TAMR, and SPRING. Figure 3 shows the overall process of our usage of these different performance silver data. First, we use the three models to parse the large-scale BLLIP data to generate three different performance silver data. Second, we use the silver data to train three different pre-trained AMR models with TAMR. Third, we fine-tune these pre-trained models with the gold-standard AMR data and investigate whether the model performance can boost or not. Furthermore, we also investigate whether we can utilize all the silver data with some ensemble techniques for further improvements, which can also be seen as an ensemble of different AMR parsing models. To this purpose, we propose a stack pre-training method that progressively learning with these silver data, which is depicted by the bottom workflow in Figure 3. We will discuss these details with the experimental results in Section 6.2.

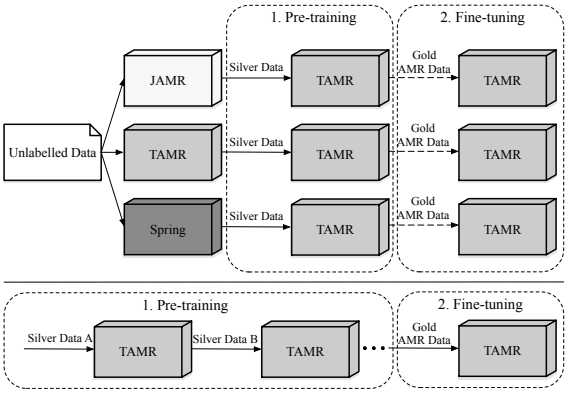

Figure 3: The process of utilizing silver data on TAMR, where the top workflow shows the usage of separate silver data and the bottom workflow shows the usage of ensemble silver data (stacked pre-training).

5 Experiments

5.1 Settings.

Data. We conduct extensive experiments on the commonly used AMR2.0 (LDC2017T10) dataset. Following the standard data split, AMR2.0 contains 36,521, 1,368, and 1,371 samples in the training, development, and test data. BLLIP[2] data is chosen as the large-scale unlabeled data, which contains 1,795,984 sentences that belong to the newswire domain. We implement our model with Pytorch[3].

Hyper-parameters. Table 1 shows the detailed hyper-parameter settings of our model. We use the base-cased version BERT to enhance the sentence

[2]https://catalog.ldc.upenn.edu/LDC2000T43

[3]We release our code, configurations, and silver data at https://github.com/KiroSummer/AMR.

Input Layer	
character	32
lemma	300
PoS tag	32
NER tag	16
dependency label	64
concept	300
BERT	BERT-base-cased
Encoder Layer	
Transformer encoder	4
Decoder Layer	
Transformer decoder	8
Concept Generator	
hidden size	1024
Relation Classifier	
hidden size	100

Table 1: Hyper-parameter settings.

representation and BERT is fixed in our work due to the GPU memory limitation. The encoder and decoder consist of 4 and 8 Transformer blocks, respectively. Each Transformer block has 8 heads, the feed-forward hidden size is 1024, and the hidden size is 512.

Implementation Details. We use Stanford CoreNLP (Manning et al., 2014) for tokenization, lemmatization, part-of-speech tagging, and named entity tagging. The dependency relations are obtained by the bi-affine dependency parser (Dozat and Manning, 2017) implemented in SuPar (Zhang et al., 2020). Previous works (Zhang et al., 2019b; Cai and Lam, 2020) usually use graph recategorization to reduce the complexity and sparseness of the AMR graph. In this work, we use the same script from Cai and Lam (2020) for pre- and post-processing. Our models are trained with Adam (Kingma and Ba, 2015) optimizer and learning rate with warm-up same as to Vaswani et al. (2017). We use the evaluation tool of Damonte et al. (2017) to test our model.

Training Criterion. We train our models for at most 2,020 epochs and choose the best model to evaluate the test data according to the performance on development data.

5.2 Experimental Results.

Results of TAMR. Table 2 shows the experimental results of our base model TAMR and comparison with previous graph-based models. We can see that our model achieves slight improvements over the previous best seq2graph-based model of Cai and Lam (2020) on Smatch score. Besides, it is interesting that our model achieves a better unlabeled Smatch score than Cai and Lam (2020) with +0.7. Cai and Lam (2020) conduct edge prediction based on an iterative state, while we directly model the edge information on the decoder self-attention.

Methods	Smatch	Unlabeled-Smatch
Zhang et al. (2019a)	76.3	–
Zhang et al. (2019b)	77.0	80.0
Cai and Lam (2020)	80.2	82.8
TAMR	**80.3**	**83.5**

Table 2: Smatch scores of our model TAMR and comparison with previous seq2graph-based models on AMR2.0 test data.

	Silver Data		Pre-train		Fine-tune	
	Dev	Test	Dev	Test	Dev	Test
JAMR	–	67.0*	70.5	70.8	80.8	81.0
TAMR	80.6	80.3	81.5	81.2	82.4	82.2
SPRING	–	83.8	83.0	82.7	83.8	83.7

Table 3: Smatch scores of our pre-trained models with separate silver data and fine-tuning models. "Silver Data" shows the original model performance. * JAMR is evaluated on LDC2015E86, which contains 16,833 training sentences and the same development and test data with AMR2.0.

Results of TAMR with Separate Silver Data. Table 3 shows the experimental results of our pre-trained models with separate silver data and the corresponding fine-tuning results on the AMR2.0 dataset. The "Silver Data" column shows the silver data performance, i.e., the original model performance (training with the AMR2.0 training data set). The "Pre-train" column shows the results of the three pre-trained models with the separate silver data. We define the pre-trained model with specific silver data d as TAMR_d^{pre} and fine-tuned model with specific silver data d as TAMR_d^{ft}. For example, the pre-trained model with JAMR silver data is denoted as $\text{TAMR}_{\text{JAMR}}^{pre}$. We can see one interesting finding that pre-trained models $\text{TAMR}_{\text{JAMR}}^{pre}$ and $\text{TAMR}_{\text{TAMR}}^{pre}$ both outperform their original models, while $\text{TAMR}_{\text{SPRING}}^{pre}$ does not. We think this is mainly because the average annotator vs. inter-annotator agreement (Smatch) is 83.0 (Banarescu et al., 2013). Thus, the SPRING silver data can bring a limited benefit of Smatch score around 83.0. Besides, we can see that fine-tuning the pre-trained

Pre-train Methods	Pre-train		Fine-tune	
	Dev	Test	Dev	Test
TAMR	81.5	81.2	82.4	82.2
SPRING	83.0	82.7	83.8	83.7
→JAMR→TAMR	81.9	81.9	82.4	82.2
→JAMR→SPRING	83.2	83.0	84.0	83.9
→TAMR→SPRING	83.3	83.0	84.3	84.0
→JAMR→TAMR→SPRING	83.4	**83.2**	84.3	**84.2**

Table 4: Smatch scores of our ensemble pre-training models on AMR2.0 development and test data with stack pre-training, where "→A→B" means we stack pre-training with silver data A and then silver data B.

models with gold-standard AMR data can consistently improve a lot, even on $\text{TAMR}^{pre}_{\text{SPRING}}$, which already outperforms our base TAMR model. This indicates the effectiveness of the silver data for AMR parsing. There is another interesting point that the fine-tuned $\text{TAMR}^{ft}_{\text{SPRING}}$ did not outperform the pre-trained model of $\text{TAMR}^{pre}_{\text{SPRING}}$, which indicates the upper-bound of utilizing silver data?

Results of TAMR with Stacking Silver Data. Previous works (van Noord and Bos, 2017; Zhou et al., 2021) usually use one single silver data for improving AMR parsing performance. We think it is interesting to explore the effects of different performance silver data, which is seldom discussed before. In this work, we propose a stacking pre-training approach, i.e., pre-training different silver data from low-performance silver data to high-performance silver data one by one. Table 4 shows the results of stack pre-training experiments. First, we can see that stack pre-training TAMR silver data on $\text{TAMR}^{pre}_{\text{JAMR}}$ can bring slight improvement of +0.4 Smatch score (81.9-81.5=0.4), indicating the usefulness of JAMR silver data even though it is generated from a relatively lower performance model. The improvements are consistent in all the combinations, demonstrating the effectiveness of our proposed stack pre-training. Second, we can observe two other interesting findings of the fine-tuning results: 1) Fine-tuning on $\text{TAMR}^{pre}_{\text{TAMR}}$ and $\text{TAMR}^{pre}_{\text{JAMR}\rightarrow\text{TAMR}}$ with the gold-standard AMR data achieve the same result of 82.4, even though their pre-trained models differ. We think this is because TAMR silver data comes from TAMR model, which can not provide additional valid information. 2) Fine-tuning the $\text{TAMR}^{pre}_{*\rightarrow\text{SPRING}}$ pre-trained models achieve promising improvements over $\text{TAMR}^{pre}_{\text{SPRING}}$. This demonstrates an interesting conclusion that lower performance silver data that comes from different sources can still provide beneficial information, which is also effective for higher performance silver data. So, we break the "upper-bound" with the stack pre-training method. In addition, we also try to generate silver data with the best performed model of $\text{TAMR}^{ft}_{\text{JAMR}\rightarrow\text{TAMR}\rightarrow\text{SPRING}}$ and use it in the pre-training process for another iteration. The pre-training step reaches 83.9 Smatch score on the development data, outperforming $\text{TAMR}^{pre}_{\text{JAMR}\rightarrow\text{TAMR}\rightarrow\text{SPRING}}$ by +0.5 Smatch score. However, compared with the 84.3 Smatch score of $\text{TAMR}^{ft}_{\text{JAMR}\rightarrow\text{TAMR}\rightarrow\text{SPRING}}$, the succedent fine-tuning process didn't bring further improvement in our current settings. We think our proposed stack pre-training technique has sufficiently tapped the potential of the silver data and there is little space for further iterations.

Layer Num.	Smatch	Unlabeled-Smatch
4	79.8	82.6
6	80.1	83.1
8	**80.6**	**83.6**

Table 5: Results of different number of decoder layers on the AMR2.0 development data.

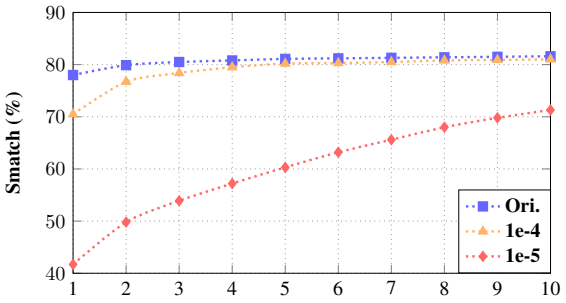

Figure 4: Convergence curves (Smatch score vs. training epochs) of pre-training TAMR silver data on AMR2.0 development data set.

6 Analysis

6.1 Effect of TAMR Decoder Layers.

The decoder plays the main role of TAMR, in which the number of layers matters a lot. Table 5 shows the results. We can see that with the increasing of decoder layers, the performance improves accordingly. The 8-layer decoder achieves the best results on the development and test data, respectively. However, we didn't increase more layers because of the GPU memory limitation.

6.2 Effect of Methods with Silver Data.

In order to find a better pre-training and fine-tuning pipeline strategy, we conduct detailed experiments.

	1e-5	5e-5	Ori.
JAMR	80.8	80.8	80.9
TAMR	82.4	82.3	81.9
SPRING	83.8	83.8	83.3

Table 6: Results of fine-tuning with different learning rates on AMR2.0 development data set.

	AVG_P	AVG_S	SP
{JAMR, TAMR}	28.7	82.2	82.4
{JAMR, SPRING}	16.9	83.0	84.0
{TAMR, SPRING}	48.6	83.5	84.3
{JAMR, TAMR, SPRING}	17.4	83.2	84.3

Table 7: Results of different ensemble techniques on AMR2.0 development data. "AVG_P" means averaging the model parameters of fine-tuned specific models, "AVG_S" means averaging the scores of last decision layers of different models, and "SP" means stack pre-training.

Model	Speed (Tokens/Second)
SPRING	31
TAMR	300

Table 8: Speed comparison with previous works.

Analyses of Pre-training. Konstas et al. (2017) propose to pre-train model with a fixed learning rate of 1e-5. In this work, we compare two learning rate strategies with the TAMR silver data: 1) training with the base model learning rate strategy (Ori.) and 2) training with fixed learning rate of 1e-3, 1e-4, and 1e-5. The convergence curves are shown in Figure 4. We can see that the original learning rate strategy achieves the best results. Besides, using the fixed learning rate of 1e-3 makes the model crashed, which only achieves 0.16 Smatch score on the AMR2.0 development data set.

Analyses of Fine-tuning. Given the pre-trained models, how to effectively fine-tune with the gold data is also a valid problem to discuss. In this work, we investigate three learning rate settings. Table 6 shows the results. We can see that 1) Fine-tuning with a small learning rate can achieve promising results, in which 1e-5 is slightly better than 5e-5. 2) Using the original learning rate strategy can achieve better results on $\text{TAMR}^{pre}_{\text{JAMR}}$, but lags behind in the other two pre-trained models. Thus, we use the fixed learning rate of 1e-5 in our experiments.

Analyses of Ensemble Techniques. In order to find a better way to use multiple silver data, we compare with the wildly used model ensemble technique of averaging model parameters (Zhou et al., 2021), averaging the last decision layer scores, and our proposed stack pre-training technique. Table 7 shows the results on different model combinations. It is surprising that the averaging method crashed in all the combinations. We think this is because these models' pre-trained models differ a lot on their representative space due to the different performance silver data. The averaging scores of the decision layer method achieves reasonable results, but cannot outperform the better one of the merged models. Our proposed stack pre-training technique achieves consistently best results in all combinations. Therefore, we use our proposed stack pre-training methods for our ensemble experiments. Besides, we experiment with different kinds of combinations and finally found that stacking high-performance silver data on low-performance silver data benefits more. From the observations, we think our proposed stack pre-training technique can also be applied into other models and other NLP tasks which have limited human annotated data.

6.3 Speed Comparison.

Table 8 shows the speed comparison between our model and SPRING. To evaluate the AMR2.0 test data, our TAMR needs 2min41s, while SPRING needs more time of 20min2s. We think there are two main reasons for the relatively low speed of SPRING. First, the BART backbone has a huge number of parameters to compute in the decoder. Second, the resulting AMR graph text sequence contains a lot of external symbols, making the number of decoding steps increased a lot. While our seq2graph model TAMR, it has a relatively small decoder and only need to generate the concepts and the connected relations at each step.

6.4 Final Results.

Table 9 shows the final results of our model and comparison with previous works. We can find that the progress of pre-trained language models influences the development of AMR parsing. Before the rise of seq2seq pre-trained language models, the BERT/RoBERTa-based seq2graph and transition approaches achieve the best results. With the seq2seq pre-trained language models, Bevilacqua et al. (2021) achieve amazing improvements compared with previous seq2seq models, resulting in new SOTA result of 84.5 Smatch score. Our final model that uses stack pre-training technique with multiple silver data achieves comparable results to

	Model	Pre-LM	G.R.	Smatch	Fine-grained Results							
					Unlabeled	No WSD	Concept	SRL	Reent.	Neg.	NER	wiki
Pipe.	Zhang et al. (2019a)	-	Y	74.6	-	-	-	-	-	-	-	-
	Zhang et al. (2019a)	BERT	Y	76.3	79.0	76.8	84.8	69.7	60.0	75.2	77.9	85.8
Tran.	Naseem et al. (2019)	BERT	Y	75.5	80	76	86	72	56	67	83	80
	Zhou et al. (2021)	RoBERTa	N	81.8	85.5	82.3	88.7	80.8	71.1	69.7	88.5	78.8
	Zhou et al. (2021)†	RoBERTa	N	83.4	-	-	-	-	-	-	-	-
S2S.	Xu et al. (2020)	-	N	71.5	-	-	-	-	-	-	-	-
	Xu et al. (2020)†	-	N	80.2	83.7	80.8	87.4	78.9	66.5	71.5	85.4	75.1
	Bevilacqua et al. (2021)	BART	N	83.8	86.1	84.4	90.2	79.6	70.8	74.4	**90.6**	84.3
	Bevilacqua et al. (2021)	BART	Y	**84.5**	**86.7**	**84.9**	89.6	**79.7**	**72.3**	79.9	83.7	**87.3**
S2G.	Cai and Lam (2020)	-	Y	77.3	80.1	77.9	86.4	69.4	58.5	76.5	78.4	86.1
	Cai and Lam (2020)	BERT	Y	80.2	82.8	80.8	88.1	74.2	64.6	78.9	81.1	86.3
	TAMR	BERT	Y	80.3	83.5	80.8	88.6	73.7	63.3	77.6	80.8	86.6
	TAMR†	BERT	Y	84.2	86.5	84.6	**90.6**	78.9	70.2	**81.4**	83.8	87.0

Table 9: Smatch scores of our final models and comparison with previous works on AMR2.0 test data. "Pipe.", "Trans.", "S2S", and "S2G." represent the pipeline-based, transition-based, seq2seq-based, and seq2graph-based methods, respectively. G.R. means using graph re-categorization and † means using silver data.

Bevilacqua et al. (2021), yet has a faster speed.

7 Conclusion

In this work, we propose a simple Transformer-based AMR parsing model that adapts AMR parsing into the Transformer architecture, in which the attention mechanisms are given more meanings of latent alignment and the connections between concepts. Based on our proposed model, we conduct detailed experiments to investigate several strategies for using silver data and propose an effective stack pre-training method for ensemble different models. The experimental results show that our proposed model achieves the best performance of seq2graph-based models and demonstrate the effectiveness of using silver data. Our final model achieves comparable results with the SOTA model, and the speed is an order of magnitude faster. Detailed analyses show the effectiveness of our proposed ensemble technique of stack pre-training for using multiple silver data.

Acknowledgments

We thank our anonymous reviewers for their helpful comments. This work was supported by the National Natural Science Foundation of China (Grant No. 62176173 and 61876116), and a Project Funded by the Priority Academic Program Development (PAPD) of Jiangsu Higher Education Institutions.

References

Ramón Fernandez Astudillo, Miguel Ballesteros, Tahira Naseem, Austin Blodgett, and Radu Florian. 2020. Transition-based parsing with stack-transformers. In *Proceedings of EMNLP: Findings*, pages 1001–1007.

Miguel Ballesteros and Yaser Al-Onaizan. 2017. Amr parsing using stack-lstms. In *Proceedings of EMNLP*, pages 1269–1275.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract meaning representation for sembanking. In *Proceedings of the 7th linguistic annotation workshop and interoperability with discourse*, pages 178–186.

Michele Bevilacqua, Rexhina Blloshmi, and Roberto Navigli. 2021. One SPRING to rule them both: Symmetric AMR semantic parsing and generation without a complex pipeline. In *Proceedings of AAAI*.

Deng Cai and Wai Lam. 2019. Core semantic first: A top-down approach for amr parsing. In *Proceedings of EMNLP-IJCNLP*, pages 3790–3800.

Deng Cai and Wai Lam. 2020. Amr parsing via graph-sequence iterative inference. In *Proceedings of ACL*, pages 1290–1301.

Marco Damonte, Shay B. Cohen, and Giorgio Satta. 2017. An incremental parser for Abstract Meaning Representation. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 536–546.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186.

Timothy Dozat and Christopher D Manning. 2017. Deep biaffine attention for neural dependency parsing. In *Proceedings of ICIR*.

Jeffrey Flanigan, Sam Thomson, Jaime G Carbonell, Chris Dyer, and Noah A Smith. 2014. A discriminative graph-based parser for the abstract meaning representation. In *Proceedings of ACL*, pages 1426–1436.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A convolutional neural network for modelling sentences. In *Proceedings of ACL*, pages 655–665.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proceedings of ICLR*.

Ioannis Konstas, Srinivasan Iyer, Mark Yatskar, Yejin Choi, and Luke Zettlemoyer. 2017. Neural amr: Sequence-to-sequence models for parsing and generation. In *Proceedings of ACL*, pages 146–157.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of ACL*, pages 7871–7880.

Chunchuan Lyu and Ivan Titov. 2018. Amr parsing as graph prediction with latent alignment. In *Proceedings of ACL*, pages 397–407.

Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of ACL: System Demonstrations*, pages 55–60.

Tahira Naseem, Abhishek Shah, Hui Wan, Radu Florian, Salim Roukos, and Miguel Ballesteros. 2019. Rewarding smatch: Transition-based amr parsing with reinforcement learning. In *Proceedings of ACL*, pages 4586–4592.

Rik van Noord and Johan Bos. 2017. Neural semantic parsing by character-based translation: Experiments with abstract meaning representations. *Computational Linguistics in the Netherlands Journal*, 7:93–108.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*, 30:5998–6008.

Chuan Wang, Sameer Pradhan, Xiaoman Pan, Heng Ji, and Nianwen Xue. 2016. Camr at semeval-2016 task 8: An extended transition-based amr parser. In *Proceedings of the 10th international workshop on semantic evaluation (semeval-2016)*, pages 1173–1178.

Qingrong Xia, Zhenghua Li, Min Zhang, Meishan Zhang, Guohong Fu, Rui Wang, and Luo Si. 2019. Syntax-aware neural semantic role labeling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 7305–7313.

Dongqin Xu, Junhui Li, Muhua Zhu, Min Zhang, and Guodong Zhou. 2020. Improving amr parsing with sequence-to-sequence pre-training. In *Proceedings of EMNLP*, pages 2501–2511.

Sheng Zhang, Xutai Ma, Kevin Duh, and Benjamin Van Durme. 2019a. Amr parsing as sequence-to-graph transduction. In *Proceedings of ACL*, pages 80–94.

Sheng Zhang, Xutai Ma, Kevin Duh, and Benjamin Van Durme. 2019b. Broad-coverage semantic parsing as transduction. In *Proceedings of EMNLP-IJCNLP*, pages 3777–3789.

Yu Zhang, Zhenghua Li, and Zhang Min. 2020. Efficient second-order TreeCRF for neural dependency parsing. In *Proceedings of ACL*, pages 3295–3305.

Jiawei Zhou, Tahira Naseem, Ramón Fernandez Astudillo, and Radu Florian. 2021. Amr parsing with action-pointer transformer. In *Proceedings of NAACL-HLT*.